

INSTITUTO POLITÉCNICO NACIONAL  
Centro de Investigación en Computación



*Taller de programación  
en Common Lisp*

02

**Funciones y sus  
argumentos...**



*Dr. Salvador Godoy Calderón*



**Observaciones  
pendientes...**

## Prefijos...

En *Common LISP* se usan prefijos para identificar algunos tipos de objetos:

Prefijo	Tipo de objeto	Ejemplos
#\	Caracteres	#\A #\@ #\`
#'	Funciones	'#+ #'sqrt #'first
#C	Números complejos	#C(0 1) #C(-3/2 5/8)
#xA	Arreglos	#2A( (NIL NIL) (NIL NIL) (NIL NIL) )
#S	Registros	#S(Terna :uno NIL :dos NIL :tres NIL)
#x #o #b	hexadecimales, octales y binarios	#x12ABF #o77 #b10110

Dr. Salvador Godoy Calderón

## El caso de NIL...

```
>> (atom NIL)
T
```

El elemento **NIL**, como átomo es el valor de verdad *falso* y como lista, es la lista vacía ().

```
>> (listp NIL)
T
```

El tercer rol de **NIL** es como terminador de listas (**listas propias**), pero en una lista nunca se representa con una celda...

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

```
>> (cons NIL !)
NIL ← !
```

Dr. Salvador Godoy Calderón

## Propiedades de NIL...

- ◆ Es un átomo (la única representación del valor *falso*).
- ◆ Es una lista (vacía y con longitud = 0 ).
- ◆ Es el único objeto *LISP* que es tanto átomo como lista.
- ◆ Marca el final de una cadena de celdas de construcción (lista propia).
- ◆ **NIL**, **( )** y **'()** son notaciones equivalentes (intercambiables) para el mismo objeto.
- ◆ **FIRST**, **REST** y **LAST** de **NIL** se definen como **NIL**.

Dr. Salvador Godoy Calderón

## Predicados de igualdad...

Resulta importante observar que en *Common LISP* existe una *Familia* de predicados para comprobar la igualdad:

más específico	<b>(eq a b)</b>	Si a y b son <i>el mismo objeto (misma posición en memoria)</i>
	<b>(eql a b)</b>	Si a y b son objetos del <b>mismo tipo</b> y con el <b>mismo valor</b>
	<b>(equal a b)</b>	Si a y b son estructuralmente iguales ( <i>isomorfos</i> )
más general	<b>(equalp a b)</b>	Si a y b son iguales en algún sentido de valor asociado

Dr. Salvador Godoy Calderón

*Algunos ejemplos...*

( eq 'A 'A ) → T

( eq 3.0 3.0 ) → T

( eq 3 3.0 ) → NIL

( eql (+ 2 1) (/ 6 2)) → T

( eql 3 3.0 ) → NIL

( eql "hola" "hola" ) → NIL

( eql '(a b c) '(a b c) ) → NIL

( equal '(a b c) '(a b c) ) → T

( equal "hola" "hola" ) → T

( equal 3 3.0 ) → NIL

( equalp 3 3.0 ) → T

( equalp (+ 2 1) (/ 6.0 2)) → T

Dr. Salvador Godoy Calderón

**EQUAL vs =**

La función **EQUAL** compara objetos *LISP* y el predicado  
**=** es un caso particular de **EQUAL** que sólo admite  
valores numéricos como argumento:

&gt;&gt; (= 3/2 9/6)

T

&gt;&gt; ( equal '(a b c) '(a b c) )

T

T

&gt;&gt; (= '(a b c) '(a b c) )

&gt;&gt; ( equal 3/2 9/6 )

T

Error: '(A B C)' is not of the expected type 'number'

[condition type:type-error]

Dr. Salvador Godoy Calderón

### Asociación...

La forma más elemental de asignar valores a variables (símbolos) es mediante las funciones **SET** y **SETQ**:

(**set** '<variable> <valor>')

(**setq** [<variable> <valor>]\*')

**SET** requiere **quote** antes de la variable y sólo puede asociar una variable con un valor.

**SETQ** no requiere **quote** y puede asociar varias variables con sus respectivos valores.

Dr. Salvador Godoy Calderón

### Ejemplos...

Ambas funciones regresan el último valor asociado...

```
>> (setq Año 2014
      Mes 'Junio
      Día 'Miércoles )
MIÉRCOLES
```

```
>> (set 'x 47)
47
>> (+ x 32)
79
>> (set 'x '(a b c d))
(A B C D)
>> (second x)
B
>> (set 'x "programar")
"programar"
>> (char x 3)
#\g
```

Dr. Salvador Godoy Calderón

**NOTA...**

El intérprete de *SBCL* emite un *warning* si se asocia una variable global que no se declaró antes como tal...

```
>> (setq X -3.5)
; in: SETQ X
;   (SETQ X -3.5)
;
; caught WARNING:
; undefined variable: X
;
; compilation unit finished
; Undefined variable:
;   X
; caught 1 WARNING condition
-3.5
```

La variable se asocia a su valor de todas formas. Para evitar el mensaje declarar antes con:

**(defparameter <variable>)**

Se verá la siguiente clase...

*Dr. Salvador Godoy Calderón*

## Definición de Funciones

## Funciones de usuario...

Para la definición de nuevas funciones *LISP* usa una macro llamada **DEFUN** que tiene sólo tres argumentos:

```
(defun <etiqueta> (<arg1> <arg2>... ) <definición>)
```

```
>> (defun promedio (x y) (/ (+ x y) 2.0))  
PROMEDIO
```

Entrega como resultado la etiqueta de la función definida

*Dr. Salvador Godoy Calderón*

## Comportamiento especial...

**DEFUN** tiene el comportamiento especial de que **NO EVALÚA** sus argumentos.

Por ello, sus argumentos no necesitan ir marcados con **QUOTE** para evitar dicha evaluación:

```
>> (defun cuadrado (x) (* x x))  
CUADRADO
```

*Dr. Salvador Godoy Calderón*

*Observaciones...*

- ◆ Si la función definida no tiene argumentos, colocar los paréntesis correspondientes vacíos.
- ◆ Los símbolos que sirven de etiqueta a los argumentos **no necesitan QUOTE**, ni en la lista de argumentos, ni en el cuerpo de la definición.
- ◆ Los símbolos-etiqueta de argumentos **tampoco necesitan paréntesis**, inclusive si representan listas.

Dr. Salvador Godoy Calderón

*Definición e invocación...*

```
>> (defun saludo () "Buenos días")
SALUDO
```

```
>> (saludo)
"Buenos días"
```

Toda función regresa SIEMPRE el último valor que procesó...

```
>> (defun presenta (x y)
      (list x 'te 'presento 'a y))
PRESENTA
```

```
>> (presenta 'Jorge 'Miguel )
(JORGE TE PRESENTO A MIGUEL )
```

Dr. Salvador Godoy Calderón

### Espacios diferentes...

*LISP* mantiene espacios de memoria separados para los símbolos asociados a funciones y para los demás símbolos.

Por ello, SÍ es posible en *LISP*, que un mismo símbolo etiquete, tanto a un valor, como a una función...

```
>> (defun suma (x y) (+ (* x 2) y) )
SUMA
>> (setq suma '(a b c) )
(A B C)
>> suma
(A B C)
>> #'suma
#<Interpreted Function SUMA>
```

Dr. Salvador Godoy Calderón

### Importante...

Las funciones son el tipo más importante de objeto *LISP*...

Sólo existe un formato universal para representar funciones, tanto *de sistema* como *definidas por el usuario*; todas las funciones se comportan igual y siguen las mismas reglas.

Dr. Salvador Godoy Calderón



*¿Cuántos argumentos?...*

Al definir una función con **DEFUN** se especifica el número y etiqueta de cada argumento. Pero, si al invocar la función no se proporcionan suficientes argumentos se produce un error:

```
>> (defun calcula (x y z) (+ (* x 3) (* y 2) z))  
CALCULA  
>> (calcula 1 2 3)  
10  
>> (calcula 4 2)  
Error: CALCULA got 2 args, wanted 3 args.  
[condition-type: PROGRAM-ERROR]
```

**Ya sabemos que...**

Varias de las funciones nativas de *LISP* trabajan con un número indeterminado de argumentos, desde **cero** hasta el valor de la constante **CALL-ARGUMENTS-LIMIT**

```
>> (+ 5 4 3 2 1)
15
>> (+ 3 4)
7
>> (+ 2)
2
>> (+)
0
```

```
>> call-arguments-limit
16384
```

Dr. Salvador Godoy Calderón

**De argumentos...**

Es posible especificar un número indeterminado de argumentos para una función usando la directiva **&rest**

```
>> (defun muestra (&rest argumentos) argumentos )
```

Todos los argumentos reales durante la invocación se almacenan en una sola lista (en este caso etiquetada **argumentos**)...

Durante su ejecución, la función debe extraer los argumentos reales almacenados en la lista **argumentos** para procesarlos...

Dr. Salvador Godoy Calderón

Ejemplo...

```
>> (defun primero-y-último (&rest arg)
      (print (length arg))
      (print (first arg))
      (print (first (last arg))))
```

PRIMERO-Y-ÚLTIMO

```
>> (primero-y-último 'a 'b 'c 'd 'e )
```

5

A

E

E

??

Dr. Salvador Godoy Calderón

## Argumentosopcionales y valores por omisión

## Argumentos opcionales...

Al definir una función es posible establecer argumentosopcionales usando la directiva **&optional**

Si al invocar la función no se proporciona un argumentoopcional deberá tomar su valor por omisión...

```
>> (defun enlista (a b &optional c d) (list a b c d))
ENLISTA
```

Al invocar la función, todos los argumentos opcionales que no tengan valor se asocian automáticamente con **NIL**

*Dr. Salvador Godoy Calderón*

## Argumentos opcionales...

Los argumentos especificados antes de la directiva **&optional** constituyen el mínimo número de argumentos para esa función.

```
>> (enlista 2 4 6)
(2 4 6 NIL)

>> (enlista 'p 'q)
(P Q NIL NIL)

>> (enlista 'Uno )
Error: ENLISTA got 1 arg, wanted at least 2 args.
[condition type: PROGRAM-ERROR]

>> (enlista 'X 'Y 'Z 'W 'S )
Error: ENLISTA got 5 args, wanted at most 4 args.
[condition type: PROGRAM-ERROR]
```

*Dr. Salvador Godoy Calderón*

## Varias opciones...

Una función puede tener todos sus argumentos como opcionales:

```
>> (defun agrega (&optional (a 10) (b 20) (c 30))
      (+ a b c))
AGREGA
>> (agrega 1 2 3)
6
>> (agrega 5 4)
39
>> (agrega 2)
52
>> (agrega )
60
```

Dr. Salvador Godoy Calderón

## ¡Reglas!

- ★ Los argumentos obligatorios de una función siempre deben anteceder a los argumentos opcionales...
- ★ Al invocar a la función, los argumentos que no tengan un valor real se asociarán con el valor por omisión especificado en la función...
- ★ Si un argumento opcional no tiene valor real, ni valor por omisión especificado, entonces se asocia a **NIL**...

Dr. Salvador Godoy Calderón

### Inclusive...

La expresión para el valor por omisión de un argumento puede inclusive hacer referencia a otros argumentos...

```
>> (defun cuadrilátero (base &optional (altura base))
      ( ... ) )
```

Esta definición permite generar un cuadrilátero con sólo un parámetro (**cuadrado**) o bien con dos (**rectángulo**)...

Cualquier expresión que se pueda evaluar antes de comenzar la ejecución de la función es válida para especificar valores por omisión...

*Dr. Salvador Godoy Calderón*

### Averiguando...

Para averiguar si el valor de un argumento es real o si es el de omisión, se agrega una variable lógica a la expresión de inicialización...

```
>> (defun pareja (x &optional (y 1 valorReal))
      (cons (list x y) (list valorReal)) )
PAREJA
```

```
>> (pareja 5 6 )
(( 5 6) T)
>> (pareja 'P 'Q )
( (P Q) T)
>> (pareja '(a b c) )
( ( (A B C) 1) NIL)
```

*Dr. Salvador Godoy Calderón*



## Argumentos Etiquetados...

### **Un problema...**

Hasta el momento el problema de los argumentosopcionales es que se definen de manera *posicional*...

Imaginemos una función con cuatro argumentos, todos ellos opcionales. Los diferentes invocadores de esta función sólo proporcionarán uno de esos argumentos PERO puede ser cualquiera de ellos, no forzosamente el primero...

*¿ Cómo se soluciona este problema ?*

Es necesario identificar cada argumento de forma independiente a su posición...

### *Directiva &key...*

Con la directiva **&key** se puede hacer referencia a un argumento mediante su etiqueta...

```
>> (defun 1de4 (&key uno dos tres cuatro)
          (list uno dos tres cuatro) )
1DE4
```

```
>> (1de4 :uno 'x :dos 'a :tres 'f :cuatro 'k )
(X A F K)

>> (1de4 :cuatro 'Arroz :dos '34 )
(NIL 34 NIL ARROZ)

>> (1de4 :tres "TORNILLO" )
(NIL NIL "TORNILLO" NIL)
```

Dr. Salvador Godoy Calderón

### *cambio de etiqueta...*

También es posible asignar a los argumentos una etiqueta para ser usada durante la invocación y otra diferente para la especificación...

Todo ello combinado con valores por omisión...

```
>> (defun fruta (&key ((:pera p) 1) ((:manzana m) 0) ((:uva u)))
          (list p m u))
FRUTA
```

```
>> (fruta :pera 2 :uva 4)
(2 0 4)

>> (fruta :manzana 6 )
(1 6 NIL)
```

Dr. Salvador Godoy Calderón

*Estudiar...*

**Chapter 4:**  
Syntax and Semantics

**Chapter 5:**  
Functions

**Chapter 3:**  
Overview of LISP

3.3 - 3.10

*Dr. Salvador Godoy Calderón*

