

Funcional

Finales

24/02/03 – Ejercicio 1

```
(define mapS (lambda (F L)
  (if (null? L) '()
      (cons (F (car L)) (mapS F (cdr L))))))

(define perConRep (lambda (L long)
  (if (= 1 long) (mapS list L)
      (distribuirPCR L (perConRep L (- long 1)))))

(define distribuirPCR (lambda (L1 L2)
  (if (null? L1) '()
      (append (mapS (lambda (X) (cons (car L1) X)) L2)
                (distribuirPCR (cdr L1) L2))))

(define menoresQue
  (lambda (max lista)
    (if (null? lista)
        ()
        (if (<= (eval (cons '+ (car lista))) max)
            (cons (car lista) (menoresQue max (cdr lista)))
            (menoresQue max (cdr lista))))))

(define combina
  (lambda (listaN maximoValor long)
    (if (null? listaN)
        ()
        (menoresQue maximoValor (perConRep listaN long)))))
```

28/02/05 – Ejercicio 1

```
(define sumarCifras
  (lambda (n)
    (if (< n 10)
        n
        (+ (remainder n 10) (sumarCifras (quotient n 10)))))

(define divisible-por-tres
```

```
(lambda (n)
  (if (> n 10)
      (divisible-por-tres (sumarCifras n))
      (if (or (= n 9) (= n 6) (= n 3))
          #t
          #f))))
```

```
(define algo
  (lambda (n)
    (if (null? n)
        '()
        (eval (cons '+ (cons (car n) (algo (cdr n))))))))
```

Sin Fecha 1 – Ejercicio 1

```
(define x1
  (lambda (lista)
    (caar lista)))
(define y1
  (lambda (lista)
    (cadar lista)))
(define x2
  (lambda (lista)
    (caadr lista)))
(define dX1
  (lambda (lista)
    (cadar lista)))
(define y1d
  (lambda (lista)
    (caddar lista)))
(define dife
  (lambda (lista)
    (if (null? (cdr lista))
        ()
        (cons (list (x1 lista) (- (x2 lista) (x1 lista)) (y1 lista)) (dife (cdr lista))))))
(define area
  (lambda (lista)
    (if (null? lista)
        0
        (+ (* (dX1 lista) (y1d lista)) (area (cdr lista))))))
```

09/02/04 – Ejercicio 3

```
(define orden
  (lambda (criterio lista)
    (if (null? lista)
        '()
        (cons (armar1Grupo criterio lista) (orden criterio (eliminarHasta (ultimoElemento (armar1Grupo
criterio lista)) lista))
        )))

(define armar1Grupo
  (lambda (criterio lista)
    (if (null? (cdr lista))
        ()
        (if(criterio (car lista) (cadr lista))
            (if(null? (cddr lista))
                (cons (car lista) (list (cadr lista)))
                (if (criterio (cadr lista) (caddr lista))
                    (cons (car lista) (armar1Grupo criterio (cdr lista)))
                    (cons (car lista) (list (cadr lista))))
            ))
        (armar1Grupo criterio (cdr lista))
        )))

(define Mayor-d
  (lambda (x y)
    (> y (+ x 2))
    ))

(define ultimoElemento
  (lambda (lista)
    (if (null? lista)
        'a
        (car (reverse lista))
        )))

(define eliminarHasta
  (lambda (n lista)
    (if(null? lista)
        ()
        (if(equal? n (car lista))
            (cdr lista)
            (eliminarHasta n (cdr lista))
            ))))
```

07/07/03 – Ejercicio 2 (hay que probarlo)

```
(define iguales
  (lambda (m1 m2)
```

```

(if(and (null? m1) (null? m2))
  #t
  (if(= (m1 -a) (m2 -a))
    (if(= ((car m1) -a) ((car m2) -a))
      (if(vectorIguales (car m1) (car m2))
        (iguales (cdr m1) (cdr m2))
        #f)
      #f)
    #f))))
(define vectorIguales
  (lambda (vector1 vector2)
    (if(and (null? vector1) (null? vector2))
      #t
      (if(= (vector1 a) (vector2 a))
        (vectorIguales (cdr vector1) (cdr vector2))
        #f))))
(define -a
  (lambda (lista)
    (if(null? lista)
      0
      (longitud lista))))
(define longitud
  (lambda (lista)
    (if(null? lista)
      0
      (+ 1 (longitud (cdr lista))))))
(define a
  (lambda (lista)
    (if(null? lista)
      ()
      (car lista))))

```