011. Aplana una lista anidada

Escriba una función que toma una lista anidada de cualquier profundidad, y retorna una lista con los átomos. Si el argumento no es una lista, la función no está definida.

Un átomo es una secuencia de caracteres que no tiene paréntesis; puede ser un número, un símbolo, un string. Una forma simple de definirlo en Scheme es decir que un átomo es algo que no es una lista, ni vacía ni con elementos:

```
(define atom?
  (lambda (x)
    (and (not (pair? x)) (not (null? x)))))
```

Entonces podemos ver que sucede:

```
(atom? '()) -> #f
(atom? '(1 2 3)) -> #f
(atom? '(1 (2 4) 3)) -> #f
(atom? 5) -> #t
(atom? 'a) -> #t
```

Supongamos unos casos de prueba para la función **aplana**:

```
(aplana '()) -> '()
(aplana '(1 2 a b)) -> '(1 2 a b)
(aplana '(1 2  (a b c (d)) (f g) 9)) -> '(1 2  a b c d f g 9)
```

¿Cuáles son los casos de este problema?

Como el argumento principal es una lista, en principio hay dos casos:

a) la lista está vacía
b) la lista tiene al menos un argumento

Ahora bien, si tiene un argumento, el mismo puede ser un átomo (que va a ir al resultado), o puede ser a su vez una lista, la cual habrá que procesar y sus elementos agregar al resultado.

Entonces hay tres casos en el problema:

a) la lista está vacía
b) la lista tiene un elemento que es un átomo
c) la lista tiene un elemento que no es una átomo

Vamos a estructurar nuestro programa en torno a esa partición de casos. Una partición es una división en subconjuntos de tal manera que todo elemento del conjunto principal

pertenece a uno de los subconjuntos, la unión de los subconjuntos nos da el conjunto principal, y la intersección dos a dos de los subconjuntos es vacía.

El aplanado de una lista vacía es la lista vacía, no hay átomos en la entrada, no los hay en la salida.

El aplanado de una lista cuya cabeza es un átomo se construye con ese átomo como primer elemento de una lista cuya cola es el aplanado de la cola de la lista dada. Aplicamos la recursión a la cola de la lista inicial, para conseguir una lista de átomos, al principio de la cual insertamos el átomo mediante **cons**.

El aplanado de una lista cuya cabeza es a su vez una lista. Ahora tenemos en la cabeza una lista, y en la cola otra lista. Aplanamos cada una de ellas y como el resultado de aplanarlas son dos listas simples, las concatenamos con append para conseguir el resultado final.

Veamos el código Scheme:

Aquí expresamos que hay una lista vacía o no vacía. Luego si no es vacía hay dos casos: el **car** es átomo o es lista:

```
(define aplana
  (lambda (lst)
    (if (null? lst) '()
      (if (atom? (car lst))
          (cons (car lst) (aplana (cdr lst)))
          (append (aplana (car lst)) (aplana (cdr lst)))))))
```

Una alternativa casi idéntica usando **cond** queda más expresiva con los tres casos:

```
(define aplana
  (lambda (lst)
    (cond ((null? lst) '())
          ((atom? (car lst)) (cons (car lst) (aplana (cdr lst))))
          (else (append (aplana (car lst)) (aplana (cdr lst)))))))
```

Alternativa con **if** y con **let** para usar nombres descriptivos:

```
(define aplana
  (lambda (lst)
    (if (null? lst) null         ; aplana de lista vacía
      (let [(cabeza (car lst))
            (cola (cdr lst))]
        (if (atom? cabeza)
          (cons cabeza (aplana cola))                ; cabeza es átomo
          (append (aplana cabeza) (aplana cola)))))) ; cabeza es lista
```