



## Tipos y definición...

Common Lisp soporta arreglos multi-dimensionales como la mayoría de los lenguajes de programación...

```
>> (make-array '(5) )
#(NIL NIL NIL NIL NIL)
>> (make-array '(3 4))
#2A((NIL NIL NIL NIL)(NIL NIL NIL NIL)(NIL NIL NIL NIL))
>> (make-array '(2 2 2))
#3A(( (NIL NIL) (NIL NIL) ) ((NIL NIL) (NIL NIL)) )
```

Dr. Salvador Godoy Calderón

## Opciones...

Se puede especificar un valor inicial para todas las celdas, o bien para cada una:

```
>> (make-array 5 :initial-element "hola")
#("HOLA" "HOLA" "HOLA" "HOLA" "HOLA")
```

```
>> (make-array '(2 4)
               :initial-contents '((0 1 2 3) (3 2 1 0)))
#2A((0 1 2 3) (3 2 1 0))
```

Dr. Salvador Godoy Calderón

**Acceso...**

Para tener acceso a los elementos de un arreglo se usan las funciones **aref** y **setf...**

**aref** (array reference) es un “posicionador” que señala alguna posición (un índice) dentro de un arreglo y, por lo tanto, sirve para leer el contenido de dicha posición...

**setf** es una función general de escritura en el lenguaje que “escribe” algún valor en alguna localidad de memoria bien especificada...

Dr. Salvador Godoy Calderón

**Acceso...**

```
>> (defvar *arreglo* (make-array '(3 4)) )
*ARREGLO*

>> (aref *arreglo* 2 2)
NIL

>> (setf (aref *arreglo* 2 2) "Manzana")
"MANZANA"

>> *arreglo*
#2A((NIL NIL NIL NIL) (NIL NIL NIL NIL) (NIL NIL "Manzana" NIL))

>> (aref *arreglo* 2 2)
"Manzana"
```

Dr. Salvador Godoy Calderón

*Sin restricción de tipos...*

```
>> (defvar *A* (make-array 10)) )
*A*
>> (setf (aref *A* 3) "Hola")
"HOLA"

>> (setf (aref *A* 5) 34/69)
"HOLA"

>> (setf (aref *A* 7) '(a b c))
"HOLA"

>> (setf (aref *A* 9) 142.698)
"HOLA"

>> *A*
#(0 0 0 "HOLA" 0 34/69 0 (A B C) 0 142.698)
```

Dr. Salvador Godoy Calderón

*Arreglos dinámicos...*

A un arreglo que se le puede modificar su tamaño (dimensiones) se le llama arreglo dinámico...

```
>> (setq A (make-array '(3 2) :adjustable T
                         :initial-contents '((A B) (C D) (E F))) )
#2A( (A B) (C D) (E F))
>> (adjust-array A '(2 4))
#2A( (A B NIL NIL)(C D NIL NIL))
```

Si el nuevo tamaño es mayor se agregan elementos, si el nuevo tamaño es menor se eliminan elementos ...

Dr. Salvador Godoy Calderón

*Otras opciones...*

El número de dimensiones  
del arreglo...

```
>>(array-rank A)
```

**2**

```
>>(array-dimensions A)  
(2 4)
```

La capacidad en cada  
dimensión de un arreglo...

El número máximo de  
elementos en el arreglo...

```
>>(array-total-size A)
```

**8**

Dr. Salvador Godoy Calderón

## 2 **Registros**



## Registros...

Al igual que la mayoría de los lenguajes de programación *Common Lisp* permite la creación de estructuras de registro:

```
( defstruct <etiqueta> <campo1> <campo2> ...)
```

Opcionalmente se puede colocar un valor inicial por omisión para cada campo.

```
( defstruct Persona nombre paterno (materno nil) )
```

*Dr. Salvador Godoy Calderón*

## Lo importante...

La cualidad única de *Common Lisp* es que, al definir una estructura de registro, automáticamente define también funciones para su manejo:

```
make-<etiqueta>
<etiqueta>-p
<etiqueta>-<campo#1>
<etiqueta>-<campo#2>
...
<etiqueta>-<campo#n>
```

constructor de registros

predicado identificador del tipo

funciones de acceso a cada campo...

*Dr. Salvador Godoy Calderón*

*Ejemplo...*

```
>>( defstruct Persona nombre paterno (materno nil) )  
PERSONA
```

```
>>( setq Yo ( make-Persona :nombre 'Salvador  
:paterno 'Godoy  
:materno 'Calderón ) )  
#S (PERSONA :NOMBRE SALVADOR :PATERNO GODOY :MATERNO CALDERÓN)
```

Dr. Salvador Godoy Calderón

*Uso de las funciones...*

```
>>(Persona-p Yo)
```

```
T
```

```
>> (Persona-paterno Yo)  
GODOY  
>> (Persona-materno Yo)  
CALDERÓN  
>> (Persona-nombre Yo)  
SALVADOR
```

Dr. Salvador Godoy Calderón

## *Jerarquías...*

Es posible definir estructuras jerarquizadas (por inclusión) para darle forma a algún dominio de conocimiento:

```
>> (defstruct Persona nombre paterno (materno nil))
PERSONA
```

```
>> (defstruct (Astronauta (:include Persona))
              tamaño_casco
              (bebida_favorita 'Tang))
ASTRONAUTA
```

**Astronauta** incluye los mismos campos que **Persona**, pero además, las mismas funciones definidas para **Persona**...

Dr. Salvador Godoy Calderón

## *Entidades...*

```
>> (setq X1 (make-Astronauta :nombre 'Yuri
                               :paterno 'Gagarin
                               :tamaño_casco 17.5
                               :bebida_favorita 'Vodka))
#S(ASTRONAUTA :NOMBRE YURI :PATERNO GAGARIN :MATERNO NIL ...)
```

```
>> (setq X2 (make-Astronauta :nombre 'Neil
                               :paterno 'Alden
                               :materno 'Armstrong
                               :tamaño_casco 16))
#S(ASTRONAUTA :NOMBRE NEIL :PATERNO ALDEN ...)
```

Dr. Salvador Godoy Calderón

*Funciones...*

```
>> (Astronauta-p X1)
T
>> (Persona-p X2 )
T
>> (Astronauta-bebida_favorita X2 )
TANG
>> (Astronauta-bebida_favorita X1 )
VODKA
>> (Astronauta-materno X1 )
NIL
>> (Persona-materno X2 )
ARMSTORNG
```

Dr. Salvador Godoy Calderón



### 3 Listas de asociación

## ¿Qué son?...

Una lista de asociación (*alist*) es una lista que contiene parejas llave–valor. Cada una de esas parejas es una asociación

```
( (uno . 1) (dos . 2) (tres . 3) (cuatro . 4) ... )
```

Se trata de un subconjunto de un producto cruz que asocia elementos de conjuntos distintos pero que se encuentran en la misma posición relativa...

*Llaves* = uno, dos, tres, cuatro

*Valores* = 1, 2, 3, 4

Dr. Salvador Godoy Calderón

## Cómo construirlas...

Las listas de asociación son fundamentalmente el resultado de un mapeo con una función de aridad 2:

```
>> (defun asocia (llave valor)
      (cons llave valor))
ASOCIA
>> (setq llaves '(a b c d))
(A B C D)
>> (setq valores '(w x y z))
(W X Y Z)
>> (mapcar #'asocia llaves valores )
((A . W) (B . X) (C . Y) (D . Z))
```

Dr. Salvador Godoy Calderón

**Soporte...**

Existen varias funciones en *Common LISP* para administrar listas de asociación:

```
( pairlis <lista1> <lista2> <lista-asociación>)
```

```
>> (setq A '(uno dos tres cuatro))
(UNO DOS TRES CUATRO)

>> (setq B '(1 2 3 4))
(1 2 3 4)

>> (pairlis A B)
((CUATRO . 4) (TRES . 3) (DOS . 2) (UNO . 1))
```

*Dr. Salvador Godoy Calderón*

**El tercer argumento...**

Si a **pairlis** se le proporciona el tercer argumento (una lista de asociación), entonces agrega las parejas formadas a la lista indicada (de forma no-destructiva):

```
>> (setq Lista-A '((diez . 10) (once . 11)))
((DIEZ . 10) (ONCE . 11))

>> (pairlis A B Lista-A)
((CUATRO . 4) (TRES . 3) (DOS . 2) (UNO . 1) (DIEZ . 10) (ONCE . 11))

>> Lista-A
((DIEZ . 10) (ONCE . 11))
```

*Dr. Salvador Godoy Calderón*

**Agregar...**

También se puede usar la función **acons** para agregar una asociación a una lista de asociaciones (también de forma no-destructiva):

```
( acons <llave> <valor> <lista-asociación>)
```

```
>> (acons 'nueve 9 Lista-A)
((NUEVE. 9) (DIEZ. 10) (ONCE. 11))

>> Lista-A
'((DIEZ. 10) (ONCE. 11))
```

*Dr. Salvador Godoy Calderón*

**Buscar...**

Para recuperar la información contenida en una lista de asociación existen cuatro funciones :

**assoc , assoc-if , rassoc , rassoc-if**

```
>> (setq lista '((x . 100) (y . 200) (z . 50)) )
((X. 100)(Y. 200)(Z. 50))

>> (assoc 'y lista)
(Y. 200)

>> (assoc 'x lista)
(X. 100)

>> (assoc 'w lista)
NIL
```

*Dr. Salvador Godoy Calderón*

*Assoc-If...*

La función **assoc-if** opera de forma muy semejante a **assoc** pero entrega la primera asociación que cumple alguna condición expresada sobre su llave:

```
>> (setq lista2 '((1 . uno) (2 . dos) (3 . tres) (4 . cuatro)) )
((1 . UNO) (2 . DOS) (3 . TRES) (4 . CUATRO))

>> (assoc-if #'evenp lista2)
(2 . DOS)

>> (assoc-if #'(lambda (x) (> x 3.5)) lista2)
(4 . CUATRO)
```

Dr. Salvador Godoy Calderón

*Al revés...*

**rassoc** y **rassoc-if** rastrean la lista de asociaciones en el sentido opuesto de la relación, es decir, buscando la asociación a partir de algún valor:

```
>> (setq lista3 '((uno . one) (dos . two) (tres . three)) )
((UNO . ONE) (DOS . TWO) (TRES . THREE))

>> (rassoc 'two lista3)
(DOS . TWO)

>> (rassoc-if #'(lambda (x) (eql x 'two)) lista3)
(DOS . TWO)
```

Dr. Salvador Godoy Calderón

**Nota...**

Si la lista de asociaciones contiene listas normales (propias), entonces **assoc** también funciona , sin embargo, **rassoc** no:

```
>> (setq otralista '((uno 1) (dos 2) (tres 3)))  
((UNO 1) (DOS 2) (TRES 3))  
  
>> (assoc 'dos otralista)  
(DOS 2)  
  
>> (rassoc 2 otralista)  
NIL
```

Dr. Salvador Godoy Calderón

## 4 Tablas Hash



## Tablas hash...

Existen en *LISP* estructuras de datos más eficientes que otras, aunque menos flexibles:

Listas → Arreglos

Listas de asociación → Tablas hash  
(alists)

Una tabla hash es una estructura para almacenar parejas llave – valor en forma eficiente...

Su principal ventaja es que el tiempo de acceso a cada elemento es CONSTANTE...

*Dr. Salvador Godoy Calderón*

## Desventajas...

- ★ Con una tabla hash sólo se puede consultar la asociación de forma directa ...
- ★ El compilador no puede fácilmente desplegar el contenido de la tabla ...

```
>> (defparameter *Tabla* (make-hash-table))
*TABLA*

>> *Tabla*
#<HASH-TABLE :TEST EQL :COUNT 0 {1004522133}>
```

*Dr. Salvador Godoy Calderón*

*Uso...*

Para crear una tabla hash se usa **make-hash-table**...

```
>> (setq Tabla (make-hash-table))
#S(HASH-TABLE ...)
```

Para insertar y buscar un valor en la tabla se usa la función **get-hash**...

```
>> (gethash <llave> Tabla)
<valor>
T

>> (setf (gethash <llave> Tabla) <valor>)
<valor>
```

*Dr. Salvador Godoy Calderón*

*Observación...*

De manera normal, las tablas hash usan **eq** para comparar la llave de una búsqueda ...

Si usamos una lista (de asociación) como llave de una tabla hash, resulta necesario indicarle que debe usar equal para manejo de las llaves:

```
>> (setq Tabla (make-hash-table :test #'equal))
#S(HASH-TABLE ...)
```

*Dr. Salvador Godoy Calderón*

