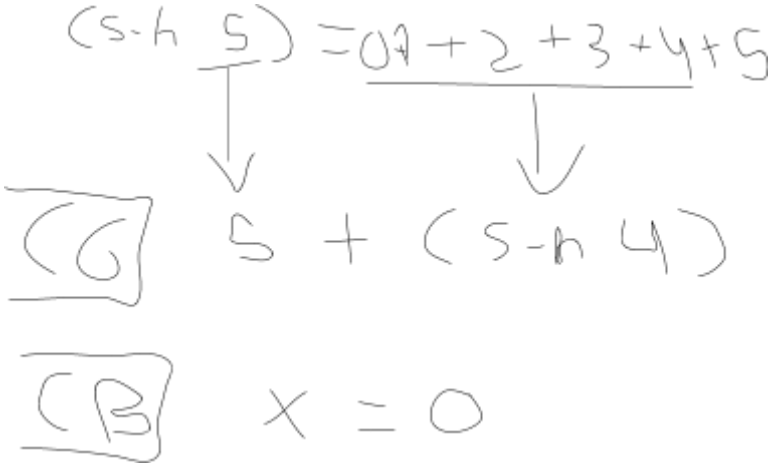
Modelo de sustitución

Ej:
```
(define (zero x) (- x x))
   (zero (random 10))

   1. Modelo de sustitución orden aplicativo

(zero (random 10))R3, evaluamos random 10
(zero 7)               R4, eval arg, sustituir el cuerpo de zero
(- 7 7)                R3, eval arg, eval función primitiva -
0

   2. Modelo de sustitución orden normal

(zero (random 10))R4, sustituimos la fón por cuerpo
(- (random 10) (random 10))   R3, Evaluamos los argumentos uno a uno
(- 7 3)                R3, Evaluamos la expresión primitiva -
4

*** En este caso no devuelven el mismo resultado porque la función random
no pertenece al paradigma funcional ***
```

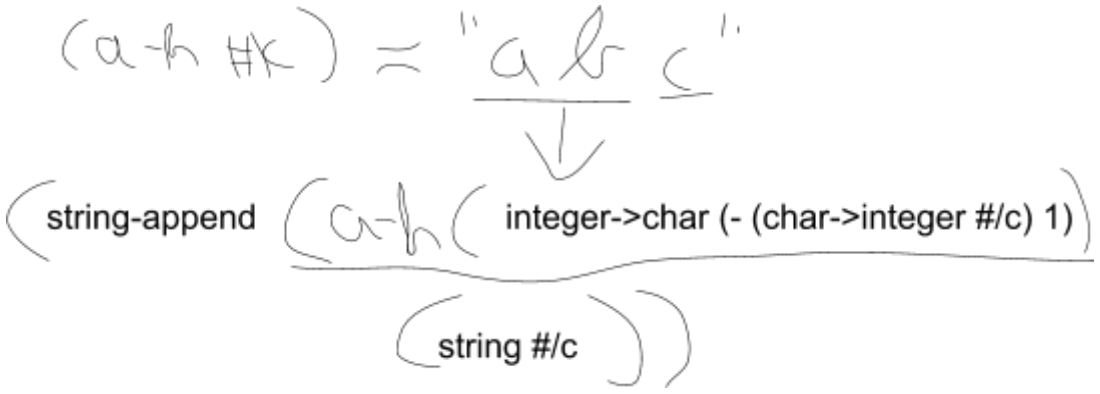
Funciones Recursivas

Ej: Suma-hasta x




```
(define (suma-hasta x)
        (if (= x 0) 0
                (+ x (suma-hasta (- x 1))))
```

Ej2: Alfabeto-hasta x

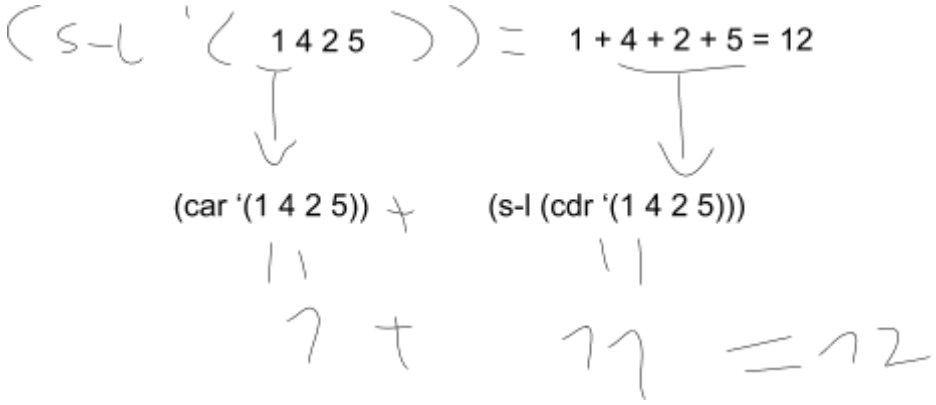



```
(define (alfabeto-hasta x)
      (if (equal? x #/a) “a”
                  (string-append (alfabeto-hasta (integer->char (-
(char->integer x) 1)) (string x))))
```

```
*** Se puede mejorar sustituyendo la parte que devuelve el carácter
anterior a x, por la función anterior que realice dicha tarea ***
```

Funciones recursivas con listas

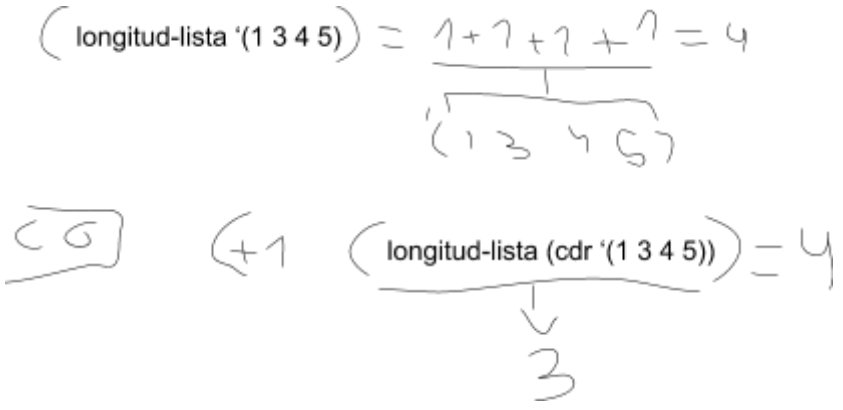
Ej: (suma-lista lista-numeros)




```
(define (suma-lista lista)
        (if (null? lista) 0
                        (+ (car lista) (suma-lista (cdr lista)))))
```

Escogemos como **caso base que la **lista sea vacía**, en ese caso **devolvemos** el núm. **0****

Ej: (longitud lista)

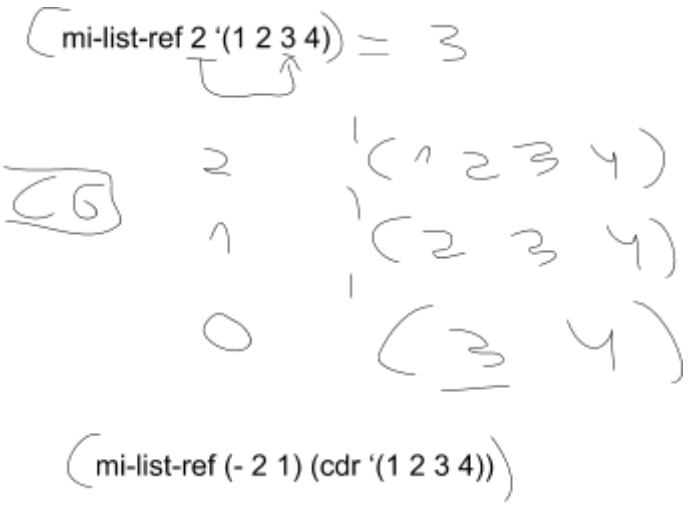



```
(define (longitud lista)
        (if (null? lista) 0
                (+ 1 (longitud (cdr lista)))))
```

*** **Caso base**, cuando me den una **lista vacía, devolveré 0.** Nótese que puede **depender del enunciado**, si este nos dice que siempre recibiremos una **lista con mínimo un elemento**, el caso base será: (if **(null? (cdr lista))** 1 ***

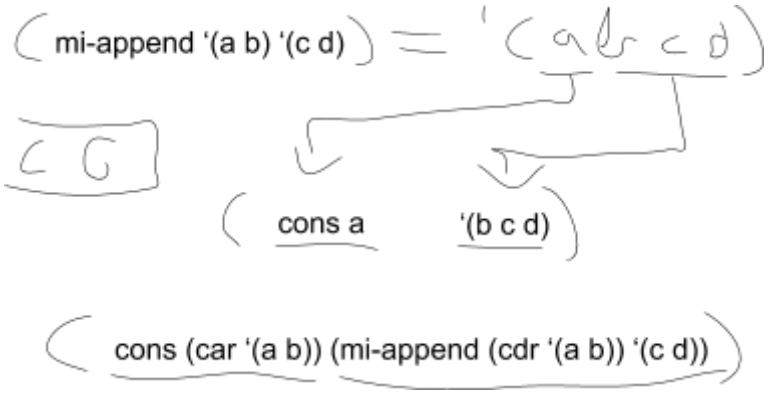
Funciones rec. constructoras de listas

Ej: (mi-list-ref n lista)




```
(define (mi-list-ref n lista)
        (if (= 0 n) (car lista)
                (mi-list-ref (- n 1) (cdr lista))))
```

Ej: (mi-append lista1 lista2)

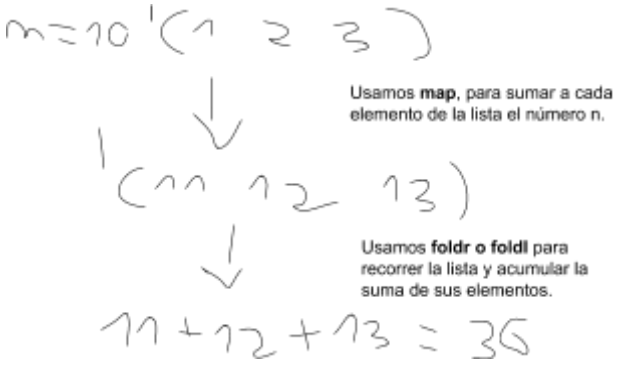



```
(define (mi-append lista1 lista2)
        (if (null? lista1) lista2
                (cons (car lista1) (mi-append (cdr lista1) lista2))))
```

*** Caso base, cuando la lista1 no tenga elementos (lista vacia '()), devolvemos la lista2 ***

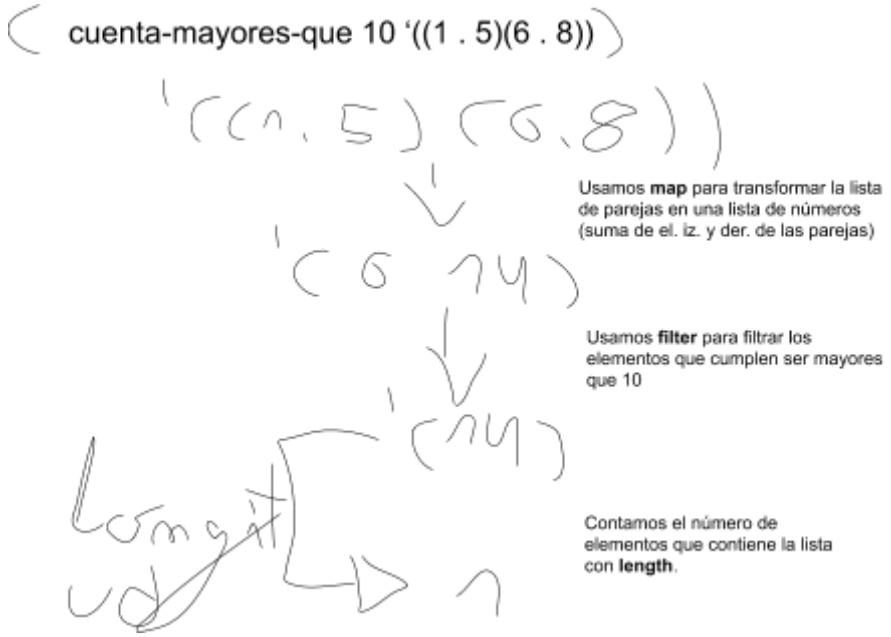
Funciones que usan FOS

Ej: (suma-n-total n lista)




```
(define (suma-n-total n lista)
        (foldr + 0
               (map (lambda (x) (+ x n)) lista)))
```

Ej: (cuenta-mayores-que n lista-parejas)




```
(define (cuenta-mayores-que n lista)
      (length
            (filter (lambda (num) (> num n))
                  (map (lambda (pareja) (+ (car pareja) (cdr pareja)))
                  lista))))
```

```
//p1 ej 3 apartado b
MODELO NORMAL

(g 0 (f 10))
(if (= 0 0)             //R4, sust la fon por cuerpo
      0
      (f 10))
0                       //R3, fon primitiva
(if #t 0 (f 10))        //
0

MODELO APLICATIVO -> resolvemos desde dentro hacia fuera

(g 0 (f 10))
(g 0 (/ 10 0))    //R4, evaluado primero arg. sustituimos el cuerpo
error //R3, error
```

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.