Lenguajes de Programación

Práctica 1 - Fundamentos de Racket

Fecha de inicio: 11 de agosto de 2016

Fecha de término: 25 de agosto de 2016

Instrucciones

- Resolver los siguientes ejercicios de manera clara y ordenada en un archivo `practica1.rkt`.
- Todas las funciones deben ir acompañadas de al menos 5 pruebas unitarias en un archivo separado `test.rkt`.
- Para ejecutar las pruebas unitarias agregar al inicio la línea `(require "practica1.rkt")`.
- La entrega es por equipos de máximo 3 integrantes. Deben seguir los lineamientos especificados en: `http://lenguajesfc.com/formato.html`.

Ejercicios

1. **Ecuación de primer grado**

   Escribir una función `(ec-lin a b)` que resuelva la ecuación $Ax+B=0$. El primer argumento de la función corresponderá al valor de A y el segundo argumento al valor de B. Ejemplos:

```
>(ec-lin 2 3)
-1 1/2
>(ec-lin 8 1)
-1/8
>(ec-lin 6 9)
-1 1/2
```

2. **Fórmula de Herón**

   Escribir una función `(area-heron a b c)` que encuentre el área de un triángulo dados sus lados, usando la fórmula de Herón. Fórmula:

$$A = \sqrt{S\left(S-a\right)\left(S-b\right)\left(S-c\right)}$$

   donde S es el semiperímetro:

$$S = \frac{a+b+c}{2}$$

   Ejemplos:

```
>(area-heron 3 25 26)
36
>(area-heron 3 4 5)
6
>(area-heron 10.09 3.83 10.46)
19.24
```

3. **Triángulos rectángulos**

   Escribir una función `(triangulo-rec? a b c)` que dados tres números enteros que representan los lados de un triángulo, determina si forman un triángulo rectángulo. Ejemplos:

```
>(triangulo-rec? 5 3 4)
#t
>(triangulo-rec? 5 12 13)
#t
>(triangulo-rec? 1 7 9)
#f
```

4. **s-dígitos**

   Escribe una función recursiva `(s-digito n)` que calcula el s-dígito de `n`. El s-dígito de un número entero positivo `n` se define como sigue:

   - Si `n` consiste de un solo dígito, entonces su s-dígito es `n`.
   - En otro caso, el s-dígito de `n` es igual al s-dígito de la suma de los dígitos de `n`.

   Por ejemplo, el s-dígito de 984 se calcula como sigue:

   $sdigito\,(984) = sdigito\,(9+8+4) = sdigito\,(21) = sdigito\,(2+1) = sdigito\,(3) = 3.$

   Ejemplos:

```
>(s-digito 8)
8
>(s-digito 984)
3
>(s-digito 1729)
1
```

5. **Inversión de números**

   Escribir una función recursiva `(invierte n)` que permita invertir un número.

   Hint: Un número expresado en notación exponencial se ve de la siguiente manera:

   $123 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0$

   Ejemplos:

```
>(invierte 1)
1
>(invierte 123)
321
>(invierte 1729)
9271
```

6. **Eliminar duplicados**

   Escribe una función recursiva `(elim-dup l)` que dada una lista `l` elimina los elementos duplicados adyacentes de una lista dejando únicamente una aparición de cada elemento.

   Ejemplos:

```
>(elimina-dup '(1 1 2 2 3 3 1 1 1 2 3)
'(1 2 3 1 2 3)
>(elimina-dup '(b c c a a a a b b))
'(b c a b)
>(elimina-dup '(1 7 2 9)
'(1 7 2 9)
```

7. **Funciones sobre listas**

   Definir las siguientes funciones:

   a) Escribir una función `(binarios l)` que dada una lista de números, regrese otra con la representación binaria de cada uno de ellos. Ejemplos:

   ```
   >(binarios '(1 2 3))
   '(“1” “10” “11”)
   >(binarios '(1 7 2 9))
   '(“1” “111” “10” “1001”)
   >(binarios '(0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15))
   '("0" "1" "10" "11" "100" "101" "110" "111" "1000" "1001" "1010" "1011"
   "1100" "1101" "1110" "1111")
   ```

   b) Escribir una función `(primos l)` que dada una lista de números, regrese otra conteniendo únicamente los números primos de la original. Ejemplos:

   ```
   >(primos '(1 2 3))
   '(2 3)
   >(primos '(1 7 2 9))
   '(7 2)
   >(primos '(1 2 3 4 5 6 7 8 9 10))
   '(2 3 5 7)
   ```

   c) Escribir una función `(reversar l)` y una función `(reversal l)` que devuelvan la reversa de una lista. Usando la función `foldr` y `foldl` respectivamente. Ejemplos:

   ```
   >(reversar '(1 7 2 9))
   '(9 2 7 1)
   >(reversal '(1 7 2 9))
   '(9 2 7 1)
   ```

8. **Operaciones sobre listas**

   Redefinir cada una de las siguientes operaciones sobre listas. No está permitido usar funciones de Racket que resuelvan directamente los ejercicios (`append, map, filter, take` o `drop`). Apoyarse de la técnica de cazamiento de patrones de ser necesario.

   a) Redefinir la función `append` como `(concatena l1 l2)` que regresa la concatenación de dos listas.

   b) Redefinir la función `map` como `(mapea f l1)` que regresa el resultado de aplicar la función `f` a cada uno de los elementos de la lista.

   c) Redefinir la función `filter` como `(filtra p l1)` que devuelve una lista con los elementos que cumplen el predicado `p`.

d) Redefinir la función `take` como `(toma l n)` que devuelve una lista con los primeros `n` elementos de la lista original.

e) Redefinir la función `drop` como `(quita l n)` que devuelve una lista sin los elementos antes de la posición `n` de la lista original.

9. **Expresiones `let` y lambdas**

   Definir las siguientes funciones combinando expresiones `let` y lambdas. Por ejemplo, para calcular el factorial de 10 puedes escribir:

```
(letrec ([fact (lambda (n)
            (if (< n 2) 1
               (* n (fact (- n 1)))))])
  (fact 10))
```

   a) Definir una función que regrese el mayor de los números 1834 y 1729.

   b) Definir una función recursiva que calcule la suma de los primeros 100 naturales.

Puntos extra

Los puntos extra son individuales.

- (1 pt.) Dada una lista de números enteros, definir una función que devuelve la mayor de las sumas de las sublistas de la misma. En este caso, las sublistas de una lista son todas las listas no vacías de elementos consecutivos de la lista original.

  Por ejemplo:

```
> (sublistas '(1 7 2 9))
((1) (7) (2) (9) (1 7) (7 2) (2 9) (1 7 2) (7 2 9) (1 7 2 9))
> (max-sumas '(1 7 2 9))
19
```