Junior S. Hernández Jimenez **ID:20180999**

L1 Práctica con Racket

En esta primera práctica te pido que resuelvas cinco pequeños problemas

1.- Problema de números primos simples.

En clase hemos mostrado como en un lenguaje funcional se resuelve determinar si un número es primo. En ese caso usamos un cierto número de funciones auxiliares fuera del algoritmo principal

Seguirás ese camino probando cada función.

pero continuarás generando una lista de enteros primos en un rango que escojas

Solución:

```
(display "Solucion a Problema 1. Numeros primos simples")
(newline)

(define (square x ) ( * x x))

(define (smallest-divisor n)
  (find-divisor n 2))

(define (find-divisor n test-divisor)
  (cond ((> (square test-divisor) n) n)
        ((divides? test-divisor n) test-divisor)
        (else (find-divisor n (+ test-divisor 1)))))

(define (divides? a b)
  (= (remainder b a) 0))

(define (prime? n)
  (cond [(= n (smallest-divisor n)) n]
        [else null]
   ))

(define (rango n m)
    (if (> n m) null
        (cons n (rango(+ n 1) m))
  )
 )

(filter (lambda (n) (not (null? n))) (map prime? (rango 5 17)))
```

```
Welcome to DrRacket, version 7.9 [3m].
Language: racket/base, with debugging; memory limit: 128 MB.
Solucion a Problema 1. Numeros primos simples
'(5 7 11 13 17)
```

2.- Vamos a continuar trabajando con números primos, pero esta vez utilizaremos a Fermat para hallar tales números.

Fermat establece que un número p es primo si para entero a desde 2 hasta p - 1 el resto de a^(p -1) mod p es igual a 1. Lógicamente no se escogen todos los valores a, antes mencionados, sino sólo unos pocos escogidos al azar. Una vez realizado tal algoritmo. Generarás una lista de primos como en el caso anterior.

```
#lang racket/base

(define (square x) (* x x))

(define (expmod base exp m)
  (cond ((= exp 0) 1)
        ((even? exp)
         (remainder (square (expmod base (/ exp 2) m))
                    m))
        (else
         (remainder (* base (expmod base (- exp 1) m))
                    m))))

(define (fermat n)
  (define (test a)
    (= (expmod a n n) a))
  (define (iter a)
    (if (< a n)
        (if (test a)
            (iter (+ a 1))
            null)
        n))
  (iter 1))

(define (range-help cur goal lst)
  (cond
    [(= cur goal) lst]
    [else (range-help (+ 1 cur) goal (append lst (list cur)))]))

(define(range num)
  (range-help 0 num '()))

(filter (lambda (n) (not (null? n))) (map fermat (range 100000)))
```

Solución:

```
Welcome to DrRacket, version 7.9 [3m].
Language: racket/base, with debugging; memory limit: 512 MB.
' (0
   1
   2
   3
   5
   7
   11
   13
   17
   19
```

9883
9887
9901
9907
9923
9929
9931
9941
9949
9967
9973)

3.- Vamos a probar el sistema de repartición de curules según el método Dhondt.
Se tienen los resultados de las votaciones de n partidos políticos y la cantidad de cargos eleccionarios (curules).

Dhondt determina como se reparten las curules.
Sea Sp la cantidad de asientos asignados a cada partido p. Inicialmente 0
Sea Vp la cantidad de votos de cada partido p.
Se ordenan la votación por número de votos.
Entonces se procede así con el partido de mayor votación.
Se obtiene Qp = Vp/(Sp+1)
Sp = Sp + 1
Vp = Qp
Ya tenemos un nuevo ordenamiento, y se procede de igual manera con el partido que tiene ahora mayor votación.
Hasta que todos los asientos sean asignados.

Ejemplo Se tienen cuatro partidos (A b C D) a repartirse 8 asientos.

ronda	1	2	3	4	5	6	7	8
A votos	100,000	50,000	50,000	33,333	33,333	25,000	25,000	25,000
Asientos	1		2		3		4	
B votos	80,000	80,000	40,000	40,000	26,667	26,667	26,667	20,000
Asientos	0	1		2				
C votos	30,000	30,000	30,000	30,000	30,000	30,000	15,000	15,000
Asientos	0				1			
D votos	20,000	20,000	20,000	20,000	20,000	20,000	20,000	20,000
Asientos	0							

Solución:

```
(display "Solucion a Problema 3. El método Dhondt")
(newline)

(define partidos (list 100000 80000 30000 20000))


(define (highest-num lst)
  (define (iter lst accu)
    (cond ((null? lst) accu)
          ((< accu (car lst)) (iter (cdr lst) (+ accu (- (car lst) accu))))
          (else (iter (cdr lst) accu))))
  (iter lst 0))

(define (find-smallest l)
  (let loop ((l l)
             (sm (car l)))
    (cond
      [(null? l) sm]
      [(< sm (car l))
       (loop (cdr l) sm)]
      [else
       (loop (cdr l) (car l))])))

(define (Qp Vp Sp)
  (cond [(= Vp (highest-num partidos)) (/ Vp (+ Sp 1))]
        [(and(< Vp (highest-num partidos)) (> Vp (find-smallest partidos)))  Vp]
        [else (/ Vp (+ Sp 1))]
  )
 )


(define (rondas q)
    (define (loop i)
      (if (= i (- num-rondas 1))
          '()
          (cons (Qp q (+ i 1)) (loop (+ i 1)))))
    (loop -1)
)

(define (highm lst)
  (define (loop i)
    (if (= i 4)
          '()
          (cons (list-ref lst i) (loop (+ i 1)))))
  (loop 0)
)

(define num-rondas 8)
(map rondas (highm partidos))
```

Welcome to DrRacket, version 7.9 [3m].
Language: racket/base, with debugging; memory limit: 128 MB.
Solucion a Problema 3. El método Dhondt

```
'((100000 50000 33333 1/3 25000 20000 16666 2/3 14285 5/7 12500)
  (80000 80000 80000 80000 80000 80000 80000 80000)
  (30000 30000 30000 30000 30000 30000 30000 30000)
  (20000 10000 6666 2/3 5000 4000 3333 1/3 2857 1/7 2500))
```

En lo que elaboramos un reglamento te pido que
En un único documento pdf que depositaras en esta area de la pva me presentarás
a.- El enunciado de cada problema
b.- Tu solución programática (agregando los comentarios de lugar)
c.- Las corridas (plural) de lugar