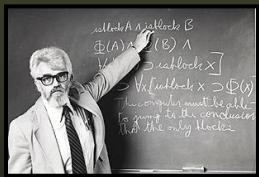




*El inicio...*

*John McCarthy* (M.I.T.) crea *LISP* en 1956, para realizar Procesamiento Simbólico.

*LIS<sup>t</sup> Processor*

Especialidad para expresar algoritmos recursivos y manejar tipos de datos dinámicos.



Dr. Salvador Godoy Calderón

*El boom de la década 1980...*

*LISP* se había convertido en el gran favorito para solución de problemas.

Gracias al presupuesto de *DARPA* se crearon y después comercializaron computadoras especiales llamadas



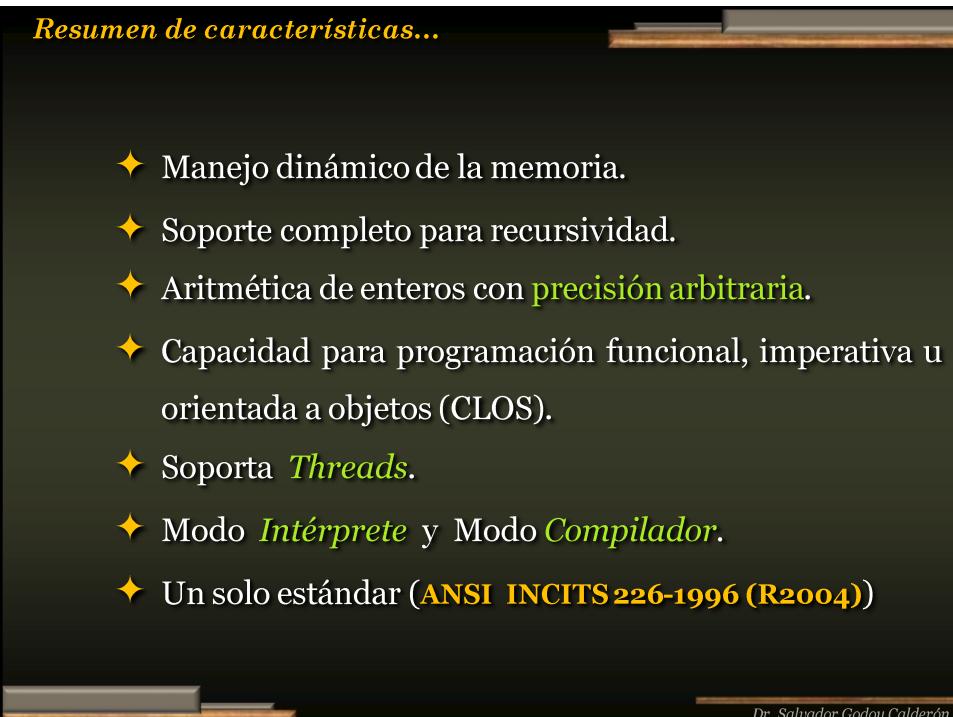
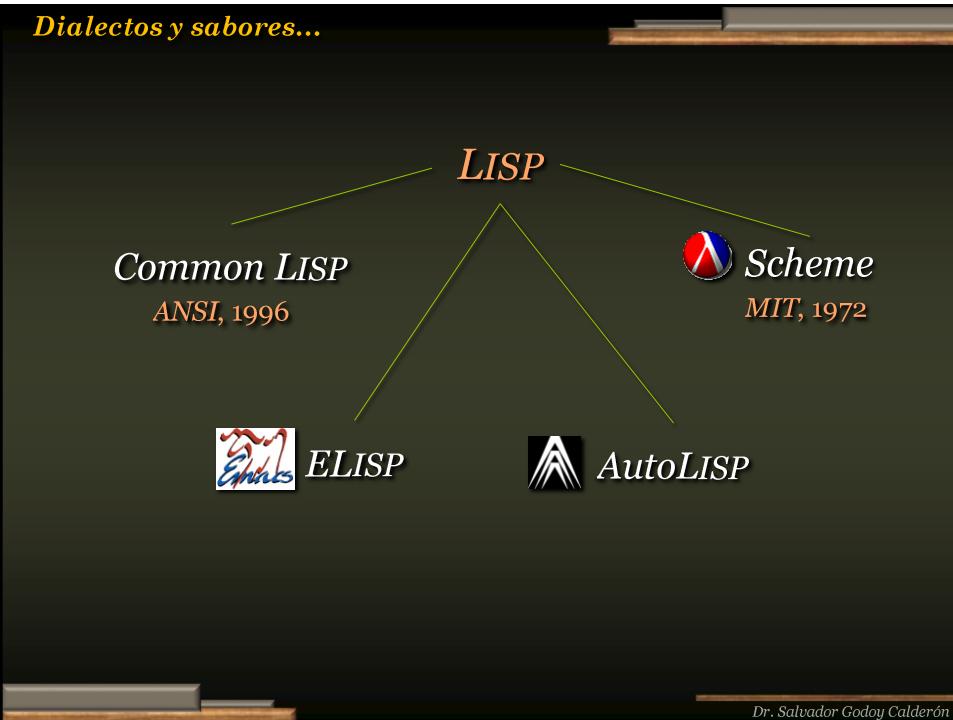
Symbolics 3640

*LISP Machines.*

*Symbolics, Texas Instruments, LMI, etc.*



Dr. Salvador Godoy Calderón





## Steel Bank Common Lisp (SBCL)

### *Lo básico...*

- ◆ Intérprete/Compilador sobre línea de comandos...
- ◆ Gratuito (licencia GNU)...
- ◆ Multiplataforma (Linux, Mac OS/X, Windows, etc)...

Para programar pueden usar el editor de su preferencia, pero se recomienda usar EMACS...

<http://functionalrants.wordpress.com/2008/09/06/how-to-set-up-emacs-slime-sbcl-under-gnulinux/>

***SBCL...***<http://www.sbcl.org>*Steel Bank Common Lisp*

The most recent version of SBCL is 1.1.10, released July 28, 2013. The release notes are available on the [news](#) page.

**Source:** [sbcl-1.1.10-source.tar.bz2](#)

The development version is available from git:

```
git clone git://git.code.sf.net/p/sbcl/sbcl
```

**Binaries:**

The table below links to the latest binaries for SBCL on each platform, where are available.

After downloading SBCL, refer to the [getting started](#) page for instructions on how to install the release.

If a binary of this version of SBCL is not available for your platform, or if you'd like to customize the binary, download the [sources](#) and follow the directions for [compiling](#).

	X86	AMD64	PPC	SPARC	Alpha	MIPSbe	MIPSle
<b>Linux</b>	1.0.58	1.1.10	1.0.98	1.0.98	1.0.98	1.0.95	1.0.79
<b>Darwin (Mac OS X)</b>	1.1.6	1.1.8	1.0.47				
<b>Solaris</b>	1.1.0	1.0.05		1.0.23			
<b>FreeBSD</b>	1.0.23	1.0.22					
<b>NetBSD</b>	1.0.22	1.1.0	1.0.05				

In addition to the original SBCL, a Windows fork exists that improves support for the Windows platform, especially in the area of threads, I/O, and x86\_64 support. Though it has not yet been incorporated into mainline, Windows users may want to consider using it in the meanwhile.

**Key**

- Available and supported
- Port in progress
- Not available (porters welcome!)
- No such system

Dr. Salvador Godoy Calderón

***Muy recomendable...***

Video en YouTube sobre la instalación de SBCL y otras herramientas útiles...

<http://youtu.be/VnWVu8VVDbI>

This is GNU Emacs, one component of the [GNU/Linux](#) operating system.

GNU Emacs 23.4.1 (x86\_64-pc-linux-gnu, GTK+ Version 2.24.18)  
Copyright (C) 2013 Free Software Foundation, Inc.

**Authors** Many people have contributed code included in GNU Emacs  
**Contributing** How to contribute improvements to Emacs

**GNU and Freedom**

**System**  
[Absence of Warranty](#)  
[Copying Conditions](#)  
[Getting New Versions](#)  
[Ordering Manuals](#)

**Why we developed GNU Emacs, and the GNU operating system**

**GNU Emacs comes with ABSOLUTELY NO WARRANTY**  
Conditions for redistributing and changing Emacs  
How to obtain the latest version of Emacs  
Buying printed manuals from the FSF

**Emacs Tutorial**  
**Emacs Guided Tour**

**Learn basic Emacs keystroke commands**  
See an overview of Emacs features at [gnu.org](#)

\*About GNU Emacs\*

Polling "/tmp/slime.17466"... (Abort with 'M-x slime-abort-connection').

Dr. Salvador Godoy Calderón

## Uso práctico...

The screenshot shows an Emacs window with two buffers:

- sr-yyc.lisp:** Contains the source code for the `sr-yyc` function. The code implements a YYY algorithm for basic matrix row analysis, including detection of incompatible rows and verbose output.
- SLIME REPL:** Shows the SBCL port (41584), PID (5681), and SLIME version (2013-06-26). The prompt `CL-USER>` is visible, indicating an interactive session.

A yellow arrow points from the text "Archivo con código fuente" to the sr-yyc.lisp buffer. Another yellow arrow points from the text "Intérprete" to the SLIME REPL buffer.

Archivo con código fuente      Intérprete

Dr. Salvador Godoy Calderón

Conceptos  
básicos...



## Dos opciones...

- ◆ Átomo: • Números  
7, 3, 45.28, 79.12,  
• Símbolos  
HOLA, CASA, VALOR23, ERROR, NIL, T

- ◆ Lista: Secuencia de átomos, separados por espacio y entre paréntesis.

( ESTA ES UNA LISTA )  
 ( CASA 3 POLLO 45.2 (5 6 7) (3.1 3.2 3.3))  
 ( ROJO (VERDE AZUL) () AMARILLO )  
 ( (UNO DOS) (TRES CUATRO CINCO) )

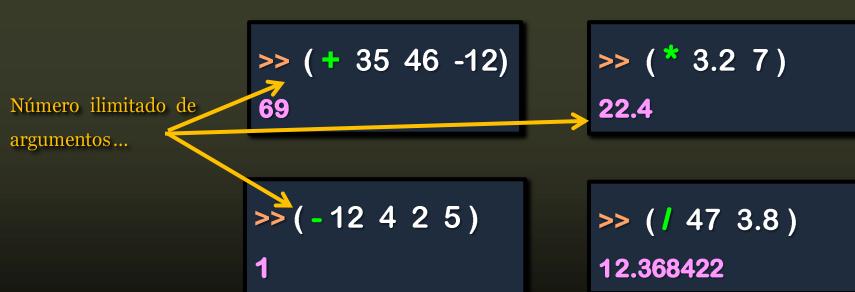
Dr. Salvador Godoy Calderón

## Lo más importante...

Todas las funciones de LISP se expresan siempre en:

*Notación Prefija (de Cambridge)*

$f(x, y, z)$ etiqueta      argumentos	$(f\ x\ y\ z)$ etiqueta      argumentos
--	--



Dr. Salvador Godoy Calderón

## Representación...

En *lisp*, la lista es la forma universal para representar, tanto datos, como instrucciones...

```
>> (+ 3 -2 5 -3.2)
2.8
```

```
>> '(+ 3 -2 5 -3.2)
(+ 3 -2 5 -3.2)
```

La diferencia es determinada por un **quote** que antecede a la expresión e indica que lo siguiente debe ser considerado como datos y por tanto, no ser evaluado...

```
>> (setq x 4.6)
4.6
>> x
4.6
>> 'x
x
```

*Dr. Salvador Godoy Calderón*

## Intérprete...

La labor del intérprete es **EVALUAR** cada expresión y entregar el **VALOR** resultante de la evaluación.

Si al intérprete le entregamos:

- ◆ Un **símbolo (variable)**: entrega el valor asociado a dicho símbolo...
- ◆ La **definición** de una función: entrega el nombre de la función definida...
- ◆ La **invocación** a una función: entrega el resultado de ejecutar dicha función...

*Dr. Salvador Godoy Calderón*

*Intérprete...*

La labor del intérprete es **EVALUAR** cada expresión y entregar el **VALOR** resultante de la evaluación.

```
>> 23.47  
23.47
```

```
>> "Hola"  
"HOLA"
```

```
>> T  
T
```

```
>> nil  
NIL
```

```
>> (length "HOLA")  
4
```

```
>> (+ 11/13 17/23)  
474/299
```

```
>> (first (second '((a b c) (1 2 3) (x y z w)) ))  
1
```

Dr. Salvador Godoy Calderón

*Precisión arbitraria, no infinita...*

```
>> (expt 2 (expt 2 10))  
1797693134862315907729305190789024733617976978942306572734  
3008115773267580550096313270847732240753602112011387987139  
3357658789768814416622492847430639474124377767893424865485  
2763022196012460941194530829520850057688381506823424628814  
7391311054082723716335051068458629823994724593847971630483  
5356329624224137216
```

```
>> (expt 2 (expt 2 (expt 2 100)))
```

Error: Attempt to create an integer which is too large to represent.  
[condition type: SIMPLE-ERROR]

Dr. Salvador Godoy Calderón

## Valor asociado...

Los átomos con valor asociado pre-definido se denominan objetos *Auto-Evaluados*.

Pero los demás átomos generan un error si se intentan evaluar:

```
>> casa
```

**<Error>**

[condition type: UNBOUND-VARIABLE]

Error: no puede entregar el  
valor asociado a ese símbolo  
porque no está previamente  
asociado a ningún valor...

```
>> X
```

**<Error>**

[condition type: UNBOUND-VARIABLE]

Dr. Salvador Godoy Calderón

## La solución...

Para no evaluar átomos sin valor asociado se usa la función QUOTE:

```
>> ( QUOTE Cañaveral )
```

**CAÑAVERAL**

```
>> ( QUOTE X )
```

**X**

**Quote** evita la evaluación de una expresión *LISP* y devuelve el nombre del mismo (lo convierte en un objeto auto-evaluado).

Con listas, **Quote** hace la diferencia entre considerarlas como código o como datos...

Dr. Salvador Godoy Calderón

*Otra forma...*

La función **Quote** también se representa mediante un Apóstrofo ' antes del argumento:

```
>> 'Sandía
SANDÍA
```

```
>> "(Uno Dos Tres)
(UNODOSTRES)
```

OJO: **Quote** sólo devuelve el **PRIMER** elemento (átomo o lista) de sus argumentos:

```
>> '(1 2) (3 4) (5 6)
(1 2)
<Error>
[condition type: TYPE-ERROR]
```

El error lo causa la segunda lista pues al intentar evaluarla no encuentra la función “3”...

Dr. Salvador Godoy Calderón

*Ejemplo...*

$$2x^2 + 7x + 5 = 0$$

$$ax^2 + bx + c = 0$$

$$x = \frac{-7 \pm \sqrt{7^2 - 4(2)(5)}}{(2)(2)}$$

```
>> (/ (+ -7.0 (sqrt (- (expt 7 2) (* 4 2 5)))) (* 2 2))
```

-1.0

```
>> (/ (- -7.0 (sqrt (- (expt 7 2) (* 4 2 5)))) (* 2 2))
```

-2.5

División...  
Substracción ...

Raíz cuadrada...

Exponente ...

Multiplicación...

Multiplicación...

Dr. Salvador Godoy Calderón

*Ejemplo...*

Recordemos que *Common Lisp* maneja números complejos:

$$5x^2 - 6x + 9 = 0$$

$$ax^2 + bx + c = 0$$

$$x = \frac{-(b) \pm \sqrt{(-b)^2 - 4(a)(c)}}{(2)(a)}$$

```
>> (/ (+ 6.0 (sqrt (- (expt 6 2) (* 4 5 9)))) (* 2 5))
#C(0.6 1.2)
>> (/ (- 6.0 (sqrt (- (expt 6 2) (* 4 5 9)))) (* 2 5))
#C(0.6 -1.2)
```

Dr. Salvador Godoy Calderón

*Representación...*

En *lisp*, la lista es la forma universal para representar, tanto datos, como instrucciones...

```
>> (+ 3 -2 5 -3.2)
2.8
```

```
>> '(+ 3 -2 5 -3.2)
(+ 3 -2 5 -3.2)
```

La diferencia es determinada por un **quote** que antecede a la expresión e indica que lo siguiente debe ser considerado como datos y por tanto, no ser evaluado...

```
>> (setq x 4.6)
4.6
>> x
4.6
>> 'x
x
```

Dr. Salvador Godoy Calderón

## Las cadenas...

Tipo especial de átomo alfanumérico, que se expresa entre comillas dobles y que también es un objeto auto-evaluado.

```
>> "Star Trek"
"StarTrek"
```

```
>> Star Wars
<Error>
[condition type: UNBOUND-VARIABLE]
```

```
>> ( length "Inteligencia Artificial" )
```

```
23
>> (string= "Inteligencia" "Sabiduría" )
NIL
```

```
>> (string= "Yo soy Investigador" "Yo soy Investigador" )
```

```
T
```

Dr. Salvador Godoy Calderón

## Caracteres ...

Las cadenas también son vistas como arreglos de caracteres ( con índice inicial = 0 )

Encontrar el carácter, dentro de una cadena, que ocupa la posición indicada ...

```
>> (char "Star Trek" 0)
```

```
#\S
```

El prefijo `\#` es la forma estándar de identificar a un carácter

```
>> (char "Star Trek" 7)
```

```
#e
```

```
>> (char "Star Trek" 4)
```

```
#\Space
```

Dr. Salvador Godoy Calderón

*Más sobre caracteres ...*

Los caracteres también son objetos auto-evaluados

```
>> #\K
```

```
#\K
```

```
>> (quote #\$)
```

```
#\$
```

```
>> '#\Q
```

```
#\Q
```

```
>> (quote #\ )
```

```
#\Space
```

```
>> '#\
```

```
#\NewLine
```

OJO: cuidado con la diferencia entre escribir y no escribir un espacio después del prefijo **#\**

*Dr. Salvador Godoy Calderón*

*Otros ejemplos...*

```
>> (char "Ahuehuéte" 6)
```

```
#\'é
```

```
>> (char= #\u (char "fUnCiOnAl" 1) )
```

```
NIL
```

Determinar si dos caracteres son iguales...

```
>> (char= (char "Dos abanicos" 2) (char "simpático" 0) #\$ )
```

```
T
```

*Dr. Salvador Godoy Calderón*



## Procesamiento básico de Listas...

### Celdas de Construcción...

En *LISP* existe una estructura de datos fundamental llamada **Celda de Construcción (*Cons Cell*)**...

Se trata de una estructura con sólo dos campos y cada uno de ellos contiene un apuntador...

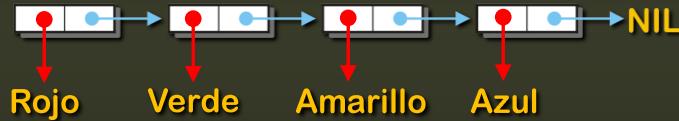


En particular, una celda puede apuntar a otra, con lo que se pueden construir cadenas ligadas de celdas

*Representación interna...*

Las listas siempre se representan internamente como listas ligadas de *Celdas de Construcción* (*Cons Cells*).

( Rojo Verde Amarillo Azul )

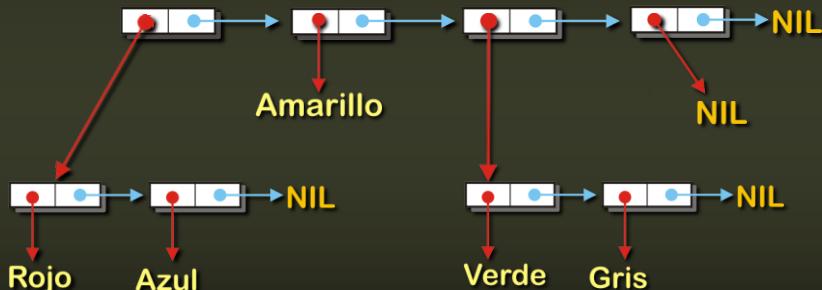


Una lista puede representar un conjunto de datos, o bien, la invocación a una función previamente definida...

Dr. Salvador Godoy Calderón

*Inclusive con sublistas...*

( ( Rojo Azul ) Amarillo ( Verde Gris ) () )



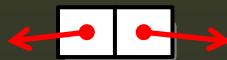
Dr. Salvador Godoy Calderón

*Creación...*

La función **CONS** crea una celda de construcción y apunta cada una de sus secciones a los argumentos...

```
>> (cons 'A '(B C) )
(A B C)
```

```
>> (cons '(1 2) '(A B C) )
((1 2) A B C)
```



```
>> (cons 2 (cons 4 (cons 6 '(8))) )
```

(2 4 6 8)

```
>> (cons 'a (cons 'b (cons 'c '() )) )
```

(A B C)

*Dr. Salvador Godoy Calderón*

*Construcción alternativa...*

Otra forma de construir listas (alternativa a **CONS**) es la función **LIST**:

```
>> (list 'A 'B '(C D) 'E )
(A B (CD) E)
```

```
>> (list nil)
(NIL)
```

```
>> (list '(nil))
((NIL))
```

A diferencia de **CONS**, la función **LIST** acepta cualquier número de argumentos y construye la lista a partir de ellos como sus elementos integrantes...

*Dr. Salvador Godoy Calderón*

***CONS vs LIST...***

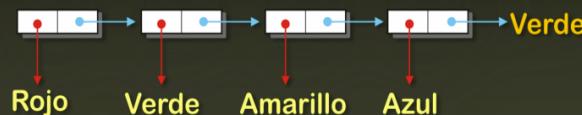
- ◆ **CONS** agrega un elemento a una lista (una celda cons)
- ◆ **LIST** construye a partir de puros elementos
- ◆ Las listas construidas por **LIST** siempre terminan en **NIL** (listas propias).
- ◆ Es posible usar **CONS** para construir una lista no-propia (no terminada en **NIL**).

**¿?      ¿?¿?      !!!**

Dr. Salvador Godoy Calderón

***Notación de componentes...***

¿Cómo se puede representar una lista con la siguiente estructura interna?



Se usa una notación alternativa a las listas llamada *Notación de componentes* o *Notación de Parejas Punteadas* (*separadas por un punto*) (*Dotted-pair*).

( Rojo Verde Amarillo Azul . Verde )

Dr. Salvador Godoy Calderón

*Otro uso de CONS...*

Si los dos parámetros de **CONS** resultan ser átomos (en lugar de un elemento y una lista), entonces construye una *pareja punteada*:

```
>> ( cons 'Buenos 'Días )
```

```
( BUENOS . DÍAS)
```

```
>> ( cons 'A (cons 'B (cons 'C 'D) ) )
```

```
( A B C . D )
```

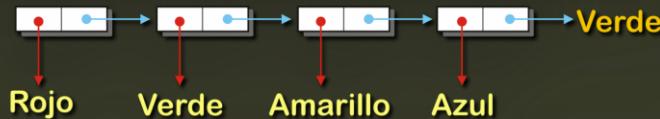
```
>> ( list (cons 'A 'B) (cons 'C 'D) )
```

```
( (A . B) (C . D))
```

Dr. Salvador Godoy Calderón

*Cuidado...*

La longitud de una lista se define como el **número** de celdas, de 1er nivel, que la componen



Por lo tanto, la longitud de:

```
( Rojo Verde Amarillo Azul . Verde )
```

es 4 y no 5 .

Dr. Salvador Godoy Calderón

*Inicio y resto...*

Las funciones FIRST y REST regresan el elemento inicial de una lista y su resto, respectivamente ...

```
>> (first '(A B C D))  
A
```

```
>> (rest '(A B C D))  
(B C D)
```

```
>> (first (cons 'a '(b c)))  
A
```

```
>> (rest (cons 'a '(b c)))  
(B C)
```

```
>> (first '())  
NIL
```

```
>> (rest '())  
NIL
```

Dr. Salvador Godoy Calderón

*Último...*

La función LAST regresa la última celda (cons) que compone una lista. La respuesta, al ser una celda cons, siempre tiene forma de lista ...

```
>> (last '(A B C D))  
(D)
```

```
>> (first (last '(a b c d)))  
D
```

```
>> (rest (last '(a b c d)))  
NIL
```

```
>> (last '())  
NIL
```

Dr. Salvador Godoy Calderón

*Curiosamente...*

Además de las funciones FIRST, REST y LAST están definidas también:

```
>> (second '(A B) C D (E) F) )  
C
```

```
>> (third '(A B C D E F))  
C
```

fourth, fifth, sixth, seventh, eighth,  
ninth, tenth

*Dr. Salvador Godoy Calderón*

*Posición de un elemento ...*

Cuando no resultan útiles las funciones FIRST, SECOND, ..., TENTH ó para realizar búsqueda aleatoria, se puede usar la función NTH :

```
>> (nth 12 '(a b c d e f g h i j k l m n o p))  
M
```

OJO: La función NTH considera los índices de posición comenzando en cero.

```
>> (third '(a b c d))  
C  
>> (nth 3 '(a b c d))  
D
```

*Dr. Salvador Godoy Calderón*

**Nota: Length y Equal...**

La función **LENGTH** de cadenas también se puede usar sobre listas

```
>> (length '( a (b c) d ) )
3
```

```
>>(length'(())
3)
```

**equal** compara listas

```
>> (equal '(a) (first '((a)) ) )
T
```

```
>>(equal '( a (b c) d ) '( a b c d ) )
NIL
```

Dr. Salvador Godoy Calderón

**\* Nota histórica ...**

Antes del estándar ANSI de *Common LISP*, las funciones **FIRST** y **REST** no existían y sus equivalentes se llamaban **CAR** y **CDR** (/cou-der/)

Esas siglas son reliquias del pasado y correspondían al nombre de secciones de instrucción en la **IBM 704** (1958)

*Contents of Address portion of Register*  
*Contents of Decrement portion of Register*

Aún hoy es fácil encontrar código escrito usando CAR y CDR, sin embargo no es lo recomendado ...

Dr. Salvador Godoy Calderón

*Antiguamente...*

En “*LISP antiguo*” las funciones **CAR** y **CDR** se podían “combinar” para examinar listas anidadas:

Las letras **A** (en **CAR**) y **D** (en **CDR**) se encadenaban en orden inverso para generar nuevas funciones...

```
>> ( CADR '( (A B) C D (E) F ) )
C
```

/ kae-der/

/ cou-dar /

```
>> ( CDAR '( (A B) C D (E) F ) )
(B)
```

*Dr. Salvador Godoy Calderón*

*Más...*

```
>> ( CAAAR '( (A B) C D (E) F ) )
```

**A**

```
>> ( CDDR '( (A B) C D (E) F ) )
(D (E) F)
```

Pero, evidentemente, la combinación dependía de la estructura de la lista de mayor nivel:

```
>> ( CDADR '( (A B) C D (E) F ) )
```

Error: Attempt to take the CDR of C which is not listp.  
[condition type: TYPE-ERROR]

*Dr. Salvador Godoy Calderón*

*Equivalencia parcial...*

<i>f</i>	pronunciación	equivalencia
CAR	/ kar /	<b>FIRST</b>
CDR	/ cou-der/	<b>REST</b>
CAAR	/ka-ar/	
CADR	/kae-der/	
CDAR	/cou-dar/	<b>SECOND</b>
DDR	/cou-dih-der/	
CAAAR	/ka-a-ar/	
CAADR	/ka-ae-der/	
CADAR	/ka-dar/	
CADDR	/ka-dih-der /	<b>THIRD</b>
CDAAR	/cou-da-ar/	
CDADR	/cou-dae-der/	
CDDAR	/cou-dih-dar/	
CDDDR	/cou-did-dih-der/	
CADDAR	/ka-dih-dih-der/	<b>FOURTH</b>
...	...	...

Modernamente se usan sólo las nuevas funciones, aunque las antiguas aún sirven...

Dr. Salvador Godoy Calderón

## Predicados de Identificación...



## *¿Qué son los predicados?*

Se trata de funciones cuyo resultado es siempre un valor de verdad (**T**, **NIL**) y, sobre los cuales, es posible siempre aplicar los conectores lógicos

```
( AND <arg1> <arg2> ... )
( OR <arg1> <arg2> ... )
( NOT <arg> )
```

- ◆ **NOT** acepta estrictamente sólo un argumento
- ◆ **AND** y **OR** aceptan cualquier número de argumentos y evalúan secuencialmente...

Dr. Salvador Godoy Calderón

## *En listas...*

El predicado **NULL** verifica si su único argumento, es o no, una lista vacía:

```
>> ( null '(1 2 3) )
NIL
```

```
>> ( null '() )
T
```

```
>> ( null NIL )
T
```

```
>> ( null () )
T
```

```
>> ( null '(NIL) )
NIL
```

Dr. Salvador Godoy Calderón

## Algunos más...

Otros predicados en *LISP* son los siguientes:

```
( numberp <arg>
  ( oddp <arg>)
  ( evenp <arg>)

  ( > <arg1> <arg2> )
  ( >= <arg1> <arg2> )
  ( < <arg1> <arg2> )
  ( <= <arg1> <arg2> )
```

Además de los ya vistos:

```
( EQUAL <arg1> <arg2> ... )
( STRING= <arg1> <arg2> ... )
( CHAR= <arg1> <arg2> ... )
```

*Dr. Salvador Godoy Calderón*

## Los predicados de identificación...

Los predicados de identificación permiten saber si un objeto es de algún tipo de datos específico...

```
>> (atom '(a b c))
NIL
>> (atom 312.26)
T
```

Predicado	Tipo de datos que identifica
<b>atom</b>	átomos
<b>numberp</b>	átomos numéricos
<b>symbolp</b>	átomos simbólicos
<b>listp</b>	listas
<b>realp</b>	números reales
<b>complexp</b>	números complejos
<b>rationalp</b>	números racionales
<b>floatp</b>	números de punto flotante
<b>integerp</b>	números enteros
<b>ratiofp</b>	fracciones
<b>characterp</b>	caracteres
<b>alpha-char-p</b>	caracteres alfabéticos
<b>alphanumericp</b>	caracteres alfanuméricos
<b>keywordp</b>	llaves
<b>stringp</b>	cadenas

*Dr. Salvador Godoy Calderón*

### *La tradición...*

Por tradición, la mayoría de los predicados de identificación terminan con el carácter ‘p’

Yo prefiero usar el carácter ‘?’

Predicado	Tipo de datos que identifica
<b>atom</b>	átomos
<b>numberp</b>	átomos numéricos
<b>symbolp</b>	átomos simbólicos
<b>listp</b>	listas
<b>realt</b>	números reales
<b>complexp</b>	números complejos
<b>rationalp</b>	números racionales
<b>floatp</b>	números de punto flotante
<b>integerp</b>	números enteros
<b>ratiofp</b>	fracciones
<b>characterp</b>	caracteres
<b>alpha-char-p</b>	caracteres alfabéticos
<b>alphanumericp</b>	caracteres alfanuméricos
<b>keyworp</b>	llaves
<b>stringp</b>	cadenas

*Dr. Salvador Godoy Calderón*

**¡ Gracias !**

*Dr. Salvador Godoy Calderón*

CIC - IPN