**TC2006 – Lenguajes de Programación***Funciones de Orden Superior en Racket*

Matrícula1\_\_\_\_\_\_\_\_\_\_ Matrícula2 \_\_\_\_\_\_\_\_\_\_\_ Matrícula3 \_\_\_\_\_\_\_\_\_\_\_

1. Programar **SIN usar primitivos de orden superior** (como el map, apply, etc.)
   1. Programar la función de orden superior *recursiva* **filtra** que elimine de una lista, que se le dé como primer argumento, todos los elementos que no cumplan con un predicado unario, que se le dé como segundo argumento.

Probar con:

> (filtra ‘(1 -2 -3 4 -5) negative?) **=> (-2 -3 -5)**

> (filtra ‘(1 -2 -3 4 -5) (lambda (x) (x > -3))) **=> (1 -2 4)**

* 1. Programar la función de orden superior **compon** que reciba dos funciones unarias (f y g) como argumentos, y regrese una función que aplique la composición de funciones f(g(x)) a un valor x.

Probar con:

> ((compon sqrt abs) -4) **=> 2**

> ((compon car cdr) ‘(1 2 3)) **=> 2**

* 1. Programar la función de orden superior **agregaAlFinal** que toma un elemento e y regresa una función que toma una lista y agrega e al final de la lista.

Probar con:

> ((agregaAlFinal ‘a) ‘(2 3 4)) => **(2 3 4 a)**

> ((agregaAlFinal ‘(3 4)) ‘(1 2)) => **(1 2 3 4)**

1. Programar **USANDO los primitivos de orden superior MAP y APPLY**, pero SIN utilizar recursividad explícita
   1. Programar la función **impares** que elimine todos los elementos que no sean impares de una lista de sublistas.

Probar con:

> (impares ‘((1 2 3)(4 5 6))) **=> ((1 3)(5))**

> (impares ‘((2 2)(2 2)(2 2))) **=> (()()())**

* 1. Programar el predicado **impares?** que determine si hay elementos impares dentro de una lista de sublistas.

Probar con:

> (impares? ‘((1 2 3)(4 5 6))) **=> #t**

> (impares? ‘((2 2)(2 2)(2 2))) **=> #f**

* 1. Programar la función **empareja** que genere la lista de pares que se puede generar con los elementos de una lista plana.

Probar con:

> (empareja ‘(1)) **=> ((1 1))**

> (empareja ‘(1 2)) **=> ((1 1)(1 2)(2 1)(2 2))**