Guía de Trabajos Prácticos V – Programación Funcional

1. Convierta las siguientes expresiones aritméticas en expresiones Scheme y evalúelas:
   - 7 + (2 * -1/3) + −10.7
   - (7/3 * 5/9) ÷ (5/8 − 2/3)
   - 1 + 3 ÷ (2 + 1 ÷ (5 + 1/2))
   - 1 × −2 × 3 × −4 × 5 × −6 × 7

2. Determine el valor de las siguientes expresiones. Use el DrScheme para verificar su respuesta

   - (cons 'car '+)
   - (list 'esto '(es muy fácil))
   - (cons 'pero '(se está complicando...))
   - (cons '(y ahora no se que) 'hizo)
   - (quote (+ 7 2))
   - (cons '+ '(10 3))
   - (car '(+ 10 3))
   - (cdr '(+ 10 3))
   - cons
   - (quote (cons (car (cdr (7 4)))))
   - (quote cons)
   - (car (quote (quote cons)))
   - (+ 2 3)
   - (+ '2 '3)
   - (+ (car '(2 3)) (car (cdr '(2 3))))
   - ((car (list + - * /)) 2 3)

3. Reescriba las siguientes expresiones, usando Let para remover las subexpresiones comunes y para mejorar la estructura del código. No realice ninguna simplificación algebraica.

```
a. (+ (/ (* 7 a) b) (/ (* 3 a) b) (/ (* 7 a) b))

b. (cons (car (list a b c)) (cdr (list a b c)))
```

4. Determine el valor de la siguiente expresión. Explique como ha derivado este valor.

```
(let ((x 9))
  (* x
     (let ((x (/ x 3)))
       (+ x x))))
```

5. Reescriba las siguientes expresiones para darles un único nombre a cada variable let-bound diferente de forma que ninguna quede *ensombrecida*. Verifique que el valor de su expresión es el mismo que el de la expresión original.

```
a. (let ((x 'a) (y 'b))
     (list (let ((x 'c)) (cons x y))
           (let ((y 'd)) (cons x y))))
b. (let ((x '(a b) c)))
     (cons (let ((x (cdr x)))
             (car x))
           (let ((x (car x)))
             (cons (let ((x (cdr x)))
                     (car x))
                   (cons (let ((x (car x)))
                           x)
                         (cdr x))))))
```

6. Determine el valor de las siguientes expresiones.

```
a. (let ((f (lambda (x) x)))
     (f 'a))
b. (let ((f (lambda x x)))
     (f 'a))
c. (let ((f (lambda (x . y) x)))
     (f 'a))
d. (let ((f (lambda (x . y) y)))
     (f 'a))
```

7. El procedimiento **length** retorna la longitud de su argumento, que debe ser una lista. Por ejemplo, **(length '(a b c))** es 3. Usando **length**, defina el procedimiento **shorter**, que retorna la lista más corta de los dos argumentos pasados o la primera lista si tienen el mismo largo.

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

8. Defina una función que devuelva el área de un circulo.

   ```
   Ej: (area 3) → 28.26
   ```

9. Defina una función que devuelva el perímetro de un circulo.

   ```
   Ej: (perímetro 1) → 6.28
   ```

10. Defina una función llamada **circlestuff** que compute el área y el perímetro de un círculo y los devuelva en una lista, con el área primero.

    ```
    Ej: (circlestuff 3) → (28.26 18.84)
    ```

11. Defina una función **distance2d** que reciba como parámetros dos puntos en el plano y devuelva su distancia. Utilice una lista impropia para la declaración de x e y.

    ```
    Ej: (distance2d '(1 . 1) '(2 . 2)) → 1.41
    ```

12. Defina una función **próximo** que reciba un punto en el plano y una lista da puntos y devuelva la distancia al más cercano.