UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA

DEPARTAMENTO DE
INFORMÁTICA Y ANÁLISIS NUMÉRICO

# PROGRAMACIÓN DECLARATIVA

INGENIERÍA INFORMÁTICA

CUARTO CURSO

PRIMER CUATRIMESTRE

## Tema 5.- Tipos de datos compuestos

**Primera parte: Scheme**

- Tema 1.- Introducción al lenguaje Scheme
- Tema 2.- Expresiones y funciones
- Tema 3.- Predicados y sentencias condicionales
- Tema 4.- Iteración y recursión
- Tema 5.- Tipos de datos compuestos
- Tema 6.- Abstracción de datos
- Tema 7.- Lectura y escritura

**Segunda parte: Prolog**

- Tema 8.- Introducción al lenguaje Prolog
- Tema 9.- Elementos básicos de Prolog
- Tema 10.- Listas
- Tema 11.- Reevaluación y el “corte”
- Tema 12.- Entrada y salida

# Primera parte: Scheme

Tema 1.- Introducción al lenguaje Scheme

Tema 2.- Expresiones y funciones

Tema 3.- Predicados y sentencias condicionales

Tema 4.- Iteración y recursión

Tema 5.- Tipos de datos compuestos

Tema 6.- Abstracción de datos

Tema 7.- Lectura y escritura

# Índice

1. Vectores
2. Pares
3. Listas
4. Funciones con parámetros obligatorios u opcionales

# Índice

1. Vectores
2. Pares
3. Listas
4. Funciones con parámetros obligatorios u opcionales

## 1. Vectores

- Definición y ejemplos
- Predicado
- Igualdad
- Longitud
- Creación
- Acceso a un elemento
- Modificación de un elemento
- Conversión entre vector y lista
- Otros procedimientos
- Procedimientos especiales de racket/vector
- Ejemplos de uso de vectores

## 1. Vectores

- Definición y ejemplos
- Predicado
- Igualdad
- Longitud
- Creación
- Acceso a un elemento
- Modificación de un elemento
- Conversión entre vector y lista
- Otros procedimientos
- Procedimientos especiales de racket/vector
- Ejemplos de uso de vectores

## 1. Vectores

- **Definición**

  *# (objeto $_1$ objeto $_2$ … objeto $_n$)*

  donde *objeto $_i$* es un dato permitido por Scheme

  - **Ejemplos**

    *#(1 2 3)*

    *#("Pedro" "Ana" "Miguel")*

    *#(#(1 0 0) #(0 1 0) #(0 0 1))*

    *#(*

    *#("Juan Campos Luque" 12 #t)*

    *#("Ana Luna Prado" 14 #f)*

    *)*

# 1. Vectores

## ○ Observación

- Algunos intérpretes utilizan la siguiente notación

  *#3(1 2 3)*

  *#2(*

  *#3("Juan Campos Luque" 12 #t)*

  *#3("Ana Luna Prado" 14 #f)*

  *)*

  *#5(1 9) → #(1 9 9 9 9)*

  en la que se indica la longitud del vector

## 1. Vectores

- Definición y ejemplos
- Predicado
- Igualdad
- Longitud
- Creación
- Acceso a un elemento
- Modificación de un elemento
- Conversión entre vector y lista
- Otros procedimientos
- Procedimientos especiales de Racket/Vector
- Ejemplos de uso de vectores

## 1. Vectores

- **Predicado**

  *(vector? objeto)*

  - Será verdadero (*#t*) si y solamente si objeto es un vector.
  - En caso contrario, será falso (*#f*).

  - **Ejemplos**

    *(**vector**? #(1 2 3))* ➔ *#t*

    *(**vector**? '(1 2 3))* ➔ *#f*

    *(**vector**? #(#(1 2) #(3 4)))* ➔ *#t*

## 1. Vectores

- Definición y ejemplos
- Predicado
- Igualdad
- Longitud
- Creación
- Acceso a un elemento
- Modificación de un elemento
- Conversión entre vector y lista
- Otros procedimientos
- Procedimientos especiales de racket/vector
- Ejemplos de uso de vectores

# 1. Vectores

- **Igualdad**

  *(**equal?**   vector1 vector2)*

  - Será verdadero (*#t*) si  y solamente si los dos vectores tienen la misma **longitud** y las mismas **componentes**
  - En caso contrario, será falso (*#f*).
  - **Ejemplo**

    *(**equal?**  #(1 2 3)  (**vector** 1 2 3))* → *#t*

# 1. Vectores

- Definición y ejemplos
- Predicado
- Igualdad
- Longitud
- Creación
- Acceso a un elemento
- Modificación de un elemento
- Conversión entre vector y lista
- Otros procedimientos
- Procedimientos especiales de racket/vector
- Ejemplos de uso de vectores

## 1. Vectores

- **Longitud de un vector**

  *(vector-length objeto)*

  - Devuelve el **número** de componentes del vector

  - **Ejemplos**

    ***(vector-length*** *#(1 2 3))* ➔ *3*

    ***(vector-length*** *#(#(1 2) #(3 4))* ➔ *2*

## 1. Vectores

- Definición y ejemplos
- Predicado
- Igualdad
- Longitud
- Creación
- Acceso a un elemento
- Modificación de un elemento
- Conversión entre vector y lista
- Otros procedimientos
- Procedimientos especiales de racket/vector
- Ejemplos de uso de vectores

# 1. Vectores

- **Creación**
  - *vector constante*
  - *vector*
  - *vector-immutable*
  - *make-vector*

## 1. Vectores

- **Creación**
  - ***vector constante***

    *# (objeto $_1$ objeto $_2$ … objeto $_n$)*
  - **Ejemplos**

    *(define v #(1 2 3))*

    *v → #(1 2 3)*
  - **Observación**
    - No se pueden modificar los valores de las componentes

# 1. Vectores

- **Creación**
  - ***vector***

    *(vector objeto $_1$ objeto $_2$ … objeto $_n$)*

    ➔ *# (objeto $_1$ objeto $_2$ … objeto $_n$)*

  - **Ejemplos**

    *(**vector** 1 2 3)*

    ➔ *#(1 2 3)*

    *(**vector** "Pedro" "Ana" "Miguel")*

    ➔ *#("Pedro" "Ana" "Miguel")*

    *(**vector** (vector 1 2) (vector3 4))*

    ➔ *#(#(1 2) #(3 4))*

## 1. Vectores

- **Creación**
  - ***vector-immutable***

    *(vector-immutable objeto $_1$ objeto $_2$ … objeto $_n$)*

    ➔ *# (objeto $_1$ objeto $_2$ … objeto $_n$)*
  - **Observación**
    - No se pueden modificar los valores de las componentes
  - **Ejemplos**

    *(**vector-immutable** 1 2 3)* ➔ *#(1 2 3)*

    *(**vector-immutable** "lunes" "martes")*

    ➔ *#("lunes" "martes")*

# 1. Vectores

- **Creación**
  - ***make-vector***

    *(make-vector tamaño [objeto])*

    ➔ *# (objeto objeto … objeto )*

    - ***tamaño***: número natural
    - ***objeto***:
      - ✓ valor asignado a las componentes (opcional)
      - ✓ valor por defecto: 0

# 1. Vectores

- Creación
  - *make-vector*
  - Ejemplos

    *(make-vector 0) → #()*

    *(make-vector 3) → #(0 0 0)*

    *(make-vector 3 7) → #(7 7 7)*

    *(make-vector 2 #3(1)) → #(#(1 1 1) #(1 1 1))*

## 1. Vectores

- Definición y ejemplos
- Predicado
- Igualdad
- Longitud
- Creación
- Acceso a un elemento
- Modificación de un elemento
- Conversión entre vector y lista
- Otros procedimientos
- Procedimientos especiales de racket/vector
- Ejemplos de uso de vectores

# 1. Vectores

- **Acceso a un elemento de un vector**
  - **vector-ref**

    *(**vector-ref** v posición)*

    - v: vector
    - posición: número entero no negativo
  - **Ejemplos**

    *(**define** v #(0 1 2 3 4))*

    *(**vector-ref** v 0)* ➔ *0*

    *(**vector-ref** v 2)* ➔ *2*

    *(**vector-ref** v (- (**vector-length v**) 1))* ➔ *4*

# 1. Vectores

- **Acceso a un elemento de un vector**
  - **Ejemplos**

*(**define** v #(#(1 2 3) #(4 5 6) #(7 8 9))*

*(**vector-ref** (**vector-ref** v 0) 2) → 3*

*(**vector-ref** (**vector-ref** v 2) 1) → 8*

## 1. Vectores

- Definición y ejemplos
- Predicado
- Igualdad
- Longitud
- Creación
- Acceso a un elemento
- Modificación de un elemento
- Conversión entre vector y lista
- Otros procedimientos
- Procedimientos especiales de racket/vector
- Ejemplos de uso de vectores

# 1. Vectores

- **Modificación de un elemento**
  - vector-set!
  - vector-fill!

# 1. Vectores

- **Modificación de un elemento**
  - ***vector-set!***

    *(**vector-set!** v posición valor)*

    - ***v***: vector que se puede modificar
    - ***posición***: número entero no negativo
    - ***valor***: dato que se va a asignar a la componente del vector "*v*" indicada por "*posición*".
  - **Observación**
    - No devuelve ningún resultado

## 1. Vectores

- **Modificación de un elemento**
  - ***vector-set!***
  - **Ejemplos**

*(**define** v1 (vector 0 1 2 3 4))*

*v1*

*➔ #(0 1 2 3 4)*

*(**vector-set!** v1 1 9)*

*v1*

*➔ #(0 9 2 3 4))*

## 1. Vectores

- **Modificación de un elemento**
  - ***vector-set!***
  - **Ejemplos**

*(**define** v1 #(0 1 2 3 4))*

*(**vector-set!** v1 1 9)*

*v1 ➔ Error*

*(**define** v2 (**vector-immutable** 1 2 3)*

*(**vector-set!** v2 1 9) ➔ Error*

*(**vector-set!** #(1 2 3) 0 9) ➔ Error*

# 1. Vectores

- **Modificación de un elemento**
  - *vector-set!*
  - **Ejemplos**

```
(define v #(#(1 2 3) #(4 5 6) #(7 8 9)))

(vector-set! (vector-ref v 0) 2 0) → Error

(define v (vector (vector 1 2 3)
                  (vector 4 5 6)
                  (vector 7 8 9)
          )
)

(vector-set! (vector-ref v 0) 2 0)

v → #(#(1 2 0) #(4 5 6) #(7 8 9))
```

## 1. Vectores

- **Modificación de un elemento**
  - ***vector-fill!***

    *(**vector-fill!** v valor)*

    - ***v***: vector **existente** que se puede modificar
    - ***valor***: dato que se va a asignar a todas las componentes del vector "*v*"

  - **Observación**
    - No devuelve ningún resultado

## 1. Vectores

- **Modificación de un elemento**
  - ***vector-fill!***
  - **Ejemplos**

*(**define** v (**make-vector** 3))*

*v ➔ #(0 0 0)*

*(**vector-fill!** v 9)*

*v ➔ #(9 9 9)*

# 1. Vectores

- Definición y ejemplos
- Predicado
- Igualdad
- Longitud
- Creación
- Acceso a un elemento
- Modificación de un elemento
- Conversión entre vector y lista
- Otros procedimientos
- Procedimientos especiales de racket/vector
- Ejemplos de uso de vectores

# 1. Vectores

- **Conversión entre vector y lista**
  - **vector->list**
  - **list->vector**

## 1. Vectores

- **Conversión entre vector y lista**
  - **vector->list**

    *(**vector->list** v)*

    - ***v***: vector
    - Devuelve una lista con los elementos del vector
  - **Ejemplos**

    *(**vector->list** #(1 2 3))* ➔ *(1 2 3)*

    *(**vector->list** #(#(1 2) #(3 4)))* ➔ *(#(1 2) #(3 4))*

# 1. Vectores

- **Conversión entre vector y lista**
  - **list->vector**

    *(list->vector lista)*

    - Devuelve un vector con los elementos de la *lista*
  - **Ejemplos**

    ***(list->vector** '(1 2 3)) → #(1 2 3)*

    ***(list->vector** '((1 2) (3 4))) → #((1 2) (3 4))*

# 1. Vectores

- Definición y ejemplos
- Predicado
- Igualdad
- Longitud
- Creación
- Acceso a un elemento
- Modificación de un elemento
- Conversión entre vector y lista
- Otros procedimientos
- Procedimientos especiales de racket/vector
- Ejemplos de uso de vectores

# 1. Vectores

- **Otros procedimientos**
  - build-vector
  - vector->values

# 1. Vectores

- **Otros procedimientos**
  - ***build-vector***

    *(build-vector  n procedimiento)*

    - Devuelve un vector con los resultados del aplicar el **procedimiento** a los números naturales desde 0 hasta n-1.
  - **Ejemplos**

    *(**build-vector** 5 +) ➔ #(0 1 2 3 4)*

    *(**build-vector** 5 sqrt) ➔ #(0  1  1.42…   1.73…   2)*

    *(**build-vector** 5 (**lambda** (x) (* x x)))*

    *➔ #(0 1 4 9 16)*

# 1. Vectores

- **Otros procedimientos**
  - ***vector->values***

    *(vector->values v [inicial final])*

    - Devuelve los elementos del vector *v* que ocupan las posiciones situadas entre inicial (inclusive) y final (exclusive):
    - ***v***: vector
    - ***inicial*** y ***final***: números enteros no negativos
  - **Ejemplo**

    *(**vector->values** #(0 1 2 3 4 5 6 7) 3 5)*

    ➔ *3*

    *4*

## 1. Vectores

- **Otros procedimientos**
  - *vector->values*
  - **Ejemplo**

```
(define v  #(0 1 2 3 4 5 6 7))

(call-with-values
   (lambda () (vector->values v 3 5))
  +
 )
 → 7
```

## 1. Vectores

- **Otros procedimientos (digresión)**
  - *(**call-with-values** generador receptor)*
    - *generador*: procedimiento, función u operador de Scheme que **devuelva valores o resultados**
    - *receptor*: procedimiento, función u operador de Scheme
    - **Descripción**
      - Invoca a *generador*, que genera unos valores que son usados como argumentos de *receptor*

# 1. Vectores

- **Otros procedimientos (digresión)**
  - *(call-with-values generador receptor)*
    - Ejemplos

*(call-with-values * + ) → 1*

*(call-with-values (lambda () (values 2 3 4)) *)*
*→ 24*

*(call-with-values (lambda () (values 2 3 4))*
*vector)*

*→ #(2 3 4)*

## 1. Vectores

- Definición y ejemplos
- Predicado
- Igualdad
- Longitud
- Creación
- Acceso a un elemento
- Modificación de un elemento
- Conversión entre vector y lista
- Otros procedimientos
- Procedimientos especiales de racket/vector
- Ejemplos de uso de vectores

## 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - *vector-copy*
  - *vector-map*
  - *vector-append*
  - *vector-filter, vector-filter-not*
  - *vector-count*
  - *vector-argmin, vector-argmax*
  - *vector-member, vector-memv, vector-memq*

*(require racket/vector)*

# 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - **Otros**

    *(require racket/vector)*

    - *vector-set*!*
    - *vector-take*, *vector-take-right*
    - *vector-drop*, *vector-drop-right*
    - *vector-split-at*, *vector-split-at-right*
  - Más información
    - http://docs.racket-lang.org/reference/vectors.html

# 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - ***vector-copy***

    *(**vector-copy** v [inicial [final]] )*

    - Crea un nuevo vector
      - ✓ de longitud inicial – final
      - ✓ con los elementos del vector "v" que ocupan las posiciones comprendidas entre inicial (inclusive) y final (exclusive)

# 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - ***vector-copy***
  - **Ejemplos**

    *(**vector-copy** #(1 2 3 4))* ➔ *#(1 2 3 4)*

    *(**vector-copy** #(1 2 3 4) 1)* ➔ *#( 2 3 4)*

    *(**vector-copy** #(1 2 3 4) 2 3)* ➔ *#(3)*

## 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - ***vector-map***

    (***vector-map*** *procedimiento* $v_1$ $[v_2 \ldots v_n]$ )

    - **Aplica** el procedimiento a los elementos de los vectores desde el primero hasta el último.
    - Devuelve otro vector con los resultados del procedimiento
    - **Restricciones**
      - ✓ Debe haber tantos vectores como argumentos requiera el procedimiento
      - ✓ Todos los vectores deben tener el mismo número de elementos

# 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - ***vector-map***
  - **Ejemplos**

    *(**define** v1 #(1 2 3))*

    *(**define** v2 #(5 6 7)*

    *(**vector-map** sqrt v1) ➔ #(1 1.41… 1.73…)*

    *(**vector-map** + v1 v2 v1) ➔ #(7 10 13)*

    *(**vector-map** (**lambda** (x) (* x x)) v1) ➔ #(1 4 9)*

    *(**vector-map** (**lambda** (x y) (* x y)) v1 v2)*

    *➔ (5 12 21)*

## 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - ***vector-append***

    (***vector-append*** $v_1$ $[v_2 \ldots v_n]$ )

    - Crea un nuevo vector con la **concatenación** de los elementos de los vectores
  - **Ejemplo**

```
(define v1 #(1 2 3))
(define v2 #(5 6 7))
(vector-append v1 v2) → #(1  2 3 5 6 7)
```

# 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - ***vector-filter***

    *(**vector-filter** predicado v)*

    - Crea un **nuevo vector** con los elementos del vector que hacen **verdadero** el predicado
  - **Ejemplo**

    *(**define** v #(1 2 3))*

    *(**vector-filter odd?** v) ➔ #(1 3)*

## 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - ***vector-filter-not***

    *(**vector-filter-not** predicado v)*

    - Crea un **nuevo vector** con los elementos del vector que **no** hacen verdadero el predicado
  - **Ejemplo**

    *(**define** v #(1 2 3))*

    *(**vector-filter-not** odd? v) ➔ #(2)*

## 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - ***vector-count***

    (***vector-count*** *predicado* $v_1$ [$v_2$ … $v_n$ ] )

    - Devuelve el **número de elementos** de los vectores (tomados en paralelo) que no hacen falso el predicado
  - **Ejemplo**

    *(**define** v1 #(1 2 3))*

    *(**define** v2 #(5 6 7))*

    *(**vector-count** odd? v1) → 2*

    *((**vector-count** = v1 v2) → 0*

## 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - ***vector-argmin***

    (***vector-argmin*** *procedimiento v)*

    - Devuelve el **primer** elemento del vector "v" que **minimiza** el procedimiento.
  - **Ejemplo**

    ***(vector-argmin vector-length***

    *#(#("Juan") #("Ana" "Pablo") #("Luis")))*

    ➔ *#("Juan")*

# 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - ***vector-argmax***

    *(**vector-argmax** procedimiento v)*

    - Devuelve el **primer** elemento del vector "v" que **maximiza** el procedimiento.
  - **Ejemplo**

    ***(vector-argmax   vector-length***

    *#(#("Juan") #("Ana" "Pablo") #("Luis")))*

    ➔ *#("Ana" "Pablo")*

## 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - ***vector-member***

    *(**vector-member** valor v)*

    - Devuelve el **índice** del **primer** elemento del vector "v" que es igual que el ***valor*** indicado usando el predicado ***equal***?
    - En caso contrario, devuelve falso (*#f*)
  - **Ejemplo**

    *(**define** v #(1 0 3 2))*

    *(**vector-member** 2) ➔ 3*

    *(**vector-member** 9) ➔ #f*

# 1. Vectores

- **Procedimientos especiales de Racket/Vector**
  - ***vector-memv, vector-memq***
    - Tienen un funcionamiento similar a ***vector-member***
    - Utilizan el predicado ***eqv?*** y ***eq?***, respectivamente

## 1. Vectores

- Definición y ejemplos
- Predicado
- Igualdad
- Longitud
- Creación
- Acceso a un elemento
- Modificación de un elemento
- Conversión entre vector y lista
- Otros procedimientos
- Procedimientos especiales de racket/vector
- Ejemplos de uso de vectores

# 1. Vectores

- **Ejemplos de uso de vectores**
  - *Máximo de un vector: versión iterativa*

```
(define (maximo-vector v)
  (do
     (
      (n          (- (vector-length v) 1))
      (i          1                      (+ i 1))
      (maximo (vector-ref v 0))  (if (< maximo (vector-ref v i))
                                     (vector-ref v i)
                                      maximo)
       )
      )
    ;; Condición de salida
    ((> i n) maximo)
    ;; No hay cuerpo del bucle
    )
  )
;; Ejemplo de llamada a la función
(maximo-vector   #(1 2 3 4 3 2 1)) → 4
```

## 1. Vectores

- **Ejemplos de uso de vectores**
  - *Máximo de un vector: usando vector-argmax*

*(require racket/vector)*

*(**vector-argmax** (**lambda** (x) x)  #(1 2 3 4 3 2 1))* → *4*

# 1. Vectores

- **Ejemplos de uso de vectores**

```
(define (ver-vector v)
 (do
    (
     (i          0                  (+ i 1))
     (longitud (vector-length v))
     )
   ;; Condición de salida
   ((>= i longitud) (newline))
   ;; Cuerpo del bucle
   (display " ")
   (display (vector-ref v i))
   (display " ")
   )
 )
```

## 1. Vectores

- **Ejemplos de uso de vectores**

```
(define (ver-matriz m)
  (do
     (
      (i        0                       (+ i 1))
      (filas    (vector-length m))
     )
    ;; Condición de salida
    ((>= i filas) (newline))
    ;; Cuerpo del bucle
     (newline)
     (ver-vector (vector-ref m i))
    )
  )
```

# 1. Vectores

- **Ejemplos de uso de vectores**
  - Véase el fichero *producto-matrices.rkt*

# Índice

1. Vectores
2. Pares
3. Listas
4. Funciones con parámetros obligatorios u opcionales

## 2. Pares

- Definición
- Creación
- Predicado
- Acceso
- Modificación

## 2. Pares

- Definición
- Creación
- Predicado
- Acceso
- Modificación

## 2. Pares

- **Definición**
  - **Par punteado o *dotted pair***
    - Registro compuesto por dos atributos denominados *car* y *cdr*

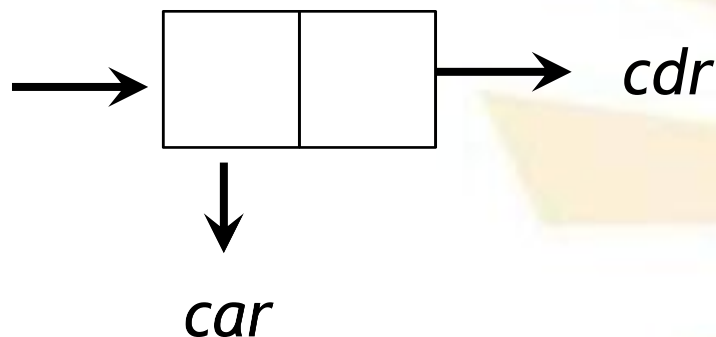
      *(car . cdr)*

    - **Reseña histórica**
      - ✓ Los nombres *car* y *cdr* provienen de su implementación en un computador IBM 704:
      - ✓ *car*: *content of the address portion of a register.*
      - *cdr*: *content of the decrement portion of a register.*

## 2. Pares

- **Definición**
  - **Par punteado o *dotted pair***
    - Registro compuesto por dos atributos denominados *car* y *cdr*

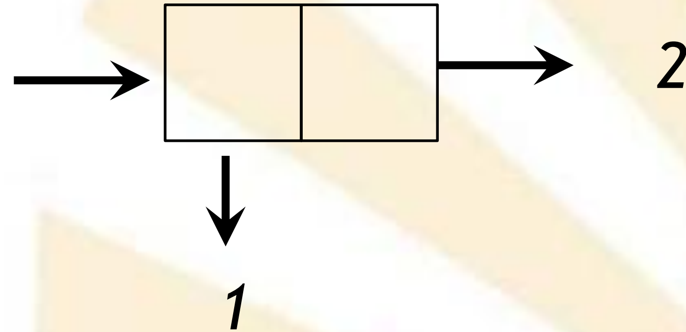
      *(car . cdr)*
  - **Representación gráfica**



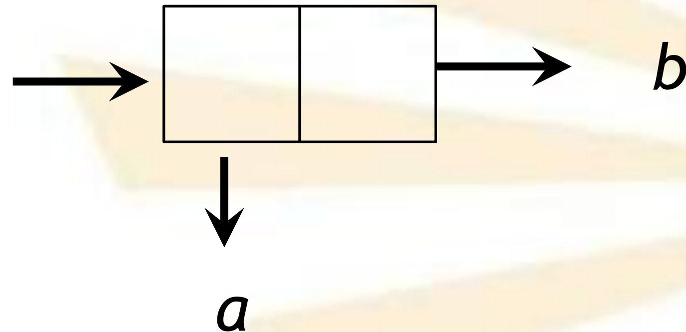
# 2. Pares

- **Definición**
  - **Ejemplos**

*( 1 . 2)*




*(a . b)*



## 2. Pares

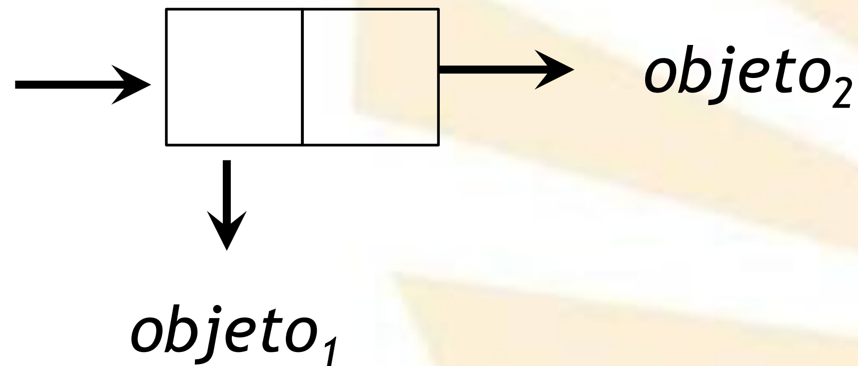
- Definición
- Creación
- Predicado
- Acceso
- Modificación

# 2. Pares

- **Creación**

(**cons** *objeto*$_1$ *objeto*$_2$)

➔ (*objeto*$_1$ . *objeto*$_2$)

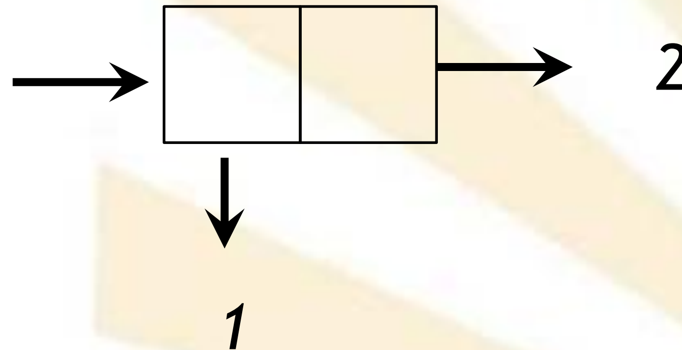


## 2. Pares

- **Creación**
  - **Ejemplo (1/8)**

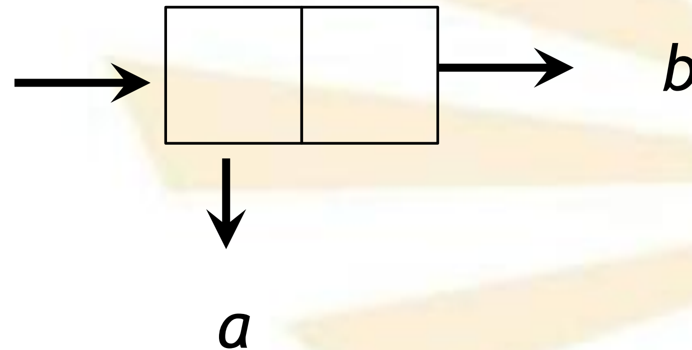
*(cons 1 2)*

➔ *(1 . 2 )*




*(cons ‘a ‘b)*

➔ *(a . b )*

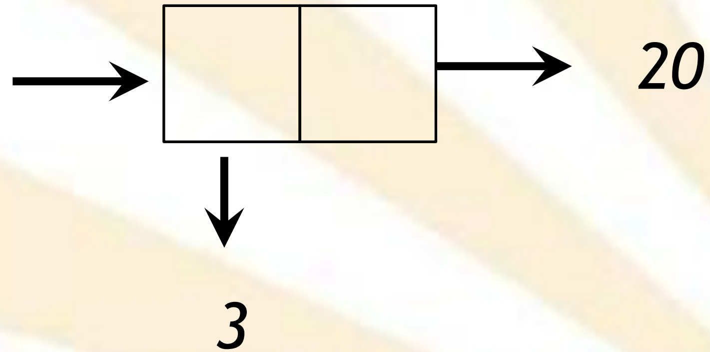


# 2. Pares

- **Creación**
  - **Ejemplo (2/8)**

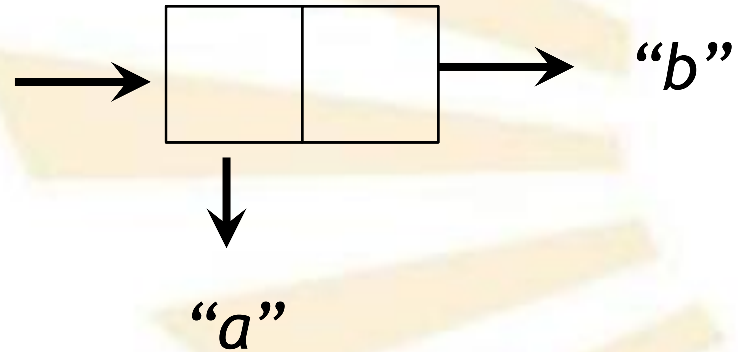
*(cons 3 (* 4 5))*

➔ *(3 . 20)*




*(cons "a" "b")*

➔ *("a" . "b" )*



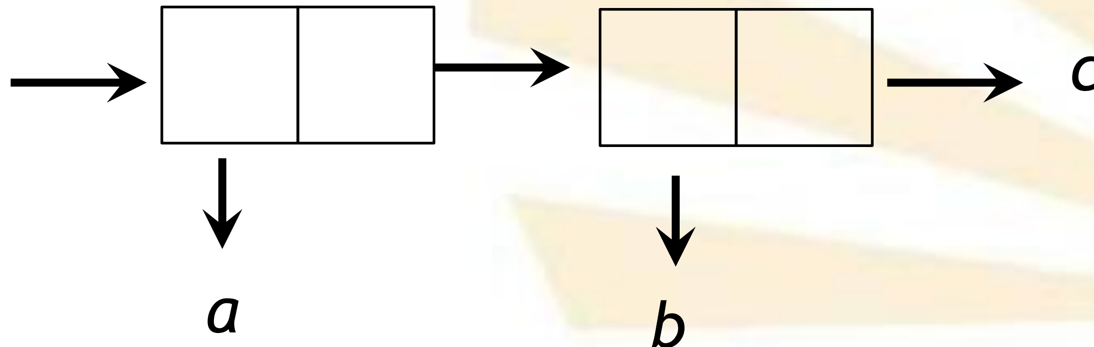
# 2. Pares

- **Creación**
  - **Ejemplo (3/8)**

    ***(cons** ‘a **(cons** ‘b ‘c))*

    ➔ *(a b . c)*

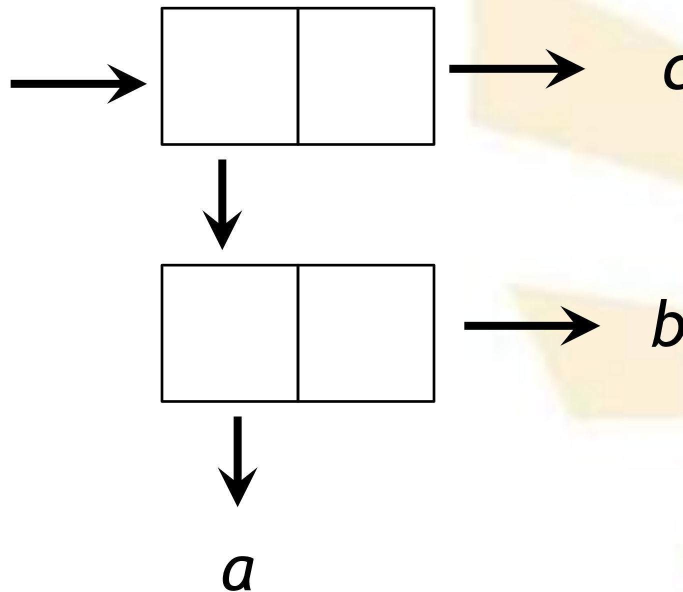
    Equivalente a: *(a . (b . c ))*



## 2. Pares

- **Creación**
  - **Ejemplo (4/8)**

    *(cons (cons ‘a ‘b) ‘c)*
    ➔ *((a . b) . c )*

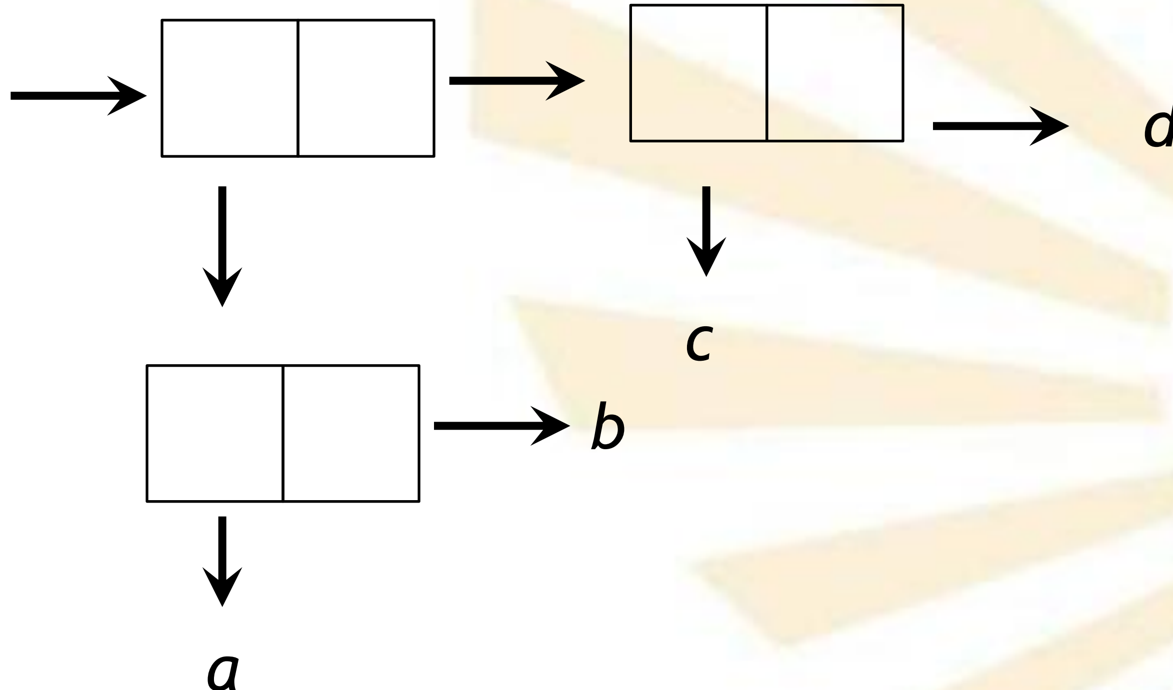


## 2. Pares

- **Creación**
  - **Ejemplo (5/8)**

*(**cons** (**cons** ‘a ‘b) (**cons** ‘c ‘d))*
➔ *((a . b) c . d)*

Equivalente a: *((a . b) . (c . d))*

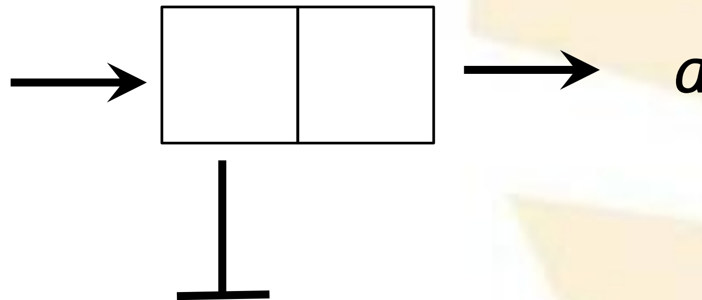


## 2. Pares

- **Creación**
  - **Ejemplo (6/8)**

*(cons () ‘a )*

➔ *( () . a)*




***(): lista vacía***

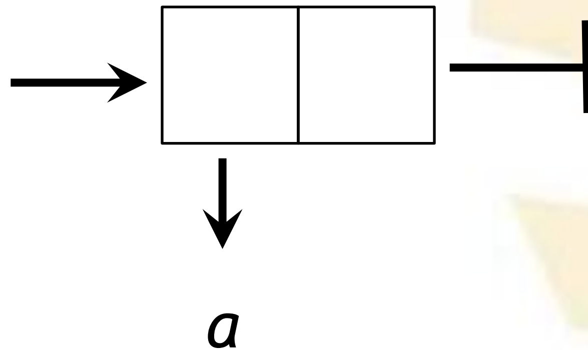
## 2. Pares

- **Creación**
  - **Ejemplo (6/8)**

    *(cons 'a () )*

    ➔ *(a)*

    Equivalente a: *(a . () )*




***Es una lista***

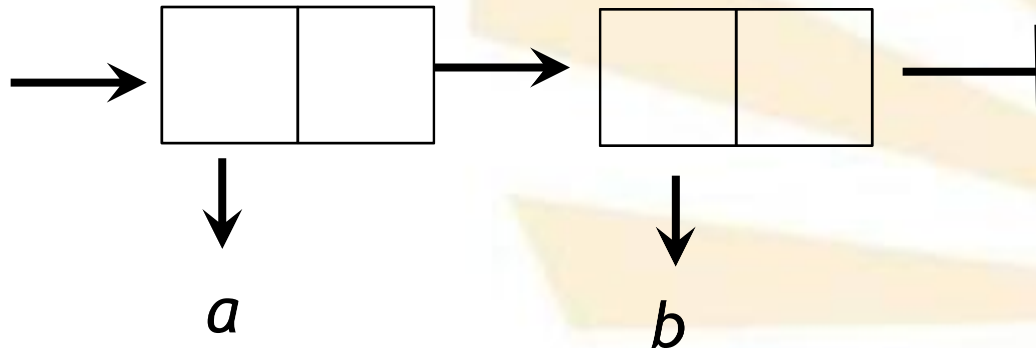
## 2. Pares

- **Creación**
  - **Ejemplo (7/8)**

*(cons ‘a (cons ‘b ())) )*

➔ *(a b)*

Equivalente a: *(a . (b . ())) )*



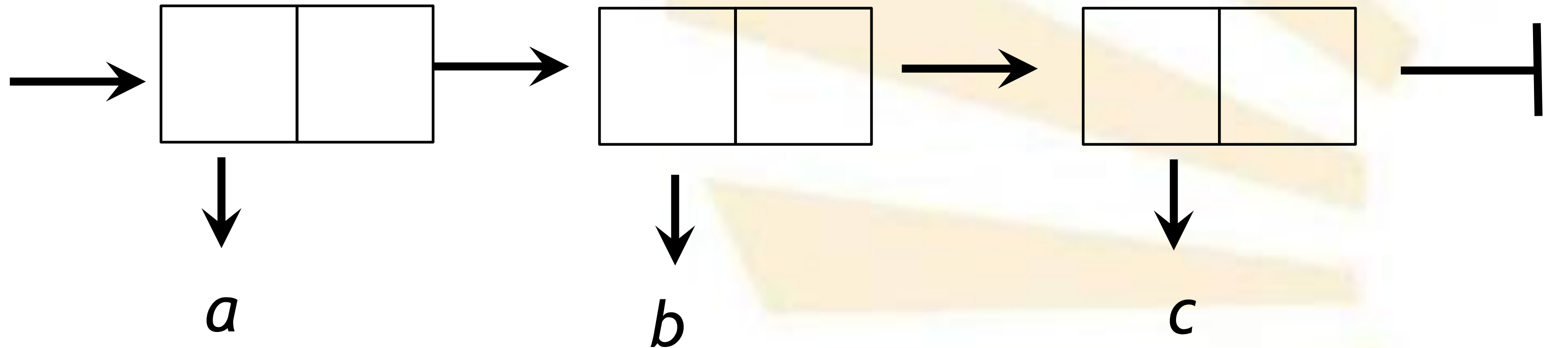

***Es una lista***

## 2. Pares

- **Creación**
  - **Ejemplo (8/8)**

    *(**cons** ‘a (**cons** ‘b (**cons** ‘c () )))*
    ➔ *(a b c)*

    Equivalente a: *(a . (b . (c . ()) ) )*




***Es una lista***

# 2. Pares

- Definición
- Creación
- Predicado
- Acceso
- Modificación

## 2. Pares

- **Predicado**
  - ***pair?***

    ***(pair? objeto)***

    - Devuelve verdadero (*#t*) si el objeto es un par
    - En caso contrario; devuelve falso (*#f*)
  - **Ejemplos**

    *(**pair**? ())* ➔ *#f*

    *(**pair**? '(a . b))* ➔ *#t*

    *(**pair**? (**cons** 'a 'b))* ➔ *#t*

## 2. Pares

- Definición
- Creación
- Predicado
- Acceso
- Modificación

## 2. Pares

- Acceso
  - *car*
  - *cdr*

## 2. Pares

- **Acceso**
  - ***car***

    *(**car** par)*

    - Devuelve el primer atributo del par
  - ***cdr***

    *(**cdr** par)*

    - Devuelve el segundo atributo del par

# 2. Pares

- **Acceso**
  - **Ejemplos**

*(**define** par (**cons** ‘a ‘b))*

*par ➔ (a . b)*

*(**car** par) ➔ a*

*(**cdr** par) ➔ b*

## 2. Pares

- Definición
- Creación
- Predicado
- Acceso
- Modificación

## 2. Pares

- **Modificación**
  - ***set-car!***
  - ***set-cdr!***

## 2. Pares

- **Modificación**
  - ***set-car!***

    *(**set-car!** par valor)*

    - Modifica el primer atributo del par
    - No devuelve ningún resultado
  - ***set-cdr!***

    *(**set-cdr!** par valor)*

    - Modifica el segundo atributo del par
    - No devuelve ningún resultado

## 2. Pares

- **Modificación**
  - **Ejemplos**

*(**define** par (**cons** ‘a ‘b))*

*par ➔ (a . b)*

*(**set-car!** par ‘c)*

*par ➔ (c . b)*

*(**set-cdr!** par d)*

*par ➔ (c . d)*

# Índice


1. Vectores
2. Pares
3. Listas
4. Funciones con parámetros obligatorios u opcionales

# 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

# 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

# 3. Listas

- **Definición**
  - Definición recursiva
  - Definición literal

# 3. Listas

- **Definición**
  - **Definición recursiva**
    - La lista vacía () es una lista
    - Si "l" es una lista entonces cualquier **par** cuyo campo ***cdr*** sea "l" también es una lista.

      *(objeto . l)*

## 3. Listas

- **Definición**
  - **Definición recursiva**
    - **Comentarios**
      - ✓ Una **lista** es un encadenamiento de pares que **finaliza** en la lista vacía
      - ✓ Un encadenamiento de pares que no finaliza en la lista vacía se denomina **“lista impropia”**

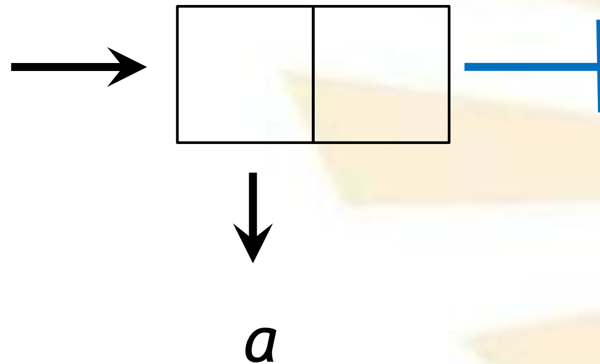
# 3. Listas

- **Definición**
  - **Definición literal**
    - Una **lista** está compuesta por
      - ✓ un paréntesis izquierdo,
      - ✓ una sucesión de átomos y listas
      - ✓ y de un paréntesis derecho.

- **Definición**
  - **Ejemplos**

*()*

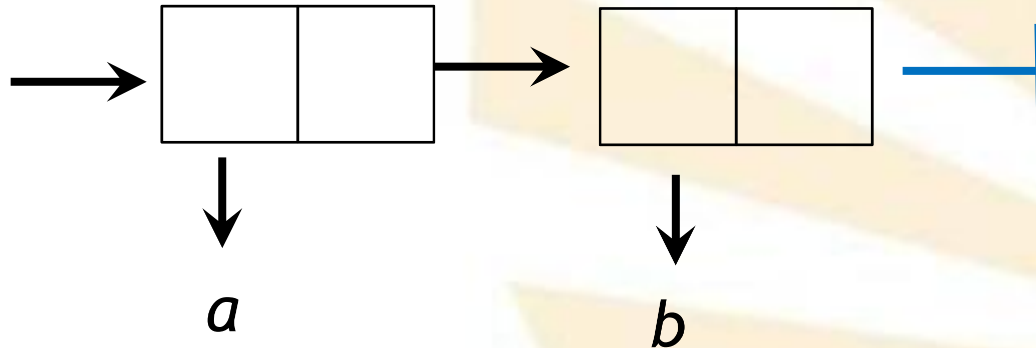
$(a \;.\; ()) \equiv (a)$



# 3. Listas

- **Definición**
  - **Ejemplos**

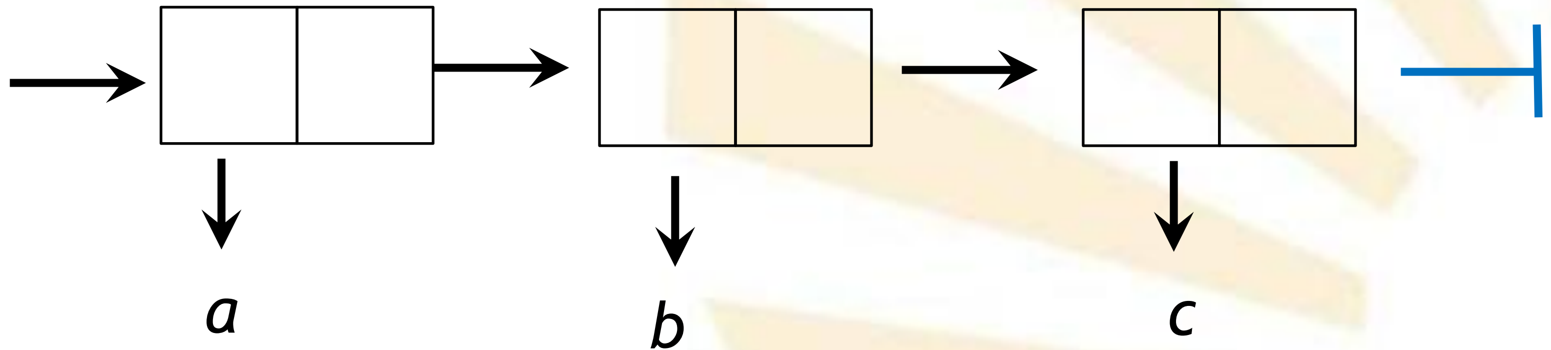
$$(a\ .\ (b\ .\ ()))\ \equiv (a\ b)$$



# 3. Listas

- **Definición**
  - **Ejemplos**

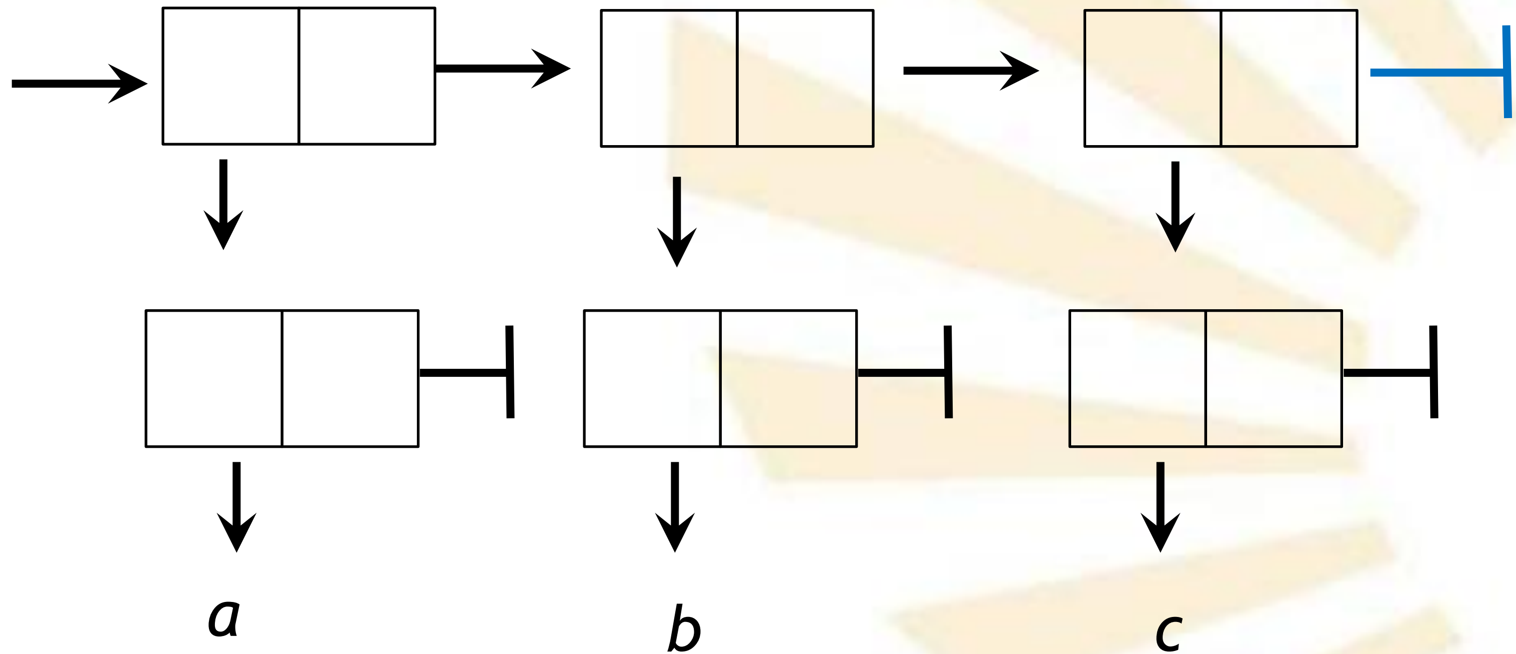
$$(a\ .\ (b\ .\ (c\ .\ ())))\equiv(a\ b\ c)$$



# 3. Listas

- **Definición**
  - **Ejemplos**

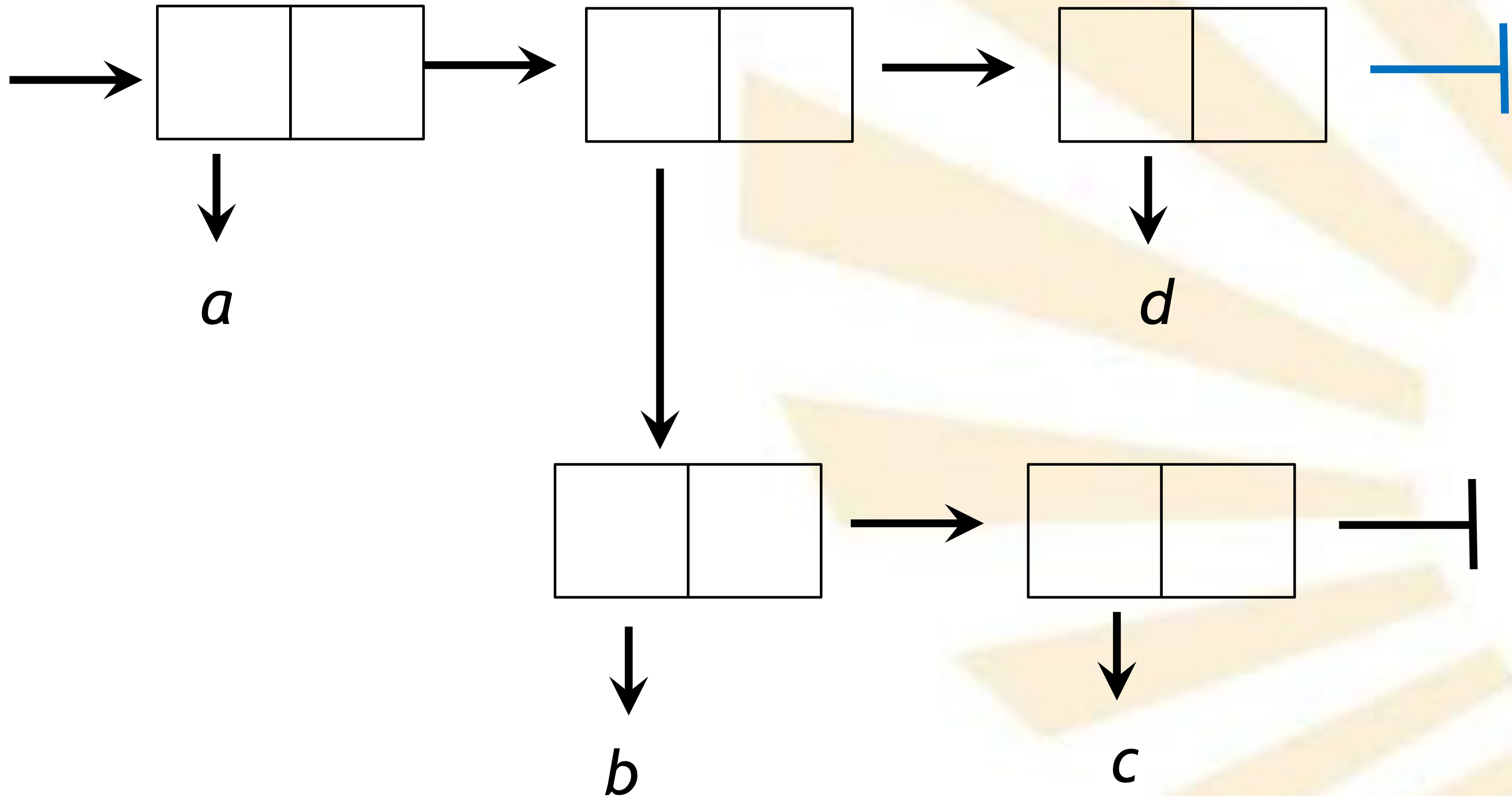
$$( (a\ .\ ())\ .\ ((b\ .\ ())\ .\ ((c\ .\ ()))))) \equiv (\ (a)\ (b)\ (c)\ )$$



# 3. Listas

- **Definición**
  - **Ejemplos**

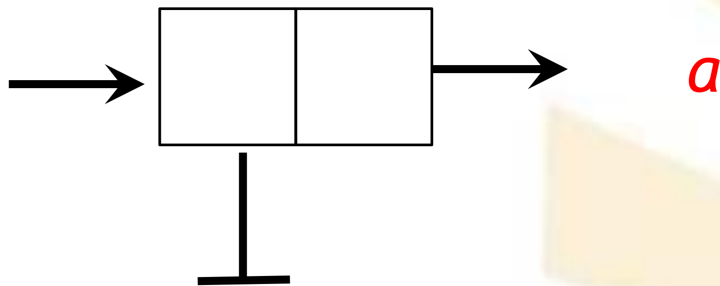
$(a\ .\ ((b\ .\ (c\ .\ ()))\ .\ (d\ .\ ())))\equiv(a\ (b\ c)\ d)$



# 3. Listas

- **Definición**
    - **Ejemplos de pares que no son listas**

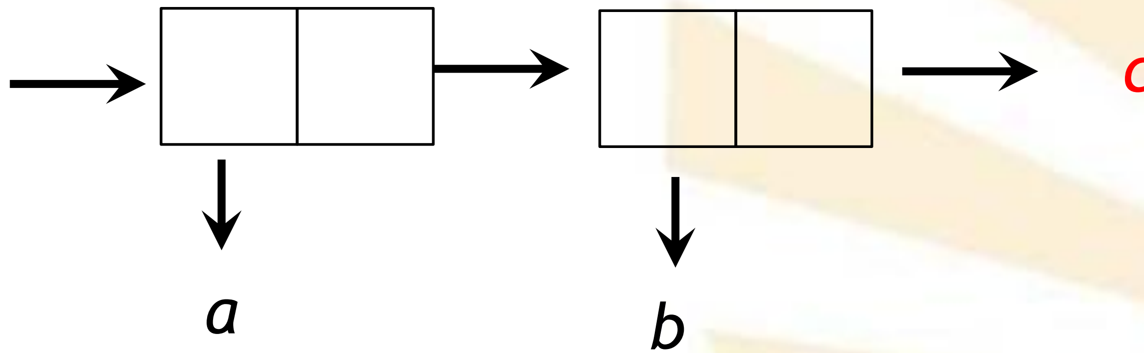
*( () . a)*



# 3. Listas

- **Definición**
  - **Ejemplos de pares que no son listas**

$$( a \; . \; (b \; . \; c)) \equiv ( a \; b \; . \; c)$$



## 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

# 3. Listas

- Creación
  - *cons*
  - *list*
  - *list**
  - *append*
  - *reverse*
  - *build-list*

## 3. Listas

- **Creación**
  - ***cons***

    *(**cons** objeto lista)*

  - ***Ejemplos***

    *(**cons** 1 ())*
    *➔ (1)*

    *(**cons** 'a (list ‘b c d) )*
    *➔ (a b c d)*

## 3. Listas

- **Creación**
  - ***list***

(*list* *[objeto $_1$ objeto $_2$ … objeto $_n$ ]* )

➔ *(objeto $_1$ objeto $_2$ … objeto $_n$ )*

que es equivalente a

*(objeto$_1$ . (objeto$_2$ . ( ... (objeto$_n$ . ()) ...)))*

# 3. Listas

- **Creación**
  - ***list***
  - **Ejemplos**

```
(list)
        → ()
(list 1 2 3)
        → (1 2 3)
(list "Pedro" "Ana" "Miguel")
        → ("Pedro" "Ana" "Miguel")
(list (list 1 2) (list 3 4))
        → ((1 2) (3 4))
```

# 3. Listas

- **Creación**
  - ***list***
  - **Ejemplos**

```
(list 'a)
        → (a)

(list 'a 'b)
        → (a b)

(list 'a 'b 'c)
        → (a b c)

(list 'a (* 3 4) "c")
        → (a 12 "c")
```

# 3. Listas

- **Creación**
  - ***list****

(*list** *[objeto* $_1$ *objeto* $_2$ … *objeto* $_n$ *]* *cola* )

➔ (*objeto* $_1$ *objeto* $_2$ … *objeto* $_n$ . *cola*)

- Funcionamiento similar a ***list***, pero el último argumento es usado como cola de la lista creada.

# 3. Listas

- **Creación**
  - ***list****
  - **Ejemplos**

*(**list*** **1 2**)*

→ *(1 . 2)*

No es una lista

*(**list*** 1 2 (**list** 3 4))*

→ *(1 2 3 4)*

# 3. Listas

- **Creación**
  - ***append***

    *(**append** lista$_1$ [lista$_2$ … lista$_n$ ] )*

    - Crea una nueva lista con la concatenación de los elementos de las listas *lista$_1$ [lista$_2$ … lista$_n$]*

# 3. Listas

- Creación
  - ***append***
  - Ejemplos

    *(**define** lista1 ‘(1 2 3))*

    *(**define** lista2 ‘(5 6 7))*

    *(**append** lista1 lista2) ➔ (1 2 3 5 6 7)*

    *(**append** ‘((a 1) (b 2)) ‘((c 3) (d 4)))*

    *➔ ‘((a 1) (b 2) (c 3) (d 4)))*

# 3. Listas

- Creación
  - *append*
  - Ejemplos

*(**append** '(a) '(b))*
→ *(a b)*

*(**append** '((a)) '((b)))*
→ *( (a) (b) )*

*(**append** '(a) '( (b) c d))*
→ *(a (b) c d)*

*(**append** '(a (b)) '(c d))*
→ *(a (b) c d)*

# 3. Listas

- **Creación**
  - ***reverse***

    *(**reverse** lista)*

    - Crea una nueva lista con los elementos de la *lista* dispuestos en orden inverso
    - Solamente invierte los elementos del primer nivel

  - **Ejemplos**

    *(**reverse** '(a b c))*
    ➔ *(c b a)*

    *(**reverse** '((a b) (c) (d e)))*
    ➔ *((d e) (c) (a b))*

- **Creación**
  - ***build-list***

    *(**build-list** n procedimiento)*

    - Crea una nueva lista compuesta por los resultados obtenidos al aplicar el **procedimiento** a los valores comprendidos entre 0 y n-1
  - **Ejemplos**

*(**build-list** 4 sqrt)*

➔ *(0 1 1.4142135623730951 1.7320508075688772)*

## 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

## 3. Listas

- **Longitud**
  - ***length***

    *(**length** lista)*

    - Devuelve el número de elementos de la *lista*
  - **Ejemplos**

    *(**length** ())*
    ➔ *0*

    *(**length** '(a b c))*
    ➔ *3*

    *(**length** '( (a b c) (d e f))*
    ➔ *2*

## 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

# 3. Listas

- **Predicados**
  - ***list?***
  - ***null?***

# 3. Listas

- **Predicados**
  - ***list?***

    *(**list?** objeto)*

    - Indica si un objeto es o no una lista
  - **Ejemplos**

    *(**list?** ()) ➔ #t*
    *(**list?** '(a)) ➔ #t*
    *(**list?** '(a (b c) d)) ➔ #t*

    *(**list?** '(() . a)) ➔ #f*

    *(**define** lista (list 'a 'b 'c 'd))*
    *(**list?** lista) ➔ #t*

# 3. Listas

- **Predicados**
  - ***null?***

    *(**null?** objeto)*

    - Indica si un objeto es o no la lista vacía
  - **Ejemplos**

```
(null? ()) → #t
(null? '(a)) → #f
(null? '(a (b c) d)) → #f

(define lista (list 'a 'b 'c 'd))
(null? lista) → #f
```

## 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

# 3. Listas

- **Igualdad**

  *(**equal?** lista1 lista2)*

  - Será verdadero (*#t*) si y solamente si las dos listas tienen la misma longitud y los mismos elementos
  - En caso contrario, será falso (*#f*).

  - **Ejemplo**

    *(**equal?** '(1 2 3) (**list** 1 2 3))* ➔ *#t*

# 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

# 3. Listas

- Acceso
  - *car y cdr*
  - *list-ref*
  - *list-tail*

## 3. Listas

- **Acceso**
  - *car* y *cdr*
  - Ejemplos

```
(define lista1 (list 'a 'b 'c 'd))
(define lista2 (list 'e))

(car lista1) ➔ a
(cdr lista1) ➔ (b c d)

(car lista2) ➔ e
(cdr lista2) ➔ ()
```

# 3. Listas

- Acceso
  - *car* y *cdr*
  - **Ejemplos: combinación de *car* y *cdr***

    *lista1 ➔ (a b c d)*

    - Segundo elemento de la lista

      *(**car** (**cdr** lista1)) ➔ b*

    - Tercer elemento

      *(**car** (**cdr** (**cdr** lista1))) ➔ c*

    - *Último elemento*

      *(**car** (**reverse** lista1)) ➔ d*

# 3. Listas

- **Acceso**
  - ***car*** **y** ***cdr***
  - **Ejemplos**

    **Abreviaturas de combinación de *car* y *cdr***

    - Segundo elemento de la lista

      *(**cadr** lista1)* ➔ *b*

    - Tercer elemento de la lista

      *(**caddr** lista1)* ➔ *c*

# 3. Listas

- Acceso
  - *car* y *cdr*
  - Ejemplos

Abreviaturas de combinación de *car* y *cdr*

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

*(**cddr** lista)*
➔ *((f g))*
*(**cdar** lista)*
➔ *(b)*
*(**cadadr** lista)*
➔ *d*

# 3. Listas

- **Acceso**
  - ***list-ref***

    *(**list-ref** lista n)*

    - Devuelve el elemento de la lista que ocupa la posición “n”
      - ✓ n $\in$ {0, 1, ..., longitud de la lista -1}

# 3. Listas

- Acceso
  - ***list-ref***
  - Ejemplos

*(**define** lista (**list** 'a 'b 'c 'd))*

*lista ➔ (a b c d)*

*(**list-ref** lista 0) ➔ a*
*(**list-ref** lista 3) ➔ d*

*(**list-ref** lista (- (**length** lista) 1)) ➔ d*

*(**list-ref** lista 4) ➔ Error*

# 3. Listas

- **Acceso**
  - ***list-ref***
  - **Definición equivalente**

```
(define (list-ref-equivalente  l n)
    (if (zero? n)
        (car l)
        (list-ref-equivalente (cdr l) (- n 1))
    )
)
```

Función recursiva de cola

# 3. Listas

- **Acceso**
  - ***list-tail***

    *(**list-tail** lista n)*

    - Devuelve la lista que resulta de **eliminar** los "n" primeros elementos
      - ✓ n $\in$ {0, 1, ..., longitud de la lista}

# 3. Listas

- **Acceso**
  - ***list-tail***
  - **Ejemplos**

(***define*** *lista* (***list*** *0 1 2*))

(***list-tail*** *lista 0*) ➔ *(0 1 2)*

(***list-tail*** *lista 1*) ➔ *(1 2)*

(***list-tail*** *lista 2*) ➔ *(2)*

(***list-tail*** *lista 3*) ➔ *()*

(***list-tail*** *lista 4*) ➔ *Error*

# 3. Listas

- **Acceso**
  - ***list-tail***
  - **Definición equivalente**

```
(define (list-tail-equivalente l n)
   (if (zero? n)
         l
         (list-tail-equivalente (cdr l) (- n 1))
   )
)
```

Función recursiva de cola

# 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

# 3. Listas

- **Modificación**
  - *set-car!*
  - *set-cdr!*

# 3. Listas

- **Modificación**
  - ***set-car!***

    *(**set-car!** lista objeto)*

    - Modifica el primer elemento de la lista
  - **Ejemplos**

    *(**define** lista (**list** 0 1 2))*

    *lista ➔ ( 0 1 2)*

    *(**set-car**! lista 'a)*

    *lista ➔ ( a  1 2)*

# 3. Listas

- **Modificación**
  - ***set-car!***

    *(**set-car!** (**list-tail** lista n) objeto)*

    - Modifica el elemento de la lista que ocupa el lugar "n"

## 3. Listas

- **Modificación**
  - ***set-car!***

    *(**set-car!** (**list-tail** lista n) objeto)*

  - **Ejemplos**

    *(**define** lista (**list** ‘a ‘b ‘c))*

    *(**set-car!** (**list-tail** lista 0) 0)*

    *lista ➔ (0 b c)*

    *(**set-car!** (**list-tail** lista 1) 1)*

    *lista ➔ (0 1 c)*

    *(**set-car!** (**list-tail** lista 2) 2)*

    *lista ➔ (0 1 2)*

## 3. Listas

- **Modificación**
  - ***set-cdr!***

    *(**set-cdr!** lista nueva-cola)*

    - Modifica la cola de la lista
    - nueva-cola **tiene** que ser una **lista**
  - **Ejemplos**

    *(**define** lista (**list** 0 1 2))*

    *lista*

    *(**set-cdr!** lista (**list** 'c 'd))*

    *lista → (0 c d)*

## 3. Listas

- **Modificación**
  - ***set-cdr!***
  - **Ejemplo**

    *(**define** lista (**list** 0 1 2))*

    *lista*

    *(**set-cdr!** lista ‘b)*

    *lista → (0 . b)*

No es una lista

## 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

## 3. Listas

- **Búsqueda**
  - *member, memv, memq, memf*
  - *find*
  - *assoc, assv, assq, assf*

## 3. Listas

- **Búsqueda**
  - ***member***

    *(**member** objeto lista)*

    - Localiza el **primer elemento** de la lista que es igual al *objeto* buscado usando el predicado ***equal?***
    - Si el objeto **pertenece** a la lista, devuelve **el resto** de la lista a partir del objeto buscado.
    - En caso contrario, devuelve *#f*
    - **Observación**
      - ✓ Si el objeto aparece **varias veces** en la lista entonces tiene en cuenta **su primera aparición.**

# 3. Listas

- **Búsqueda**
  - ***member***
  - **Ejemplos**

```
(define lista (list 'a 'b 'c 'b))
lista  ➔ (a b c b)

(member 'b lista)
       ➔ (b c b)

(member 'z lista)
       ➔ #f

(member 'a '((a) b c))
       ➔ #f
```

## 3. Listas

- **Búsqueda**
  - ***member***
  - Uso de ***member*** para definir el predicado *pertenece?*

```
(define (pertenece? x lista)
  (list? (member x lista))
  )

(define lista (list 'a 'b 'c 'b))
 lista  ➔ (a b c b)

(pertenece? 'b lista)  ➔ #t
(pertenece? 'z lista) ➔ #f
```

# 3. Listas

- **Búsqueda**
  - ***memq***, ***memv***
    - Son equivalentes a ***member*** pero usan los predicados ***eq?*** y ***eqv?***, respectivamente

# 3. Listas

- **Búsqueda**
  - ***memf***

    *(**memf** predicado lista)*

    - Es similar a ***member*** pero usa un *predicado* específico que se pasa como parámetro.
    - Busca el **primer elemento** de la lista que hace verdadero el *predicado*
      - ✓ Si existe, devuelve **el resto** de la lista a partir del objeto buscado.
      - ✓ En caso contrario, devuelve *#f*

# 3. Listas

- **Búsqueda**
  - ***memf***
  - **Ejemplos**

*(**memf** **even?** '(1 2 3 4 5 6))*
➔ *(2 3 4 5 6)*

*(**memf** (**lambda** (arg) (>= arg 5))*
*'(0 1 2 3 4 5 7 8 9 10))*

➔ *(5 6 7 8 9 10 11)*

# 3. Listas

- **Búsqueda**
  - *findf*

    (*findf* predicado lista)

    - Es similar a ***memf***
    - Busca el **primer elemento** de la lista que hace verdadero el *predicado*
      - ✓ Si existe, devuelve **elemento**
      - ✓ En caso contrario, devuelve #f

## 3. Listas

- **Búsqueda**
  - *findf*
  - **Ejemplos**

```
(findf  even?  '(1 2 3 4 5 6))
➔ 2

(findf (lambda (arg)  (>= arg 5))
         '(0 1 2 3 4 5 7 8 9 10 11))

➔ 5
```

# 3. Listas

- **Búsqueda**
  - ***assoc***

    *(**assoc** objeto lista-con-sublistas)*

    - Devuelve **la primera sublista** de la lista con sublistas cuyo campo ***car*** es el *objeto* buscado.
    - En caso contario, devuelve *#f*

# 3. Listas

- **Búsqueda**
  - ***assoc***
  - **Ejemplos**

*(**define** lista '((a 1) (b 2) (a 3)))*
*lista*
➔ *((a 1) (b 2) (a 3))*

*(**assoc** 'a lista)*
➔ *(a 1)*

*(**assoc** ‘z lista)*
➔ *#f*

# 3. Listas

- **Búsqueda**
  - ***assoc***
  - **Ejemplos**

```
(define persona
        '(
          (nombre "Juan")
          (apellidos "Campos" "Lara")
          (edad 12)
          (telefonos 957555555 655005500)
        )
)

(assoc 'apellidos persona)
➔ (apellidos "Campos" "Lara")
```

# 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

# 3. Listas

- **Conversión entre vector y lista**
  - *vector->list*
  - *list->vector*

# 3. Listas

- **Conversión entre vector y lista**
  - ***vector->list***

    *(**vector->list** v)*

    - ***v***: vector
    - Devuelve una lista con los elementos del vector
  - **Ejemplos**

    *(**vector->list** #(1 2 3))* ➔ *(1 2 3)*

    *(**vector->list** #(#(1 2) #(3 4)))* ➔ *(#(1 2) #(3 4)))*

## 3. Listas

- **Conversión entre vector y lista**
  - ***list->vector***

    *(**list->vector** l)*

    - *l*: lista
    - Devuelve un vector con los elementos de la lista
  - **Ejemplos**

    *(**list->vector** '(1 2 3))* ➔ *#(1 2 3)*

    *(**list->vector** '((1 2) (3 4)))* ➔ *#((1 2) (3 4)))*

## 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

# 3. Listas

- **Filtrar**
  - *filter*
  - *remove, remv, remq, remove*, remv*, remq**

## 3. Listas

- **Filtrar**
  - ***filter***

    *(**filter** predicado lista)*

    - Devuelve una *nueva lista* compuesta por los elementos de la lista que hacen verdadero el *predicado*.

  - **Ejemplo**

    *(**define** lista (**list** 1 2 3 4 5 6))*

    *(**filter** even? lista)*

    ➔ *(2 4 6)*

## 3. Listas

- **Filtrar**
  - ***remove, remv, remq, remove*, remv*, remq****

## 3. Listas

- **Filtrar**
  - ***remove***

    (***remove*** *valor lista [predicado]*)

    - Devuelve una nueva lista en la que se ha omitido el **primer elemento** que es igual al *valor* indicado usando el *predicado*.
    - El predicado es **opcional** y debe aceptar dos argumentos
    - El predicado por defecto es "="
  - **Ejemplos**

    (***remove*** 2 '(1 2 3 2 1)) → (1 3 2 1)

    (***remove*** 2. *lista* =) → (1 3 2 1)

    (***remove*** 2. *lista* **equal?**) → (1 2 3 2 1)

# 3. Listas

- **Filtrar**
  - ***remv***
    - Equivalente a

      *(**remove** valor lista **eqv?**)*
  - ***remq***
    - Equivalente a

      *(**remove** valor lista **eq?**)*

# 3. Listas

- **Filtrar**
  - ***remove****

    (***remove**** *lista-de-valores lista [predicado]*)

    - Elimina de *lista* **todos** los elementos que aparecen en la lista-de-valores
    - El predicado es **opcional** y debe aceptar dos argumentos
    - El predicado por defecto es "="

  - **Ejemplos**

    (***remove****  **(list** *1 3)* **(list** *1 2 3 2 1))* ➔ *(2 2)*

## 3. Listas

- **Filtrar**
  - ***remv****
    - Equivalente a

      *(**remove*** valor lista **eqv?**)*

  - ***remq****
    - Equivalente a

      *(**remove*** valor lista **eq?**)*

# 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

# 3. Listas

- **Ordenar**
  - ***sort***

    *(**sort** lista predicado-relacional)*

    - Ordena los elementos de la lista
  - **Ejemplos**

    *(**sort** '(1 3 4 2) <)*

    ➔ *(1 2 3 4)*

    *(**sort** '("perro" "gato" "oso" "liebre") string>?)*

    ➔ *("perro" "oso" "liebre" "gato")*

## 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

# 3. Listas

- **Procesar**
  - *map, andmap, ormap*
  - *apply*
  - *for-each*
  - *foldl, foldr*

## 3. Listas

- **Procesar**
  - ***map, andmap, ormap***

## 3. Listas

- **Procesar**
  - ***map***

    *(**map** procedimiento $lista_1$ [$lista_2$ … $lista_n$])*

    - Aplica el procedimiento a los elementos de la listas
    - El procedimiento debe aceptar el mismo **número de argumentos** que el **número de listas** indicadas
    - Todas las listas deben tener el **mismo número de elementos**
    - Se devuelve una lista conteniendo los **resultados generados** por el procedimiento

# 3. Listas

- Procesar
  - *map*
  - Ejemplos

*(**map** equal? '(1 2 3) '(3 2 1))*

➔ *(#f #t #f)*

*(**map** * '(1 2 3) '(10 20 30))*

➔ *(10 40 90)*

*(**map** max '(7 3 8) '(8 7 0) '(6 3 9))*

➔ *(8 7 9)*

## 3. Listas

- **Procesar**
  - ***map***
  - **Ejemplos**

*(**map cadr** '((a b) (d e) (g h)))*

*➔ (b e h)*

*(**map** (**lambda** (n) (**expt** n n))*

*'(1 2 3 4 5))*

*➔ (1 4 27 256 3125 )*

# 3. Listas

- **Procesar**
  - ***andmap***

    *(**andmap** procedimiento* $lista_1$ *[*$lista_2$ *...* $lista_n$*])*

    - Es similar a ***map***, pero
      - ✓ devuelve *#f* en cuanto el procedimiento obtiene dicho resultado
      - ✓ en caso contario, devuelve el resultado de la **última aplicación** del procedimiento

## 3. Listas

- Procesar
  - *andmap*
  - Ejemplos

*(andmap even? '(1 2 3))* ➔ *#f*

*(andmap < '(1 2 3) '(10 20 30))* ➔ *#t*

*(andmap < '(1 2 3) '(4 0 6) '(7 8 9))* ➔ *#f*

## 3. Listas

- **Procesar**
  - ***ormap***

    (***ormap*** *procedimiento* $lista_1$ $[lista_2 \ldots lista_n]$)

    - Es similar a ***map***, pero
      - ✓ devuelve *#f* si el procedimiento obtiene dicho resultado en todas sus aplicaciones
      - ✓ en caso contario, devuelve el **primer resultado** de una aplicación del procedimiento que no sea *#f*

## 3. Listas

- Procesar
  - *ordmap*
  - Ejemplos

*(ormap even? '(1 2 3)) → #t*

*(ormap = '(1 2 3) '(10 20 30)) → #f*

*(ormap + '(1 2 3) '(4 0 6) '(7 8 9)) → 12*

## 3. Listas

- **Procesar**
  - ***apply***

  *(**apply** procedimiento [valor$_1$ valor$_2$ ... valor$_n$] lista)*

    - Ejecuta el procedimiento usando como argumentos los elementos de la lista creada con

      *(list* valor$_1$ valor$_2$ ... valor$_n$ lista)*

## 3. Listas

- Procesar
  - *apply*
  - Ejemplos

*(apply + '(1 2 3 4 5))*

➔ *15*

*(apply + 1 2 3 '(4 5))*

➔ *15*

*(apply max 1 2 '(3 4 5 5 4 3 2 1))*

➔ *5*

# 3. Listas

- **Procesar**
  - *apply*
  - **Ejemplos**

Sin paréntesis: función con argumentos opcionales

```
(define (componer f g)
  (lambda args
    (f (apply g args))
  )
)
((componer sqrt *) 12 75) → 30
```

# 3. Listas

- **Procesar**
  - ***for-each***

    *(**for-each** procedimiento lista$_1$ [lista$_2$ ... lista$_n$])*

    - Es similar a ***map***, pero no devuelve ningún resultado
    - Se utiliza si se quieren obtener efectos colaterales:
      - ✓ escribir
      - ✓ modificar
      - ✓ etc.

## 3. Listas

- **Procesar**
  - *for-each*
  - **Ejemplos**

```
(define (ver-paridad dato)
    (display "El número ")   (display dato) (display " es ")
    (if (even? dato)    (display "par")
                        (display "impar")
       )
   (newline)
  )
 (for-each   ver-paridad   '(1 2 3)) →
```

*El número 1 es impar*

*El número 2 es par*

*El número 3 es impar*

# 3. Listas

- **Procesar**
  - ***foldl***

    *(**foldl** procedimiento inicial $lista_1$ [$lista_2$ ... $lista_n$])*

    - Como ***map***, ***foldl*** aplica el procedimiento a los elementos de uno o más listas
    - Recorre las listas de izquierda (*left*) a derecha (de ahí la "l" final de ***foldl***)

  - **Observación**
    - ***map*** devuelve el resultado en una lista
    - ***foldl*** combina los valores devueltos de forma arbitraria determinada por el procedimiento

## 3. Listas

- **Procesar**
  - *foldl*
  - **Ejemplos**

```
(foldl + 0 '(1 2 3)) → 6
(foldl cons '() '(1 2 3 4)) → (4 3 2 1)

(foldl (lambda (a b result)
          (* result (- a b)))
       1
       '(1 2 3)
       '(4 5 6))
→ -27
```

# 3. Listas

- **Procesar**
  - ***foldr***

    *(**foldr** procedimiento inicial* $lista_1$ *[*$lista_2$ *…* $lista_n$*])*

    - Como ***foldl***, pero los elementos de las listas son procesados de **derecha a izquierda** (de ahí la "r" final de ***foldr***)

  - **Ejemplos**

    *(**foldr** + 0 '(1 2 3))* ➔

    *(**foldr** **cons** '() '(1 2 3 4))* ➔ *(1 2 3 4)*

# 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

## 3. Listas

- **Procedimientos de racket/list**
  - *empty, empty?, cons?*
  - *first, rest, second, third, …, tenth*
  - *last, last-pair*
  - *make-list*
  - *take, takef, take-right, takef-right*
  - *drop, dropf, drop-right, dropf-right*
  - *split-at, splitf-at, split-at-right, splitf-at-right*
  - *add-between*
  - …

http://docs.racket-lang.org/reference/pairs.html

# 3. Listas

- **Procedimientos de racket/list**
  - *append*, append-map*
  - *flatten*
  - *remove-duplicates*
  - *filter-map, filter-not*
  - *count*
  - *partition*
  - *shuffle*
  - *argmin, argmax*
  - *range*

http://docs.racket-lang.org/reference/pairs.html

# 3. Listas

- Definición
- Creación
- Longitud
- Predicados
- Igualdad
- Acceso
- Modificación
- Búsqueda
- Conversión entre vector y lista
- Filtrar
- Ordenar
- Procesar
- Procedimientos de racket/list
- Ejemplos de uso de listas

## 3. Listas

- **Ejemplos de uso de listas**
  - Pertenece
  - Elementos
  - Contar elementos
  - Invertir

;; comprueba si x pertenece a una lista sin sublistas

```
(define (pertenece? x lista)
  (cond
    ;; se comprueba si es una lista
    ((not (list? lista))          #f)
    ;; se comprueba si es la lista vacía
    ((null? lista)                #f)
    ;; se comprueba si es igual al primer elemento
    ((equal? x (car lista))    #t)
    ;; se busca en el resto de la lista
    (else                         (pertenece? x (cdr lista)))
    )
  )
```

```scheme
;; comprueba si x pertenece a una lista con sublistas
(define (pertenece-completo? x lista)
  (cond
    ;; se comprueba si es una lista
    ((not (list? lista))    #f)
    ;; se comprueba si es la lista vacía
    ((null? lista)          #f)
    ;; se comprueba si el primer elemento es una sublista
    ((list? (car lista))
             (cond     ;; se comprueba si está en la sublista
                  ((pertenece-completo? x (car lista)) #t)
                  ;; se busca en el resto de la lista
                  (else    (pertenece-completo? x (cdr lista)))
             )
     )
    ;; se comprueba si es igual al primer elemento
     ((equal? x (car lista)) #t)
    ;; se busca en el resto de la lista
    (else                 (pertenece-completo? x (cdr lista)))
    )
  )
```

*(pertenece? 'a '(b a c d e)) ➔ #t*

*(pertenece? 'a '((b a) (c d e))) ➔ #f*

*(pertenece-completo? 'a '(b a c d e)) ➔ #t*

*(pertenece-completo? 'a '((b a) (c d e))) ➔ #t*

```
(define (elementos lista)
 (cond
   ;; se comprueba si es una lista
   ((not (list? lista))    ())
   ;; se comprueba si es la lista vacía
   ((null? lista)          ())
   ;; se comprueba si el primer elemento es una sublista
   ((list? (car lista))
   ;; se concatenan las listas generadas por la primera sublista
   ;;  y el resto de la lista
            (append  (elementos (car lista))
                     (elementos (cdr lista))
                  )
    )
   ;; se inserta el primer elemento en la lista generada por el resto de la lista
   (else          (cons  (car lista)
                         (elementos (cdr lista))
                 )
   )
  )
)
```

```
(define (contar-elementos lista)
  (length (elementos lista))
  )

(elementos '(a ((b c) (d e)))) → (a b c d e)

(contar-elementos '(a ((b c) (d e)))) → 5

(length '(a ((b c) (d e)))) → 2
```

```scheme
(define (invertir lista)
  (cond
    ;; se comprueba si es una lista
    ((not (list? lista))    ())
    ;; se comprueba si es la lista vacía
    ((null? lista)          ())
    ;; se comprueba si el primer elemento es una sublista
    ((list? (car lista))
     ;; se concatenan las listas generadas al invertir el resto y la primera sublista
                (append   (invertir (cdr lista))
                          (list (invertir (car lista)))
                )
     )
    ;; el primer elemento se inserta al final de la lista generada
    ;; al invertir el resto de la lista
    (else      (append   (invertir (cdr lista))
                         (list (car lista))
               )
    )
  )
)
```

*(**define** lista1 '( (a b) (c d)))*

*(**define** lista2 '(a (b (c d)) ( e f)))*

*(invertir lista1) ➔ ((d c) (b a))*

*(invertir lista2) ➔ ((f e) ((d c) b) a)*

# Índice

1. Vectores
2. Pares
3. Listas
4. Funciones con parámetros obligatorios u opcionales

## 4. Funciones con parámetros obligatorios u opcionales

- Función con un número **fijo** de parámetros **obligatorios**
- Función con un número **indeterminado** de parámetros **opcionales**
- Función con un número **fijo** de parámetros **obligatorios** y un número **indeterminado** de parámetros opcionales

## 4. Funciones con parámetros obligatorios u opcionales

- Función con un número **fijo** de parámetros **obligatorios**
- Función con un número **indeterminado** de parámetros **opcionales**
- Función con un número **fijo** de parámetros **obligatorios** y un número **indeterminado** de parámetros opcionales

# 4. Funciones con parámetros obligatorios u opcionales

- Función con un número **fijo** de parámetros **obligatorios**

```
(define (función x1 … xn)
    <Cuerpo de la función>
  )
```

Forma equivalente

```
(define función
  (lambda (x1 … xn )
              <Cuerpo de la función>
    )
  )
```

## 4. Funciones con parámetros obligatorios u opcionales

- Función con un número **fijo** de parámetros **obligatorios**
  - **Ejemplo:** 4 parámetros obligatorios

```
(define (suma a b c d)
        (+ a b c d)
)
```

  - *Equivalencia*

```
(define suma
        (lambda (a b c d)
                (+ a b c d)
        )
)
```

  - *Llamada a la función "suma"*

```
(suma 1 2 3 4) → 10
```

## 4. Funciones con parámetros obligatorios u opcionales

- Función con un número **fijo** de parámetros **obligatorios**
  - **Ejemplo:** 1 parámetro obligatorio que debe ser una lista

```
(define (suma-lista lista)
  (cond
    ((null? lista) 0)
    (else (+ (car lista)
             (suma-lista (cdr lista))
          )
       )
   )
)
(suma-lista '(1 2 3 4))  → 10
```

## 4. Funciones con parámetros obligatorios u opcionales

- Función con un número **fijo** de parámetros **obligatorios**
- Función con un número **indeterminado** de parámetros **opcionales**
- Función con un número **fijo** de parámetros **obligatorios** y un número **indeterminado** de parámetros opcionales

## 4. Funciones con parámetros obligatorios u opcionales

- Función con un número **indeterminado** de parámetros **opcionales**

```
(define función
  (lambda lista
        <Cuerpo de la función>
   )
  )
```

- **Observaciones**
  - El parámetro ***lista*** recogerá todos los parámetros reales
  - *lista* no está delimitada por paréntesis

## 4. Funciones con parámetros obligatorios u opcionales

- Función con un número **indeterminado** de parámetros **opcionales**
  - **Ejemplo**

```
(suma-lista '(1 2 3 4))

(define suma-opcionales
      (lambda lista
                  (suma-lista lista)
      )
  )

(suma-opcionales 1 2 3 4) → 10

(suma-opcionales 1 2 3 4 5 6 7) → 28
```

## 4. Funciones con parámetros obligatorios u opcionales

- Función con un número **indeterminado** de parámetros **opcionales**
  - **Ejemplo**

```
(define (componer f g)
    (lambda args
            (f   (apply g args)   )
    )
  )

((componer sqrt *) 4 9 16 25) → 120
```

## 4. Funciones con parámetros obligatorios u opcionales

- Función con un número **fijo** de parámetros **obligatorios**
- Función con un número **indeterminado** de parámetros **opcionales**
- Función con un número **fijo** de parámetros **obligatorios** y un número **indeterminado** de parámetros opcionales

# 4. Funciones con parámetros obligatorios u opcionales

- Función con un número **fijo** de parámetros **obligatorios** y un número **indeterminado** de parámetros opcionales

```
(define función
        (lambda (x1 … xn  .  lista)
                <Cuerpo de la función>
        )
)
```

- **Observaciones**
  - Los parámetros obligatorios se almacenan en parámetros formales $x_1 \ldots x_n$
  - Los parámetros opcionales se almacenan en la ***lista***
  - El **punto** . situado entre $x_n$ y ***lista*** es **imprescindible** y ha de estar **separado** de los argumentos

## 4. Funciones con parámetros obligatorios u opcionales

- Función con un número **fijo** de parámetros **obligatorios** y un número **indeterminado** de parámetros opcionales
  - **Ejemplo**

```
(define suma-mixta
  (lambda (a b c . lista)
    (+ a b c (suma-lista lista))
  )
)
(suma-mixta 1 2 3) → 6
(suma-mixta 1 2 3 4) → 10
(suma-mixta 1 2 3 4 5 6 7) → 28
```

UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA

DEPARTAMENTO DE
INFORMÁTICA Y ANÁLISIS NUMÉRICO

# PROGRAMACIÓN DECLARATIVA

INGENIERÍA INFORMÁTICA

CUARTO CURSO

PRIMER CUATRIMESTRE

## Tema 5.- Tipos de datos compuestos