Guia de Trabajos Prácticos
Paradigma Funcional

1) Convierta las siguientes expresiones aritméticas en expresiones DrRacket 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 DrRacket 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) Describa los pasos necesarios para evaluar la siguiente expresión:

```
((car (cdr (cdr (list + - * /)))) 5 5)
```

4) Obtenga el elemento x de las siguientes listas:

```
'( a b c . x)
```

```
'(a b c x)
```

(cdr (car '((a.x) b)))

```
'((a . x) b)
```

(car '(x.a))

```
'(x . a)
```

(cdr '(a.x))

```
'(a . x)
```

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

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

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

6) Determine el valor de la siguiente expresión. Explique cómo ha derivado este valor.

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

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 `mascorta`, que retorna la lista más corta de los dos argumentos pasados o la primera lista si tienen el mismo largo.

```
(mascorta '(a b) '(c d e)) → (a b)
```

```
(mascorta '(a b) '(c d)) → (a b)
```

```
(mascorta '(a b) '(c)) → (c)
```

8) Defina una función que devuelva el área de un círculo. Ej: `(area 3)` → `28.26`

9) 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`

10) Escriba una función `largo` que devuelva el largo de una lista sin utilizar la función definida en Racket.
Ej: `(largo '(1 4 8))` → `3`

11) Escriba una función que cuente la cantidad de apariciones de un elemento en una lista. El primer parámetro será el elemento a buscar y el segundo la lista en la que se debe buscar.
Ej: `(count-elem 3 '(1 2 3 4 5 4 3 2 1 ))` → `2`

12) Defina un procedimiento `subst` que reciba tres parámetros (dos valores y una lista) y devuelva la lista con todos los componentes que son iguales al primer parámetro reemplazados por el valor del segundo parámetro.
Ej: `(subst 'c 'k '( c o c o n u t))` → `(k o k o n u t)`

13) Se desea crear un función que reciba como parámetros una lista de átomos compuesto únicamente de letras y devuelva una lista agrupando los que son iguales en sublistas.
Ej: `(agrupar '(A A B C A B A D C))` → `((AAAA) (BB) (CC) (D))`

14) Escriba en Racket el procedimiento (concatenar l1 l2) que recibe dos listas l1 y l2 como argumento y retorna una única lista que contiene primero todos los elementos de l1 seguido de todos los elementos de l2 (no puede usar el procedimiento interno append que viene en algunas implementaciones de Racket).
Ej: `(concatenar '(1 2 3) '(4 5 6))` → `(1 2 3 4 5 6)`

15) Defina en Racket un procedimiento recursivo que encuentre el primer elemento de una lista que es un número. Debe retornar el número si lo encuentra, sino retornar null.

    Ej: `(primer-num '((1 . 2) 'a (b) (5) 6 8 'a 9)` → `6`

16) Defina un procedimiento en Racket llamado attach-at-end que reciba como parámetro un valor y una lista y retorne la lista con los mismos valores excepto el que se pasó por parámetro que se agregará al final.

    Ej: `(attach-at-end 'prueba '(esto es una))` → `(esto es una prueba)`

17) Se desea crear un programa que permita convertir un lote de datos de un formato a otro. Los datos llegan en formato de lista de listas, donde el primer elemento determina el contenido de la lista y el segundo tiene la lista de datos. Los datos pueden venir en formato texto, decimal o booleano y se desea obtener una lista igual pero con todos sus componentes en formato decimal y todos positivos.

```
(convdatos '(
("D" (1 2 3 4 5))
("T" ("6" "7" "8"))
("B" ("V" "F"))
)
```

    da como resultado:

```
'((1 2 3 4 5) (6 7 8) (1 0))
```

18) Definir una estructura que represente un punto en el plano. Crear una función que calcule la distancia entre dos puntos datos recibiendo como parámetros la estructura de cada uno.

19) Ordenar por Peso ASCII

    Se desea crear una función que reciba como parámetro una lista de strings y devuelva una lista con las cadenas ordenadas por su peso ASCII.
    Ej: `(ordenar '("moto" "auto" "casa" "juego" "aire"))` → `("casa" "aire" "auto" "moto" "juego")`

    El peso ascii de una palabra se calculará como la suma de los valores de cada uno de los caracteres que la componen.

    Ejemplo:

    `(pesopalabra "casa")` → `408`

    Ya que `a = 97`, `c = 99`, y `s = 115`. Entonces: `99 + 97 + 115 + 97 = 408`.

20) Cree una función llamada `fullreverse-list` que permite revertir completamente el contenido de una lista.

    `(fullreverse-list (1 (2 3 4 (4 5) (3 (5 6)) 4)))` → `(4 ((6 5) 3) (5 4) 4 3 2) 1)`

21) Utilizando la función MAP cree una función llamada `fullreverse-list` que permite revertir completamente el contenido de una lista.

22) Cree una función `app2list` que una dos elementos y los devuelve siempre como lista. Si los elementos son listas las junta, si es un elemento y una lista lo agrega y si son dos elementos crea una lista con ellos.