Tema 4: Estructuras de datos recursivas

Listas estructuradas

- Listas que contienen otras listas. Al usar car **devuelve un elemento u otra lista**.
- Lo contrario de lista estructurada es una **lista plana** (no contiene listas).
- Las **hojas** son los elementos de una lista que no son sublistas.

Ej:

(define lista (list 'a 'b (list 'c 'd 'e) (list 'f (list 'g 'h))))
(define lista '(a b (c d e) (f (g h))))

Definición Hoja

```
(define (hoja? elem)
   (not (list? elem)))
```

```
(define lista '((1 2) 3 4 (5 6)))
(hoja? (car lista)) ; ⇒ #f
(hoja? (cadr lista)) ; ⇒ #t
(hoja? (caddr lista)) ; ⇒ #t
(hoja? (cadddr lista)) ; ⇒ #f
```

Definición Lista plana

```
(define (plana? lista)
   (or (null? lista)
       (and (hoja? (car lista))
            (plana? (cdr lista)))))
```

```
(define (plana-fos? lista)
  (for-all? hoja? lista))
```

```
(plana? '(a b c d e f)) ; ⇒ #t
(plana? (list (cons 'a 1) "Hola" #f)) ; ⇒ #t
(plana? '(a (b c) d)) ; ⇒ #f
(plana? '(a () b)) ; ⇒ #f
```

Definición Lista estructurada

```
(define (estructurada? lista)
   (and (not (null? lista))
        (or (list? (car lista))
            (estructurada? (cdr lista)))))
```

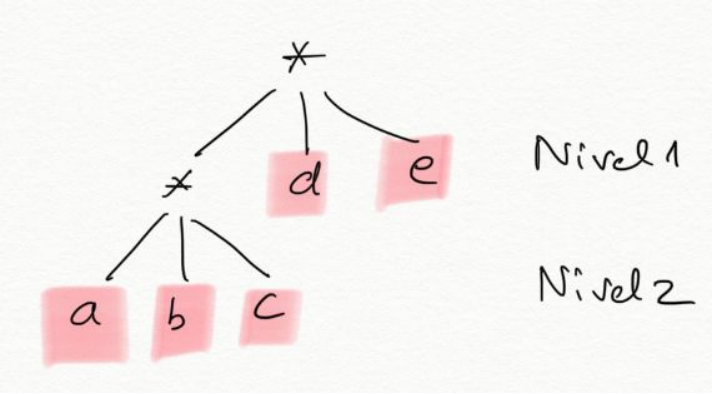
Se podría implementar también usando la función de orden superior `exists?` consultar si algún elemento de la lista es también otra lista.

```
(define (estructurada-fos? lista)
  (exists? list? lista))
```

```
(estructurada? '(1 2 3 4)) ; ⇒ #f
(estructurada? (list (cons 'a 1) (cons 'b 2) (cons 'c 3))) ; ⇒ #f
(estructurada? '(a () b)) ; ⇒ #t
(estructurada? '(a (b c) d)) ; ⇒ #t
```

Representación (Pseudo árboles con niveles)

EJ: ((a b c) d e)



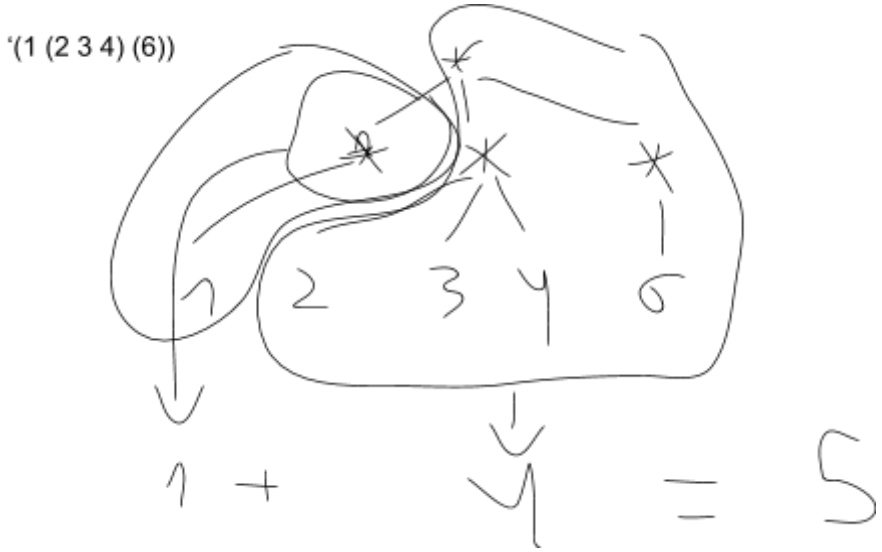
- Cada * representa una lista.
- Las ramas que salen del * representan los el. de la lista.

Funciones sobre listas estructuradas

Ejemplos:

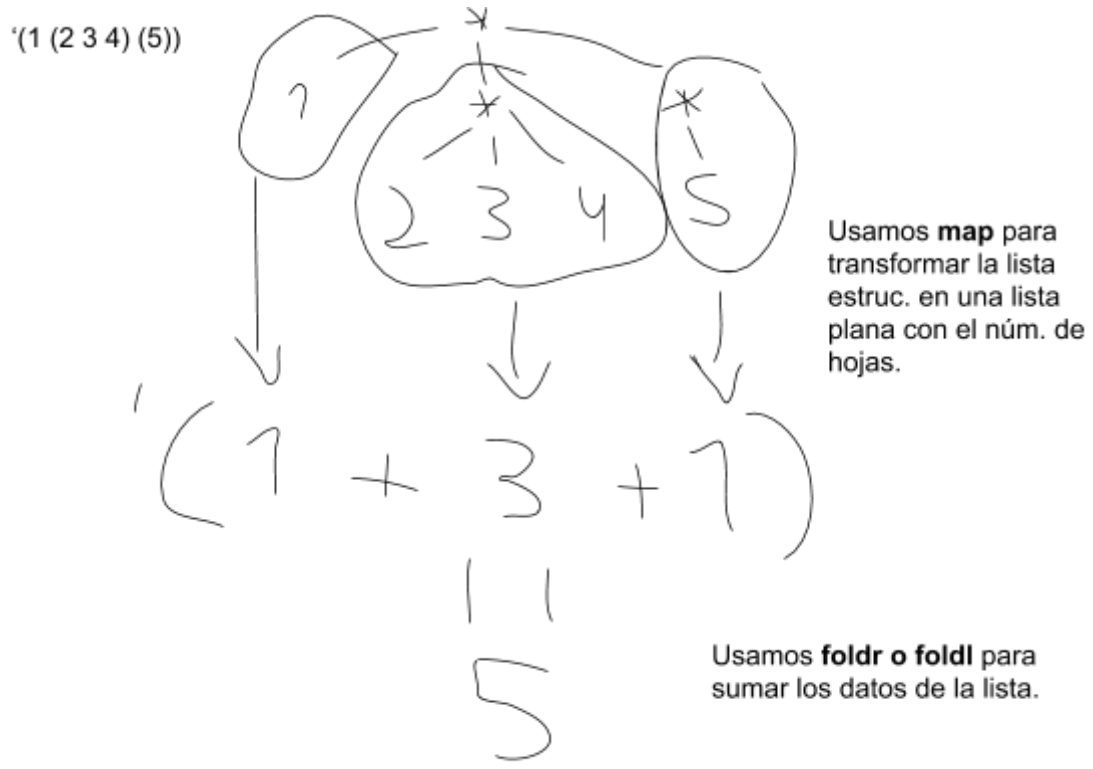
- `(aplana lista)` : devuelve una lista plana con todas las hojas de la lista
- `(pertenece-lista? dato lista)` : busca una hoja en una lista estructurada
- `(nivel-hoja dato lista)` : devuelve el nivel en el que se encuentra un dato en una lista
- `(cuadrado-estruct lista)` : eleva todas las hojas al cuadrado (suponemos que la lista estructurada contiene números)
- `(map-estruct f lista)` : similar a map, aplica una función a todas las hojas de la lista estructurada y devuelve el resultado (otra lista estructurada)

Ejercicio: Diseña la función (num-hojas lista) que cuenta el número de hojas de una lista estructurada.




```
(define (num-hojas lista)
        (cond   ((null? lista) 0)
                ((hoja? (car lista)) (car lista))
                (else (+ (num-hojas (car lista))
                         (num-hojas (cdr lista))))))
```

Ejercicio: Diseña la función (num-hojas-fos lista) que cuenta el número de hojas de una lista estructurada usando FOS.

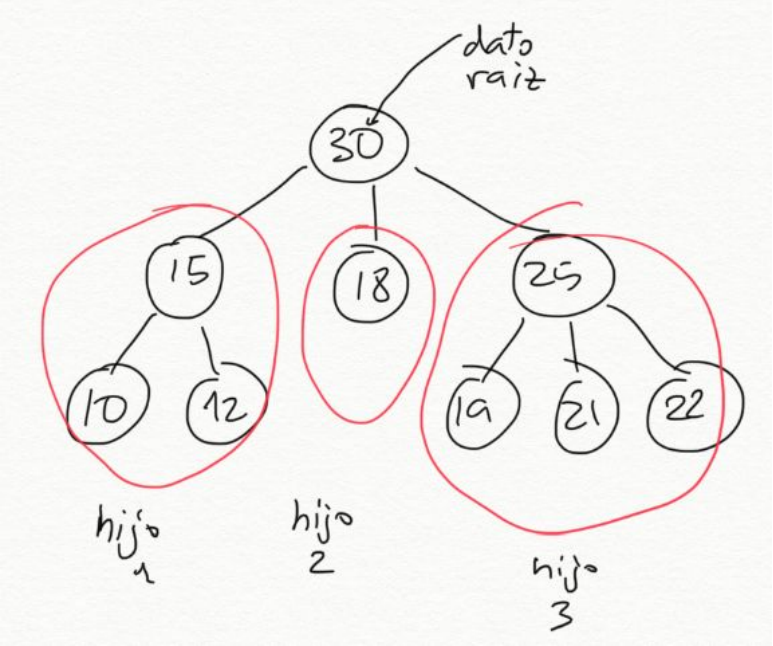



```
(define (num-hojas-fos lista)
        (foldr + 0
               (map (lambda (x)
                            (if (hoja? x) 1
                                          (num-hojas-fos x))) lista)))
```

Árboles

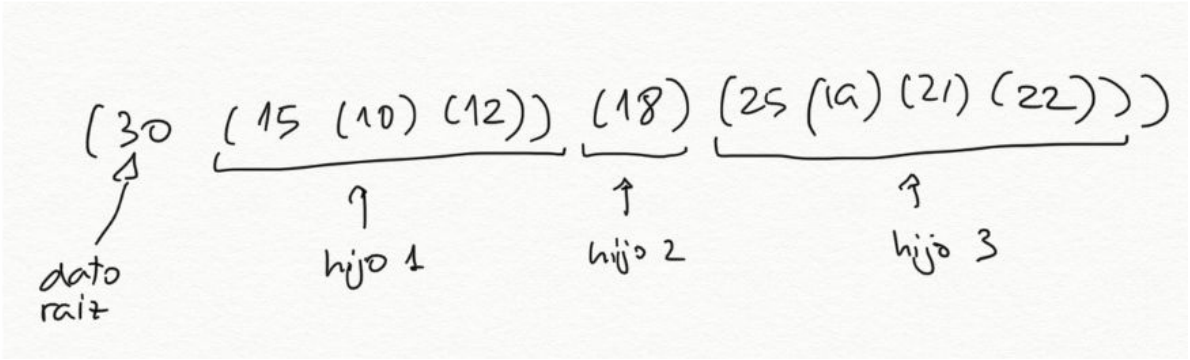
- Estructura de datos definida por un valor **raíz**, que es el padre de toda la estructura, del que salen otros **subárboles hijos**.
- Uso de la **recursión mutua** para resolver problemas de forma **recursiva**.
- Recursivamente:
  - Un **dato** (raiz) **y una lista de hijos** (árboles)
  - Una **hoja** será un **árbol sin hijos** (un dato con una lista de hijos vacía).

Ej:




```
(30 (15 (10) (12)) (18) (25 (19) (21) (22)))
```

Los elementos de esta lista son:



Barrera de abstracción (-arbol)

- Cuando trabajemos con árboles, deberemos usar las funciones propias de este tipo de estructura para contruïr un nuevo árbol (**constructores**) y obtener los elementos del árbol (**selectores**).

Selectores

```
1  (define (dato-arbol arbol)
2      (car arbol))
3
4  (define (hijos-arbol arbol)
5      (cdr arbol))
6
7  (define (hoja-arbol? arbol)
8     (null? (hijos-arbol arbol)))
```

-> **devuelve el dato** raíz

-> **devuelve una lista** de árboles hijos

-> comprueba que arbol es una hoja

```
(dato-arbol arbol1) ; ⇒ 30
(hijos-arbol arbol1) ; ⇒ ((15 (10) (12)) (18) (25 (19) (21) (22)))
(hoja-arbol? (car (hijos-arbol arbol1))) ; ⇒ #f
(hoja-arbol? (cadr (hijos-arbol arbol1))) ; ⇒ #t
```

Contructores

```
1  (define (nuevo-arbol dato lista-arboles)
2     (cons dato lista-arboles))
```

-> construye un árbol

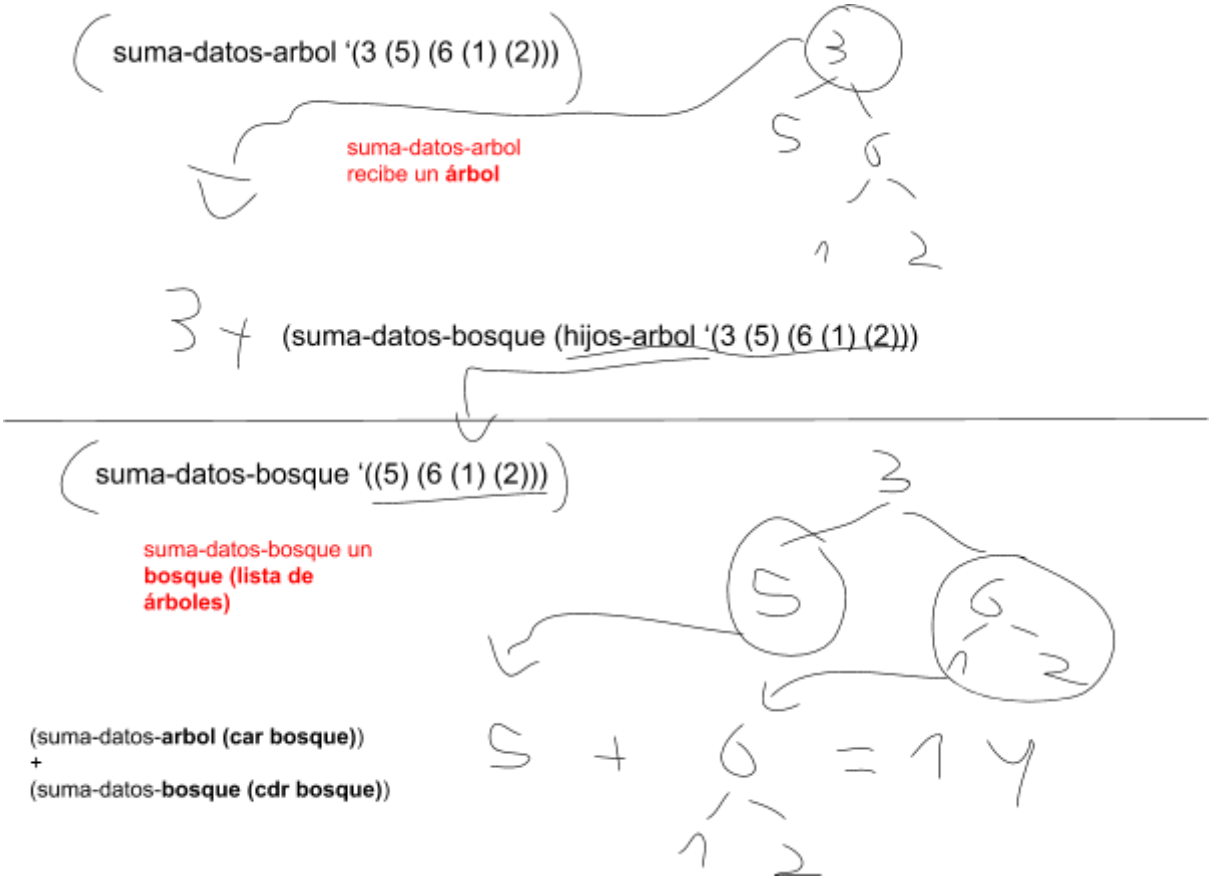
```
(define arbol-15 (nuevo-arbol 15 (list (nuevo-arbol 10 '())
                                       (nuevo-arbol 12 '()))))
(define arbol-18 (nuevo-arbol 18 '()))
(define arbol-25 (nuevo-arbol 25 (list (nuevo-arbol 19 '())
                                       (nuevo-arbol 21 '())
                                       (nuevo-arbol 22 '()))))
(define arbol1b (nuevo-arbol 30 (list arbol-15 arbol-18 arbol-25)))
arbol1b ; ⇒ (30 (15 (10) (12)) (18) (25 (19) (21) (22)))
```

Funciones recursivas sobre árboles

Ejemplos:

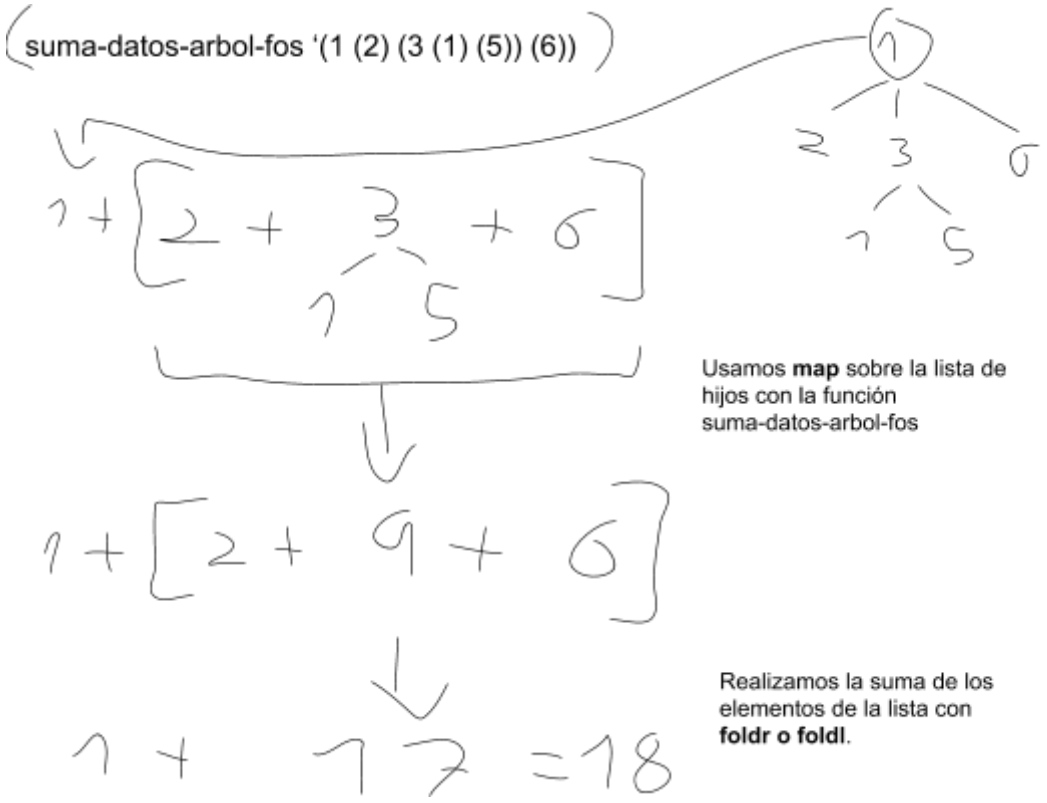
- `(suma-datos-arbol arbol)` : devuelve la suma de todos los nodos
- `(to-list-arbol arbol)` : devuelve una lista con los datos del árbol
- `(cuadrado-arbol arbol)` : eleva al cuadrado todos los datos de un árbol manteniendo la estructura del árbol original
- `(map-arbol f arbol)` : devuelve un árbol con la estructura del árbol original aplicando la función f a subdatos.
- `(altura-arbol arbol)` : devuelve la altura de un árbol

Ejercicio: Implementa una función recursiva suma-datos-arbol que sume todos los datos de un árbol.




```
(define (suma-datos-arbol arb)
        (+ (dato-arbol arb) (suma-datos-bosque (hijos-arbol arb))))
(define (suma-datos-bosque bosq)
        (if (null? bosq) 0
                (+ (suma-datos-arbol (car bosq))
                   (suma-datos-bosque (cdr bosq)))))
```

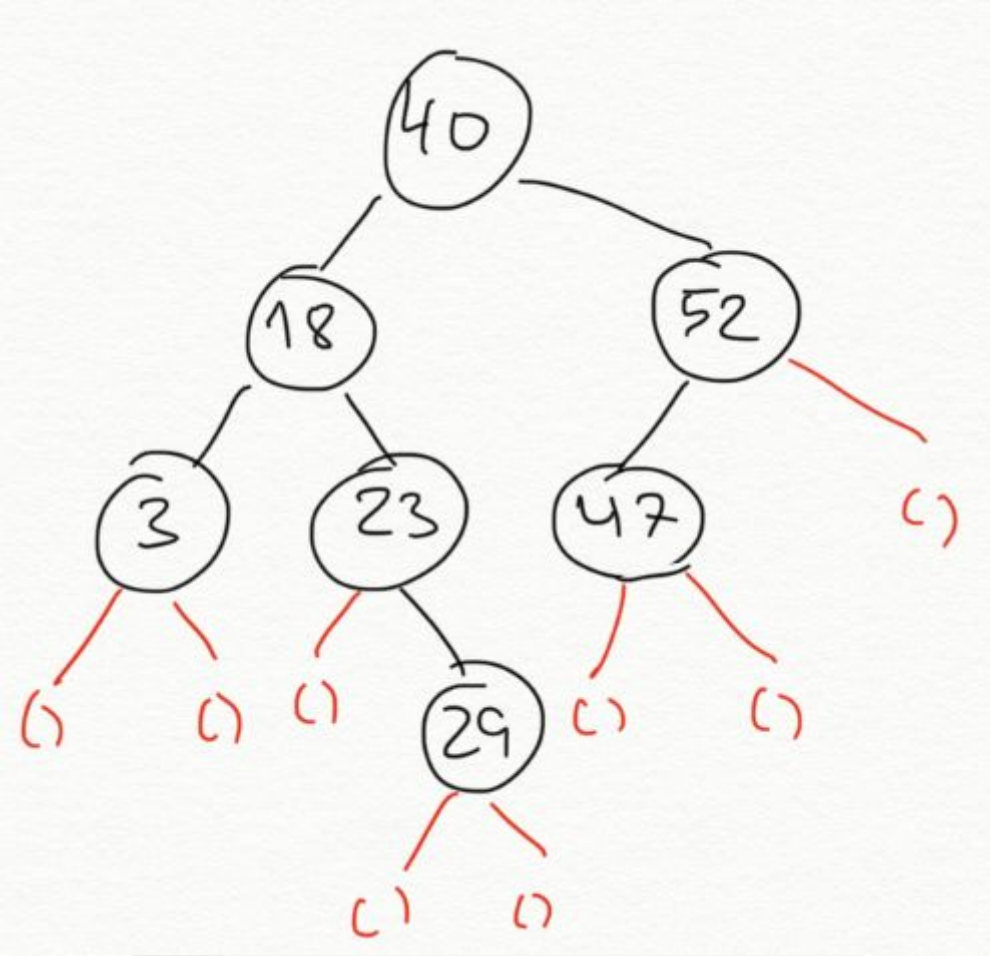
Ejercicio: Implementa la función suma-datos-arbol usando FOS que sume todos los datos de un árbol.




```
(define (suma-datos-arbol-fos arbol)
        (folr + (dato-arbol arbol)
              (map suma-datos-arbol-fos (hijos-arbol arbol))))
```

Árboles binarios

- Árboles cuyos nodos tienen **0, 1 o 2 hijos**.
- Representamos con una **lista** con: un **dato**, **hijo der.** e **hijo izq.**
- **Nodo vacío == lista vacía**




```
(40 (18 (3 () ())
        (23 ()
            (29 () ())))
    (52 (47 () ())
        ()))
```

Barrera de abstracción (-arbolb)

- Cuando trabajemos con árboles binarios, deberemos usar las funciones propias de este tipo de estructura para contruïr un nuevo árbol (**constructores**) y obtener los elementos del árbol (**selectores**).

Selectores

```
1  (define (dato-arbolb arbol)
2     (car arbol))
3
4  (define (hijo-izq-arbolb arbol)
5     (cadr arbol))
6
7  (define (hijo-der-arbolb arbol)
8     (caddr arbol))
9
10 (define arbolb-vacio '())
11
12 (define (vacio-arbolb? arbol)
13    (equal? arbol arbolb-vacio))
14
15 (define (hoja-arbolb? arbol)
16    (and (vacio-arbolb? (hijo-izq-arbolb arbol))
17         (vacio-arbolb? (hijo-der-arbolb arbol))))
```

Contructores

```
1  (define (nuevo-arbolb dato hijo-izq hijo-der)
2      (list dato hijo-izq hijo-der))
```

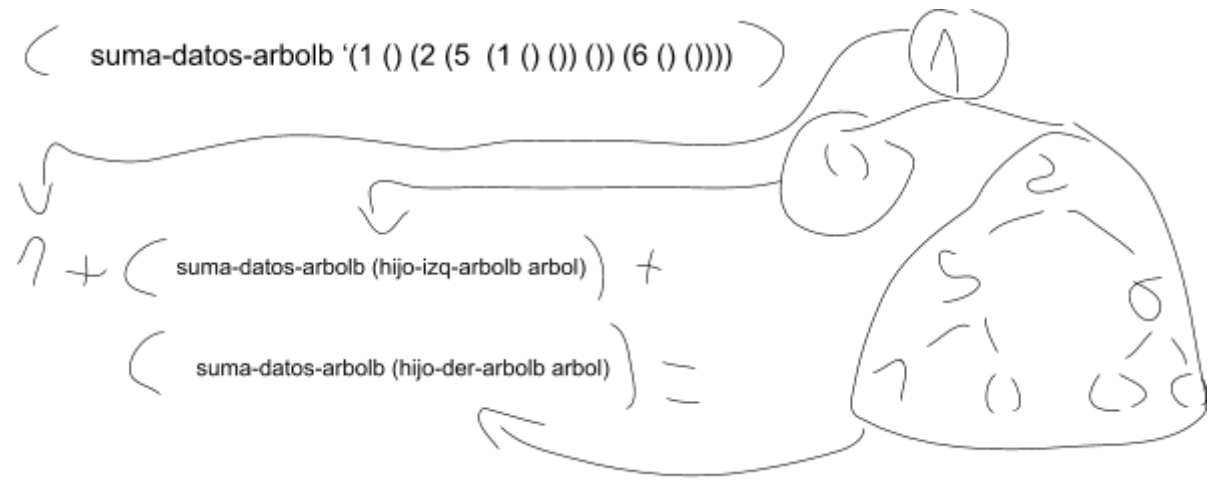
```
(define arbolb1
   (nuevo-arbolb 10 (nuevo-arbolb 8 arbolb-vacio arbolb-vacio)
                    (nuevo-arbolb 15 arbolb-vacio arbolb-vacio)))
```

Funciones recursivas sobre árboles binarios

Ejemplos:

- `(suma-datos-arbolb arbol)` : devuelve la suma de todos los nodos
- `(to-list-arbolb arbol)` : devuelve una lista con los datos del árbol
- `(cuadrado-arbolb arbol)` : eleva al cuadrado todos los datos de un árbol manteniendo la estructura del árbol original

Ejercicio: Implementa una función recursiva suma-datos-arbolb que sume todos los datos de un árbol binario.



```
(define (suma-datos-arbolb arbol)
      (if (vacio-arbolb? arbol) 0
            (+ (dato-arbolb arbol)
               (suma-datos-arbolb (hijo-izq-arbolb arbol))
               (suma-datos-arbolb (hijo-der-arbolb arbol)))))
```

Bibliografía

Los enunciados de los ejercicios resueltos, y los resúmenes, se han elaborado a partir del material publicado en https://domingogallardo.github.io/ , material del que es propietario el Departamento de Ciencia de la Computación e Inteligencia Artificial de la Universidad de Alicante, Domingo Gallardo, Cristina Pomares, Antonio Botía y Francisco Martínez.