Problemario: Programación funcional, parte 1

Instrucciones

Utilizando el lenguaje de programación funcional indicado por tu profesor (Scheme, Racket, Clojure), resuelve los problemas que se presentan a continuación. Coloca tu código en un solo archivo. Cada función debe incluir un comentario con una breve descripción de lo que hace.

1. La función `fahrenheit-to-celsius` toma como entrada una temperatura `f` en grados Fahrenheit y la convierte a su equivalente en grados Celsius usando la siguiente fórmula:

$$C = \frac{5(F - 32)}{9}$$

   Ejemplos:

```
(fahrenheit-to-celsius 212.0)
⇒ 100.0

(fahrenheit-to-celsius 32.0)
⇒ 0.0

(fahrenheit-to-celsius -40.0)
⇒ -40.0
```

2. La función `sign` recibe como entrada un valor entero `n`. Devuelve -1 si `n` es negativo, 1 si `n` es positivo mayor que cero, o 0 si `n` es cero.

   Ejemplos:

```
(sign -5)
⇒ -1

(sign 10)
⇒ 1

(sign 0)
⇒ 0
```

3. La función `roots` devuelve la raíz que resuelve una ecuación cuadrática a partir de sus tres coeficiente, `a`, `b` y `c`, que se reciben como entrada. Se debe usar la siguiente fórmula:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

   Ejemplos:

```
(roots 2 4 2)
⇒ -1

(roots 1 0 0)
⇒ 0

(roots 4 5 1)
⇒ -1/4
```

4. El índice de masa corporal, o BMI por sus siglas in inglés, se utiliza para determinar si la proporción de peso y altura de una persona es adecuada. El BMI se calcula usando la siguiente fórmula:

$$BMI = \frac{weight}{height^2}$$

Donde *weight* es el peso en kilogramos y *height* es la altura en metros. La siguiente tabla muestra cómo se clasifican los diferentes rangos del BMI:

Rango BMI	***Descripción***
$BMI < 20$	`underweight`
$20 \leq BMI < 25$	`normal`
$25 \leq BMI < 30$	`obese1`
$30 \leq BMI < 40$	`obese2`
$40 \leq BMI$	`obese3`

La función `bmi` recibe dos entrada: `weight` y `height`. Debe devolver un símbolo que represente la descripción del BMI correspondiente calculado a partir de sus entradas.

Ejemplos:

```
(bmi 45 1.7)
⇒ underweight

(bmi 55 1.5)
⇒ normal

(bmi 76 1.7)
⇒ obese1

(bmi 81 1.6)
⇒
obese2

(bmi 120 1.6)
⇒ obese3
```

5. La función `factorial` toma un entero positivo `n` como su entrada y devuelve el factorial correspondiente, que matemáticamente se define así:

$$n! = \begin{cases} 1 & \text{Si } n = 0 \\ n \cdot (n-1)! & \text{Si } n > 0 \end{cases}$$

Ejemplos:

```
(factorial 0)
⇒ 1

(factorial 5)
⇒ 120

(factorial 40)
⇒ 815915283247897734345611269596115894272000000000
```

6. La función `duplicate` toma una lista `lst` como entrada y devuelve una nueva lista en donde cada elemento de `lst` está duplicado.

   Ejemplos:

   ```
   (duplicate '())
   ⇒ ()

   (duplicate '(1 2 3 4 5))
   ⇒ (1 1 2 2 3 3 4 4 5 5)

   (duplicate '(a b c d e f g h))
   ⇒ (a a b b c c d d e e f f g g h h)
   ```

7. La función `pow` toma dos entradas como entrada: un número `a` y un entero positivo `b`. Devuelve el resultado de calcular `a` elevado a la potencia `b`.

   Ejemplos:

   ```
   (pow 5 0)
   ⇒ 1

   (pow -5 3)
   ⇒ -125

   (pow 15 12)
   ⇒ 129746337890625
   ```

8. La función `fib` toma un entero positivo `n` como entrada y devuelve el elemento correspondiente de la secuencia de Fibonacci, que se define matemáticamente como:

$$\text{fib}(n) = \begin{cases} n & \text{Si } n \leq 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{Si } n > 1 \end{cases}$$

   Ejemplos:

   ```
   (fib 6)
   ⇒ 8

   (map fib (range 10))
   ⇒
   (0 1 1 2 3 5 8 13 21 34)

   (fib 42)
   ⇒ 267914296
   ```

9. La función `enlist` coloca dento de una lista a cada elemento de nivel superior de la lista que recibe como entrada.

   Ejemplos:

   ```
   (enlist '())
   ⇒ ()

   (enlist '(a b c))
   ⇒ ((a) (b) (c))

   (enlist '((1 2 3) 4 (5) 7 8))
   ⇒ (((1 2 3)) (4) ((5)) (7) (8))
   ```

10. La función `positives` toma una lista de números `lst` como entrada y devuelve una nueva lista que solo contiene los números positivos de `lst`.

    Ejemplos:

    ```
    (positives '())
    ⇒ ()

    (positives '(12 -4 3 -1 -10 -13 6 -5))
    ⇒ '(12 3 6)

    (positives '(-4 -1 -10 -13 -5))
    ⇒ ()
    ```

11. La función `add-list` devuelve la suma de los números contenidos en la lista que recibe como entrada, o 0 si está vacía.

    Ejemplos:

    ```
    (add-list '())
    ⇒ 0

    (add-list '(2 4 1 3))
    ⇒ 10

    (add-list '(1 2 3 4 5 6 7 8 9 10))
    ⇒ 55
    ```

12. La función `invert-pairs` toma como entrada una lista cuyo contenido son listas de dos elementos. Devuelve una nueva lista con cada pareja invertida.

    Ejemplos:

    ```
    (invert-pairs '())
    ⇒ ()

    (invert-pairs '((a 1)(a 2)(b 1)(b 2)))
    ⇒ ((1 a)(2 a)(1 b)(2 b))

    (invert-pairs '((January 1)(February 2)(March 3)))
    ⇒ ((1 January)(2 February)(3 March))
    ```

13. La función de `list-of-symbols?` toma una lista `lst` como entrada. Devuelve *verdadero* si todos los elementos (posiblemente cero) contenidos en `lst` son símbolos, o *falso* en caso contrario.

    Ejemplos:

    ```
    (list-of-symbols? '())
    ⇒ #t

    (list-of-symbols? '(a b c d e))
    ⇒ #t

    (list-of-symbols? '(a b c d 42 e))
    ⇒ #f
    ```

14. El función `swapper` toma tres entradas: dos valores `a` y `b`, y una lista `lst`. Devuelve una nueva lista en la que cada ocurrencia de `a` en `lst` se intercambia por `b`, y viceversa. Cualquier otro elemento de `lst` permanece igual. Se puede suponer que `lst` no contiene listas anidadas.

    Ejemplos:

    ```
    (swapper 1 2 '())
    ⇒ ()

    (swapper 1 2 '(4 4 5 2 4 8 2 5 6 4 5 1 9 5 9 9 1 2 2 4))
    ⇒ (4 4 5 1 4 8 1 5 6 4 5 2 9 5 9 9 2 1 1 4)

    (swapper 1 2 '(4 3 4 9 9 3 3 3 9 9 7 9 3 7 8 7 8 4 5 6))
    ⇒ (4 3 4 9 9 3 3 3 9 9 7 9 3 7 8 7 8 4 5 6)

    (swapper 'purr
             'kitty
             '(soft kitty warm kitty little ball
               of fur happy kitty sleepy kitty
               purr purr purr))))
    ⇒ (soft purr warm purr little ball of fur happy purr sleepy purr kitty kitty kitty)
    ```

15. La función `dot-product` toma dos entradas: las listas `a` y `b`. Devuelve el resultado de realizar el producto punto de `a` por `b`. El producto punto es una operación algebraica que toma dos secuencias de números de igual longitud y devuelve un solo número que se obtiene multiplicando los elementos en la misma posición y luego sumando esos productos. Su fórmula es:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

    Ejemplos:

    ```
    (dot-product '() '())
    ⇒ 0

    (dot-product '(1 2 3) '(4 5 6))
    ⇒ 32

    (dot-product '(1.3 3.4 5.7 9.5 10.4)
                 '(-4.5 3.0 1.5 0.9 0.0))
    ⇒ 21.45
    ```

16. La función `average` recibe una lista de números `lst` como entrada. Devuelve la *media aritmética* de los elementos contenidos en `lst`, o 0 si `lst` está vacía. La media aritmética ($\bar{x}$) se define como:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Ejemplos:

```
(average '())
⇒ 0

(average '(4))
⇒ 4

(average '(5 6 1 6 0 1 2))
⇒ 3

(average '(1.7 4.5 0 2.0 3.4 5 2.5 2.2 1.2))
⇒ 2.5
```

17. La función `standard-deviation` recibe una lista de números `lst` como entrada. Devuelve la *desviación estándar* de la población de los elementos contenidos en `lst`, o 0 si `lst` está vacía. La desviación estándar de la población (σ) se define como:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Ejemplos:

```
(standard-deviation '())
⇒ 0

(standard-deviation '(4 8 15 16 23 42))
⇒ 12.3153

(standard-deviation '(110 105 90 100 95))
⇒ 7.07106

(standard-deviation '(9 2 5 4 12 7 8 11 9 3 7 4 12 5 4 10 9 6 9 4))
⇒ 2.983
```

18. La función `replic` toma dos entradas: una lista `lst` y un número entero `n`, donde `n` ≥ 0. Devuelve una nueva lista que replica `n` veces cada elemento contenido en `lst`.

Ejemplos:

```
(replic 7 '())
⇒ ()

⇒ (replic 0 '(a b c))
()

(replic 3 '(a))
⇒ (a a a)

(replic 4 '(1 2 3 4))
⇒ '(1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4)
```

19. La función `expand` toma una lista `lst` como entrada. Devuelve una lista donde el primer elemento de `lst` aparece una vez, el segundo elemento aparece dos veces, el tercer elemento aparece tres veces, y así sucesivamente.

    Ejemplos:

    ```
    (expand '())
    ⇒ ()

    (expand '(a))
    ⇒ (a)

    (expand '(1 2 3 4))
    ⇒ (1 2 2 3 3 3 4 4 4 4)

    (expand '(a b c d e))
    ⇒ (a b b c c c d d d d e e e e e)
    ```

20. La función `binary` recibe un entero `n` como entrada ($n \geq 0$). Si `n` es igual a cero, devuelve una lista vacía. Si `n` es mayor que cero, devuelve una lista con una secuencia de unos y ceros equivalente a la representación binaria de `n`.

    Ejemplos:

    ```
    (binary 0)
    ⇒ ()

    (binary 30)
    ⇒ (1 1 1 1 0)

    (binary 45123)
    ⇒ (1 0 1 1 0 0 0 0 0 1 0 0 0 0 1 1)
    ```