Ejercicios Scheme

Para cada función escriba su contrato, descripción, ejemplos y los tests necesarios, antes de su implementación.

1 Para soltar la mano

1. Represente la siguiente expresión en Scheme: $\frac{5+4+(2-(3-(6+\frac{4}{5})))}{3(6-2)(2-7)}$
2. Defina una función que tome como argumento tres números y retorne la suma de los cuadrados de los dos números mayores
3. El siguiente programa pretende sumar el valor de a, con el valor absoluto de b.¿Es correcto el programa? Argumente.

```
(define (a-suma-abs-b a b)
  ((if (> b 0) + -) a b))
```

4. Implemente la función factorial que calcula el factorial de un entero no negativo n.
5. Implemente la función fibs que calcula el k-ésimo número de Fibonacci.
6. Implemente la función my-expt que recibe dos enteros como argumento, el segundo siempre no negativo, y calcula el primero a la potencia del segundo.[1]
7. Implemente el algoritmo de Euclides para obtener el máximo común divisor entre dos enteros positivos
8. Implemente una función que determina si un entero positivo dado es o no primo. Retorne #t en el caso afirmativo, y #f en otro caso.[2]
9. Implemente la función suma-cuadrados que calcule la suma del cuadrado de los primeros k naturales
10. Implemente la función suma-cos que calcule la suma del coseno de los primeros k naturales

[1] Scheme trae incorporada la función *expt*

[2] Una función o expresión que retorna #t o #f es llamada *predicado*. Cualquier valor distinto de #f es considerado como verdadero. Una convención sintáctica de Scheme es finalizar el nombre de un predicado con el símbolo ?. Por ejemplo *primo?*, *null?, empty?*

11. Implemente la función suma-abs que sea una abstracción del proceso de sumar, y reimplemente suma-cuadrados y suma-cos

12. Refine la abstracción de suma-abs para que permita calcular una suma sobre un intervalo arbitrario de enteros positivos, luego redefina suma-abs, suma-cuadrados y suma-cos utilizando esta nueva abstracción

13. Refine la abstracción del ejercicio 12, de manera de que el incremento no sea siempre unitario. Por ejemplo, para sumar sobre todos los números pares , hasta cierta cota superior, se iniciaría el intervalo en 2 y el incremento sería 2

14. Implemente la función Celcius->Farenheit que convierte de grados Celsius a Farenheit

15. Implemente la función Dolar->Peso que convierte de dólares a pesos.

16. Implemente la función Euro->Peso un conversor de euros a pesos.

17. Implemente una abstracción para la conversión de unidades y reimplemente las funciones de los ejercicios 14, 15 y 16 utilizando dicha abstracción.

2 Listas

1. Implemente la función my-list-ref que retorne el k-ésimo elemento de una lista. ¿Qué debería pasar si se pide un elemento mayor al largo de la lista?[3]

2. Implemente la función my-reverse que dada una lista, construya una con los elementos en orden inverso[4]

3. Implemente la función my-take que dada una lista y un entero n, retorne una lista con los primeros n elementos de la lista original[5]

4. Implemente la función my-drop que dada una lista y un entero n, retorne una lista con todos los elementos, excepto los primeros n de la lista original[6]

5. Implemente la función rango que dados dos enteros, retorne la sublista que contiene los elementos correspondientes al intervalo dado por estos enteros (Hint: puede usar my-take y my-drop para esto)

6. Implemente la función my-append, que dadas dos listas, retorne una lista con la concatenación de los elementos de estas listas[7]

[3] Corresponde a la función *list-ref* de Scheme
[4] Corresponde a la función *reverse* de Scheme
[5] Corresponde a la función *take* de Scheme
[6] Corresponde a la función *drop* de Scheme
[7] Corresponde a la función *append* de Scheme

7. Implemente la función list-sub que, dada una lista de números, retorne la resta sucesiva de sus elementos. Ej.: (list-sub '(1 2 3 4)) debe evaluarse como -8

8. Implemente la función list-add que, dada una lista de números, retorne la suma sucesiva de sus elementos. Ej.: (list-add '(1 2 3 4)) debe evaluarse como 10

9. Implemente la función list-mult que, dada una lista de números, retorne el producto de sus elementos.

10. Implemente la función my-fold que sea una abstracción de list-sub, list-add y list-mult, y reimplemente estas funciones utilizando my-fold

11. Implemente la función list-square que, dada una lista de números, retorne una lista con los cuadrados de cada elemento. Ej.: (list-square '(1 2 3 4) debe evaluarse en (1 4 9 16)

12. Implemente la función list-halve que, dada una lista de números, retorne una lista con la mitad de cada element. Ej.: (list-halve '(1.0 2.0 3.0 4.0)) debe evaluarse en (0.5 1.0 1.5 2.0)

13. Implemente la función my-map, que sea una abstracción de list-square y list-halve. Luego reimplemente estas funciones utilizando my-map.[8]

¿Qué resultado obtiene si el parámetro es (1 2 3 4) en vez de (1.0 2.0 3.0 4.0)? ¿Puede explicar esto?

[8] Corresponde a la función *map* de Scheme