Lenguajes de Programación
Auxiliar N°1

Profesor: Éric Tanter
Auxiliar: Ismael Figueroa

Para cada función defina su contrato y sus tests
Programación en Scheme, en particular **recursión** y **funciones de orden superior**.

1. Implemente la función `fibs` que calcule el k-ésimo número de Fibonacci. Ejemplo:

```
> (fibs 4)
2
```

2. Implemente la función `rango` que genera una lista con los números enteros dado su valor inicial y final, ambos incluidos. Asuma que el valor inicial es menor o igual al valor final. En caso contrario retorne una lista vacía. Ejemplo:

```
> (rango 10 15)
'(10 11 12 13 14 15)
```

3. Implemente la función `lista-fibs` que retorne la lista de los primeros k números de Fibonacci. Ejemplo:

```
> (lista-fibs 5)
'(0 1 1 2 3)
```

4. Implemente la función `cuadrado` que retorna el cuadrado de un número.

5. Implemente la función `lista-cuadrados` que dada una lista de números retorna una lista con los cuadrados de dichos números. Ejemplo:

```
> (lista-cuadrados '(2 3 8 9 10))
'(4 9 64 81 100)
```

6. Implemente la función `transformar-lista`[1] que dada una lista y una función de transformación retorna una nueva lista con los elementos transformados, uno a uno y en el mismo orden de la lista original. Ejemplo:

```
> (define (suma1 x) (+ x 1))
> (transformar-lista suma1 '(1 2 3 4))
'(2 3 4 5)
> (transformar-lista (lambda (x) (+ x 2)) '(1 2 3 4))
'(3 4 5 6)
```

   Redefina `lista-fibs` y `lista-cuadrados` usando `transformar-lista`

[1] Esta función se conoce como `map`

7. Implemente la función `suma-lista` que sume los elementos de una lista. Asuma que la lista tendrá sólo números. En caso de lista vacía el valor será cero. Ejemplo:

```
> (suma-lista '(2 3 8 213))
226
```

8. Implemente la función `suma-fibs-cuadrado`, para los k números de Fibonacci. En caso de lista vacía el valor será cero.

```
> (suma-fibs-cuadrado 5)
15
```

9. Implemente la función `suma` que generalice la suma de elementos de una lista, con la aplicación de un transformador, y redefina suma-lista y suma-fibs-cuadrado usando `suma`. La idea, en términos matemáticos es:

$$suma = f(l, g) = \sum_{x \in l} g(x)$$

10. `suma` es aún un caso particular de una función llamada `foldl` que encadena aplicaciones de una función binaria. Su definición es:

```
(define (foldl fun value l)
  (cond
    ((empty? l) value)
    (else (foldl fun (fun value (car l)) (cdr l)))))
```

Defina `suma` en función de `foldl`.[2].

11. Implemente la función `filtrar-pares` que dada una lista de números, retorna una lista que respeta el orden de la lista original pero sólo contiene los números pares. Ejemplo:

```
> (filtrar-pares '(1 2 38 4 29 0 8))
'(2 38 4 0 8)
```

12. Implemente la función `filtrar-string` que dada una lista de elementos arbitrarios, retorna una lista que respeta el orden de la lista original pero sólo contiene strings. Ejemplo:

```
> (filtrar-string '(1 2 "asdasf" 'symbolo (1 2 "abacaba") "pecera"))
'("asdasf" "pecera")
```

13. Implemente la función `filtrar` que generaliza `filtrar-pares` y `filtrar-string`, y redefina estas dos funciones usando `filtrar`.

14. Implemente Merge para listas de números, utilizando recursión, filtros y concatenación de listas. Use el elemento $\lfloor \frac{largo-lista}{2} \rfloor$ como pivote. Hint: estudie las funciones `take` y `drop` ya incorporadas en Scheme. Ejemplo:

```
> (mergesort-num '(21 3 0 -23 832 29 2 1))
'(-23 0 1 2 3 21 29 832)
```

15. Haga una versión general de Mergesort que funcione como la anterior pero además acepte como argumento una función de comparación para el ordenamiento de los elementos. Redefina el Mergesort para números con esta nueva abstracción.

[2]La l es porque procesa comenzando por la izquierda, asímismo *foldr* comienza por la derecha. Vea http://en.wikipedia.org/wiki/Fold_%28higher-order_function%29 para una buena explicación gráfica