# TC2006 Lenguajes de Programación

## Tarea 3: Programación Básica en Racket

Fecha de entrega: martes 17 de marzo de 2020

(* Equipos de 2 integrantes *)

En esta tarea comenzarán a practicar con el lenguaje Racket. Solo uno de los integrantes del equipo deberá las soluciones a los problemas como asignación de Blackboard. Un archivo texto con nombre **M_tarea3.rkt** que contenga el código de las funciones implementadas para cada problema y donde mediante comentarios de Racket deberán incluir sus matrículas y nombres en el archivo, así como la descripción de lo que calcula cada función. Y un documento **M_tarea3.pdf** con la respuesta al problema 4. Deberán sustituir la M de los nombres de los archivos con sus matrículas separadas por guiones bajos (_).

1. Implementar el predicado **par-menor** que reciba cinco argumentos enteros no negativos y *despliegue* los dos valores menores de los 5 argumentos separados por un guión. Los valores pueden estar repetidos. **Restricción**: NO UTILIZAR LISTAS, de lo contrario no se contará.
   Probar con:
   ```
   > (par-menor 1 8 3 4 5)     => 1-3
   > (par-menor 1 0 1 0 1)     => 0-0
   ```

2. Implementar la función recursiva **logaritmo** que regrese el valor del logaritmo de y=1+x mediante el cálculo de n términos de la siguiente serie:
   $$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots + (-1)^{n-1}\frac{x^n}{n}, \qquad -1 < x \leq 1$$
   Probar con:
   ```
   > (logaritmo 0.3 1)    => 0.3
   > (logaritmo -0.5 4)   => -0.6822916
   ```

3. Implementar la función recursiva **secuencias** que *despliegue* N (1er argumento) secuencias alternadas de los enteros del 1 al M (2do argumento) primero en orden ascendente y luego descendente.
   Probar con:
   ```
   > (secuencias 4 5)
   => 1 2 3 4 5
      5 4 3 2 1
      1 2 3 4 5
      5 4 3 2 1
   ```

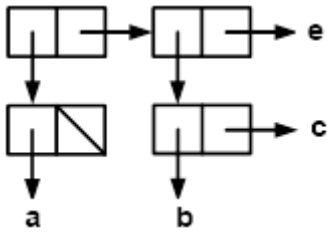
4. Para cada uno de los siguientes incisos obtén las dos representaciones faltantes de las tres vistas en clase: representación visual, representación interna, y

representación con el constructor **cons** (no se permiten otros constructores como *list* o *append*).

a. `((1 (2 . 3). 4) 5)`

b. `(cons (cons (cons 'a 'b)(cons (cons 'c '()) '())) 'd)`

c.




5. Implementar la función recursiva **repite** que, dada una lista de enteros no negativos, regrese una lista donde cada valor se repita el número de veces que representa.
   Probar con:
   ```
   > (repite '(0 1 2 3))        => (1 2 2 3 3 3)
   > (repite '(4 0 3 0 2))      => (4 4 4 4 3 3 3 2 2)
   ```

6. Implementar la función recursiva **contadores** que a partir de una lista de posiciones, dadas como enteros positivos, cree una lista de contadores donde cada valor en la lista represente la cantidad de veces que aparece esa posición en la lista de posiciones. La posición mayor determinará el tamaño de la lista de contadores.
   Probar con:
   ```
   > (contadores '(1 2 3 4 5 6))    => (1 1 1 1 1 1)
   > (contadores '(2 2 4 2 4 2))    => (0 4 0 2)
   > (contadores '(6 8 3 6 6 1))    => (1 0 1 0 0 3 0 1)
   ```

7. En Racket se puede manejar una cantidad variable de argumentos separando el último argumento (parámetro variable) con un punto, por ejemplo (define (procedimiento . args) …), donde args resultará en una lista conteniendo los argumentos dados.
   Implementar la función recursiva **enteros** que regrese la cantidad total de enteros que se encuentre en una secuencia arbitraria de listas planas que contienen enteros y símbolos.
   Probar con:
   ```
   > (enteros '(1 2 3 4 5 6))                 => 6
   > (enteros '(a b) '(c d))                  => 0
   > (enteros '(1 a) '(2 3) '(b c) '(9 d))    => 4
   ```

8. Implementar la función recursiva **forma** que dada una lista plana y dos enteros positivos N y M, regrese una lista con N sublistas que contienen M elementos cada una. Si la lista plana no contiene NxM elementos, los

elementos faltantes deberán aparecer como guiónes (-) y si la lista plana tiene más de NxM elementos, los restantes deberán desecharse.
Probar con:

```
> (forma '(1 2 3 4 5 6) 2 3)     => ((1 2 3)(4 5 6))
> (forma '(1 2 3 4 5 6) 4 2)     => ((1 2)(3 4)(5 6)(- -))
> (forma '(1 2 3 4 5 6) 1 4)     => ((1 2 3 4))
```

9. Implementar la función recursiva **enumera** que dada una lista posiblemente imbricada, regrese una lista con la misma forma, pero que en lugar de cada valor original regrese un número que indique su profundidad y su posición en la (sub)lista donde se encuentra.
   Probar con:

```
> (enumera '(a b c))   => (1.1 1.2 1.3)
> (enumera '(3 (b (c 2 (d)) a) 1))
   => (1.1 (2.1 (3.1 3.2 (4.1)) 2.3) 1.3)
```