006. Encontrar el máximo elemento de una lista

Escriba una función **maximo** que toma una lista de números como argumento y devuelve el número más grande presente en la lista.

```
(maximo ‘( 3 1 2 5 6 3 4)) -> 6
(maximo ‘( 3 1 2 5 6 3 4 6 1)) -> 6
(maximo ‘( 3 )) -> 3
```

El valor máximo puede aparecer una o más veces.

Un pregunta que debemos hacernos es que de responder si la lista está vacía. En principio una lista vacía no cumple con el enunciado, ya que no es una lista de números si no tiene ningún número. Expandimos el enunciado para indicar que si la lista argumento está vacía, se debe informar un error.

```
(maximo ‘()) -> error
```

También hay un caso especial cuando la lista tiene un único número, en cuyo caso ese valor será el máximo buscado.

Ahora debemos pasar a considerar el problema desde el punto de la recursión.

El “enano imperativo” que todo programador lleva adentro nos empuja a pensar en términos de bucles for y de índices, y recorrer desde el elemento uno, pasar al siguiente, y luego seguir así hasta que lleguemos al final de la estructura de datos. ¡Debes combatirlo!

En programación funcional pensamos el problema en términos de **valores** y **funciones**. Un valor puede tener una estructura, por ejemplo una lista de números. La estructura de una lista consta de una cabeza y una cola. La cola es una lista, y allí está la perspectiva recursiva de este problema.

Tomemos la lista del primer ejemplo:

‘(3 1 2 5 6 3 4) es una lista de números
la divido en cabeza y cola: **3** y **‘(1 2 5 6 3 4)**

donde **‘(1 2 5 6 3 4)** es una lista de números, o sea la misma entrada para nuestro procedimiento solicitado.

Ahora podemos concentrarnos en la idea de máximo. El máximo de una lista es un número que pertenece a la lista y que no hay otro más grande que él.

Lo pasamos a cabeza y cola: el máximo es la cabeza o está en la cola de la lista.

Si calculamos el máximo de la cola y la cabeza de la lista, uno de los dos será el máximo, aquel que sea mayor. En nuestro ejemplo más arriba será:

3 (la cabeza) y 6 (el máximo de la cola); entre los dos, el mayor es 6 y por eso es el resultado buscado.

Codifiquemos esta idea en Scheme:

```
(define maximo
  (lambda (lst)
    (cond ((null? lst) (error "maximo: no admite lista vacia"))
          ((null? (cdr lst)) (car lst))
          (else
            (let [(cabeza (car lst))
                  (maxcola (maximo (cdr lst)))]
              (if (> cabeza maxcola) cabeza maxcola))))))
```

Estamos usando dos construcciones nuevas: **`cond`** y **`let`**.

El **`cond`** es una especie de *switch*, donde hay una secuencia de pares *condición->acción*: la primer condición que evalúa a verdadero hace que su acción correspondiente sea el resultado del cond; al final puede haber un **`else`** con su acción, el valor de la cual será el resultado del **`cond`** si ninguna de las condiciones anteriores fue verdadera.

El **`let`** permite asociar nombres a valores, para evitar la repetición de subexpresiones en el código. En nuestro caso hemos asociado **`cabeza`** con el primer elemento en la lista argumento, y **`maxcola`** con el máximo de la cola de la lista de números.

Consideramos entonces con el cond los tres casos posibles:

a) **`lst`** está vacía y hay que emitir un error
b) **`lst`** tiene un único elemento, el cual es el resultado buscado
c) **`lst`** tiene más de un elemento, entonces podemos obtener la cabeza y la cola

Si no consideramos el caso de la lista vacía, el código se puede simplificar para que quede: (sin **`cond`** ni condición de error)

```
(define maximo
  (lambda (lst)
    (if (null? (cdr lst))
      (car lst)
      (let [(cabeza (car lst))
            (maxcola (maximo (cdr lst)))]
        (if (> cabeza maxcola) cabeza maxcola)))))
```

Supongamos que no deseamos usar let, por la razón que se nos ocurra, el código quedaría: (sin **let**)

```
(define maximo
  (lambda (lst)
    (if (null? (cdr lst))
      (car lst)
      (if (> (car lst) (maximo (cdr lst)))
        (car lst)
        (maximo (cdr lst))))))
```

Todavía nadie ha logrado convencerme que la versión sin **let** es más legible que la versión con **let**. Sin mencionar que cuando el máximo se encuentra en la cola de la lista se lo calcula dos veces.

Sólo por completitud, veamos el efecto de reintroducir en el código sin **let**, la condición de lista vacía como error:

```
(define maximo
  (lambda (lst)
    (if (null? lst) (error "maximo: no admite lista vacia")
      (if (null? (cdr lst))
        (car lst)
        (if (> (car lst) (maximo (cdr lst)))
          (car lst)
          (maximo (cdr lst)))))))
```

Los **if** anidados no son legibles en ningún lenguaje de programación.