# INF-253 Lenguajes de Programación
# Tarea 4: Scheme

29 de mayo de 2023

## 1. Objetivo

En esta tarea deberán implementar una serie de funciones para conocer y aplicar correctamente los conceptos y técnicas del paradigma de programación funcional, utilizando el lenguaje **Scheme**.

## 2. Funciones a implementar

1. **Encode**
   - **Sinopsis**: `(encode bits)`
   - **Característica Funcional**: Listas simples y funciones puras
   - **Descripción**: Se le entrega una lista de 0s y 1s (`bits`), debe retornar una lista de números con el tamaño de todos los bloques que contienen los mismos valores de bits. Se debe considerar que la `bits` siempre parte con un bloque de 0s (si el primer bit es 1, entonces el primer numero de la lista retornar debe ser 0 ya que el primer bloque esta vació).
   - **Ejemplos**:
     ```
     >(encode '(0 0 0 1 1 0 1 1 1 0 0 0))
     (3 2 1 3 3)
     >(encode '(1 1 0 1 1 1 0 0 0))
     (0 2 1 3 3)
     >(encode '(1 0 1 0))
     (0 1 1 1 1)
     >(encode '(0 0 0 0))
     (4)
     ```
2. **Decode**
   - **Sinopsis**: `(decode_simple lista)` `(decode_cola lista)`
   - **Característica Funcional**: Listas simples, recursión simple y recursión cola
   - **Descripción**: Se le entrega una lista de números (`lista`) producida utilizando la funcion anterior (`encode`). Debe retornar la lista de 0s y 1s codificada por esta secuencia, es decir `(decode (encode (bits)))` deberia ser siempre igual a `(bits)`.

     Esta función debe ser implementada de dos formas, la primera `(decode_simple lista)` debe realizar recursión de simple y la segunda `(decode_cola lista)` debe realizar recursión de cola.

- **Ejemplos**:

```
>(decode '(3 2 1 3 3))
(0 0 0 1 1 0 1 1 1 0 0 0)
>(decode '(0 2 1 3 3))
(1 1 0 1 1 1 0 0 0)
>(decode '(0 1 1 1 1))
(1 0 1 0)
>(decode '(4))
(0 0 0 0)
```

3. **Integrar**

   - **Sinopsis**: `(integrar_simple a b n f) (integrar_cola a b n f)`
   - **Característica Funcional**: Funciones lambda, recursion simple y recursion de cola
   - **Descripción**: Se le entregan tres números (`a`, `b` y `n`) y una función lambda (`f`). Debe calcular la integral de `f` en el intervalo `[a, b]` utilizando la siguiente aproximacion:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{n}\left(\frac{f(a)}{2} + \frac{f(b)}{2} + \sum_{k=1}^{n-1}\left(f\left(a + k\frac{b-a}{n}\right)\right)\right)$$

     Esta función debe ser implementada de dos formas, la primera `(integrar_simple a b n f)` debe realizar recursión de simple y la segunda `(integrar_cola a b n f)` debe realizar recursión de cola.

   - **Ejemplos**:

```
>(integrar 0 1 4 (lambda (x) (* x x))
0.315975
>(integrar 1 10 100 (lambda (x) (/ (log x) (log 2)))
20.235025
```

4. **Map en arbol binario**

   - **Sinopsis**: `(map_arbol arbol camino f)`
   - **Característica Funcional**: Manejo de listas, funciones lambda y recursion
   - **Descripción**: Se le entrega la raíz de un árbol binario (`arbol`), una lista de 0s y 1s (`camino`) y una función lambda (`f`). Deberá retornar el árbol entregado a la función, pero en donde a todos los nodos que se encuentren en el camino se les aplicara la función `f`. El camino es una secuencia de 0s y 1s que indica si se debe continuar con el nodo izquierdo (0) o el nodo derecho (1). Los nodos de un árbol binario se representaran como una lista de 3 elementos: `'(Valor_del_nodo Nodo_izquierdo Nodo_derecho)`. Si un nodo no tiene uno o ambos de sus hijos, entonces tendrá una lista vacía en esa posición.
   - **Ejemplos**:

```
>(map_arbol '(2 (3 () ()) (4 () ())) '(1) (lambda (x) (* x x))
(4 (3 () ()) (16 () ()))
>(map_arbol '(2 (3 () ()) (4 (3 (5 () ()) ()) (3 (8 () ()) ()))) '(1 0
0) (lambda (x) (* x x))
(4 (3 () ()) (16 (9 (25 () ()) ()) (3 (8 () ()) ())))
>(map_arbol '(2 (3 () ()) (4 (3 (5 () ()) ()) (3 (8 () ()) ()))) '(0) (lambda
(x) (* x x))
(4 (9 () ()) (4 (3 (5 () ()) ()) (3 (8 () ()) ())))
```

# 3. Sobre la Entrega

- Se deberá entregar un único archivo con todas las funciones implementadas en el orden descrito en el enunciado.
- Se debe programar siguiendo el paradigma de la programación funcional, no realizar códigos que siguen el paradigma imperativo. Por ejemplo, se prohíbe el uso de `for-each`.
- Para implementar las funciones utilice DrRacket.
  - http://racket-lang.org/download/
- Todo código debe contener al principio `#lang scheme`
- Se debe entregar un archivo con extensión**.rkt**
- Pueden crear funciones que no estén especificadas para utilizar en los problemas planteados, pero solo se revisará que la función pedida funcione y el problema este resuelto con la característica funcional planteada en el enunciado.
- Cuidado con el orden y la indentación de su tarea, llevará descuento de lo más 20 puntos.
- Las funciones implementadas y que no estén en el enunciado deben ser comentadas de la siguiente forma. **SE HARÁN DESCUENTOS POR FUNCIÓN NO COMENTADA**

```
;; Descripcion de la funcion
;;
;; a: Descripcion del parametro a
;; b: Descripcion del parametro a
```

- Se debe trabajar de forma individual obligatoriamente.
- **La entrega debe entregarse en .tar.gz y debe llevar el nombre: Tarea4LP_RolAlumno.tar.gz**
- **El archivo README.txt debe contener nombre y rol del alumno e instrucciones detalladas para la correcta utilización de su programa. De no incluir README se realizara un descuento.**
- La entrega será vía aula y el plazo máximo de entrega es hasta **el viernes 9 de Junio a las 23:55**.
- Las copias serán evaluadas con nota 0 y se informarán a las respectivas autoridades.
- **Solo se contestaran dudas realizadas en AULA y que se realicen al menos 48 horas antes de la fecha de entrega original.**

# 4. Calificación

## 4.1. Entrega mínima

La entrega mínima corresponde a implementar correctamente la función `encode` y al menos una de las formas de recursion en la función `decode`. Esto corresponderá a 30pts.

## 4.2. Entrega

- Entrega mínima (**30 pts**)
- `decode` (Implementar ambas) (**10 pts**)
- `integrar` (**25 pts**)
  - No hace correcto uso de funciones lambda (**max 0 pts**)
  - Hace correcto uso de funciones lambda, pero no utiliza ninguna forma de recursion (**max 7 pts**)
  - Hace correcto uso de funciones lambda, pero solo utiliza uno de los tipos de recursion (**max 15 pts**)
  - Hace correcto uso de funciones lambda y utiliza correctamente ambos tipos de recursion (**max 25 pts**)
- `map_arbol` (**35 pts**)
  - No hace correcto uso de recursion y funciones lambda (**max 0 pts**)
  - Hace correcto uso de recursion y funciones lambda, pero no recorre el árbol de forma correcta (**max 10 pts**)
  - Hace correcto uso de recursion y funciones lambda, pero no produce resultados correctos para todos los casos (**max 25 pts**)
  - Hace correcto uso de recursion y funciones lambda y entrega resultados correctos para todos los inputs validos (**max 35 pts**)

## 4.3. Descuentos

- Falta de comentarios (-5 pts c/u Max 20 pts)
- Falta de README (-20 pts)
- Falta de alguna información obligatoria en el README (-5 pts c/u)
- Falta de orden (entre -5 y -20 pts dependiendo de que tan desordenado)
- Entrega tardía (-20 pts por día, -10 pts dentro de la primera hora)
- Mal nombre en algún archivo entregado (-5 pts c/u)