Taller en clase Dr Racket
Fundamentos de Lenguajes Programación

Septiembre 2022

Implemente en Dr Racket los siguientes puntos **usando recursión**, los casos base y recursivo deben quedar bien especificados dentro del código:

1. Diseñar la función **lista-factoriales** la cual recibe un valor n, esta retorna la lista de factoriales desde 0 hasta n. Si el valor es menor que 0 se retorna una lista vacia. El factorial se define como:

$$fac(n) = \begin{cases} 1 & si \quad n = 0 \\ \\ n * fac(n-1) & si \quad n \geq 1 \end{cases} \tag{1}$$

```
> (lista-factoriales 5)
(1 1 2 6 24 120)
```

2. Diseñar la función **lista-fibunnacci** la cual recibe un valor n, esta retorna la lista de fibunnaci desde 0 hasta n. Si el valor es menor que 0 se retorna una lista vacia. La serie de fibunnacci se define como:

$$fib(n) = \begin{cases} 0 & si \quad n = 0 \\ 1 & si \quad n = 1 \\ fib(n-1) + fib(n-2) & si \quad n > 1 \end{cases} \tag{2}$$

```
> (lista-fibunnacci 5)
(0 1 1 2 3 5)
```

3. Diseña la función **predicado-numero** esta recibe una función predicado f, un número n y una lista l. Se retorna $\#t$ si existen exactamente n elementos en la lista l que cumnplen con el predicado f, ejemplo:

```
> (predicado-numero number? 5 '(0 0 1 1 2 "hola"))
#t
> (predicado-numero number? 4 '(0 0 1 1 2 "hola"))
#f
```

4. Diseña la función **predicado-conteo** esta recibe una función predicado f, y una lista l. Se retorna el número de elementos en la lista l que cumnplen con el predicado f, ejemplo:

```
> (predicado-numero number? '(0 0 1 1 2 "hola"))
5
> (predicado-numero number? '("perro" 0 1 1 2 "hola"))
4
```

5. Diseña la función **(binary-to-natural)** recibe una lista con 0 o 1 que representan un número binario. Esta función opera recursivamente la lista para obtener el número decimal equivalente.

```
> (binary-to-natural '())
0
> (binary-to-natural '(1 0 0))
4
> (binary-to-natural '(1 1 0 0))
12
> (binary-to-natural '(1 1 1 1))
15
> (binary-to-natural '(1 0 1 0 1))
21
> (binary-to-natural '(1 1 1 1 1 1 1 1 1 1 1 1 1))
8191
```

6. Diseña la función **(natural-to-binary)** recibe un número decimal y retorna una lista de 0 y 1 que representa el número binario equivalente.

```
> (natural-to-binary 0))
()
> (natural-to-binary 4)
(1 0 0)
> (natural-to-binary 12)
(1 1 0 0)
> (natural-to-binary 15)
(1 1 1 1)
> (natural-to-binary 21)
(1 0 1 0 1)
> natural-to-binary 8191)
(1 1 1 1 1 1 1 1 1 1 1 1 1)
```

7. Diseña la función **(countdown)** toma un número n y genera una lista de números naturales entre n y 0. Si el n ingresado es menor que 0 retorna una lista vacía.

```
> (countdown 5)
> (5 4 3 2 1 0)
> (countdown -1)
> ()
```