Lisp

Comentarios:

```
; Texto con comentarios…
```

Tipos:

Átomos = Variables que contienen tan solo un dato indivisible.
Listas = Listas de datos, que pueden ser átomos y/o listas.

Funciones:

```
(defun nombre (parámetros) sentencia)

(define (nombre parametros) sentencia)

(parametros) = (var1 var2 ... varN)
```

Condicionales:

```
(cond (condición sentencia)
      …
      (T sentencia))

(if condición sentencia-true sentencia-false)
```

T = True
Nil = False
() = Lista vacía o null

Funciones con operadores:

```
(operador A B)          ; A, B y X son átomos
(operador X)            ; L1 y L2 son listas
```

+ Aritméticas:

(+ A B)	Suma
(- A B)	Resta
(* A B)	Multiplicación
(/ A B)	División
(rem A B)	Módulo de A entre B (A % B)
(add1 A)	A++
(sub1 A)	A—

+ Lógicas:

(and A B)	A intersección B
(or A B)	A conjunción B
(not A)	A negado

+ Relacionales:

(= A B)	A es igual que B
(eq A B)	A es igual que B
(equal L1 L2)	L1 es igual que L2
(> A B)	A mayor que B
(greaterp A B)	A mayor que B
(< A B)	A menor que B
(lessp A B)	A menor que B

+ Manipulación de ob_list:

(intern X)	Añade a ob_list un átomo
(remob X)	Borra un átomo de ob_list

(Nota: ob_list es una variable global de tipo lista del interprete)

+ Varias:

(gensym X)	Crea un átomo nuevo
(oddp A)	A es impar
(atom X)	X es un átomo
(numberp`X)	X es un número
(null L)	L es null (L == ())
(zerop X)	Prueba en busca de un valor de cero (X == 0)

Funciones sobre listas:

(quote X)	No evalua X
(‘X) o ‘X	Igual que (quote X)
(car L)	Da el primer elemento de L (car (1 2 3)) = 1
(cdr L)	Da una copia de L sin el primer elemento (car (1 2 3)) = (2 3)
(cons A L)	Da una copia de L con A como el primer elemento (cons 1 (2 3)) = (1 2 3)
(list X Y … Z)	Forma una nueva lista con los datos pasados (list 1 (2 3) 4) = (1 (2 3) 4)
(append L1 L2)	Devuelve una lista, resultado de fusionar L1 y L2 (append (1 2) (3 4)) = (1 2 3 4)

Las funciones car y cdr se puden combinar formando funciones más complejas como por ejemplo sería: (cadar L) = (car (cdr (car L)))
El único límite de esto suele ser de unas 7 letras, dependiendo del interprete que estemos usando.

Propiedades:

(put L Lsub1 … Lsubn valor)	Lsubn = valor
(get L Lsub1 … Lsubn)	Devuelve el valor de Lsubn
(remprop L Lsub1 … Lsubn)	Elimina el valor de Lsubn

Enunciados:

(prog (variables)
(sentencia1)
…
(sentenciaN))

(progn (sentencia1) … (sentenciaN))

(return valor)

Manejo de entrada y salida:

(print X Y … Z)	Muestra por pantalla las variables indicadas
(read)	Lee una variable del teclado
(load rutafichero)	Carga un fichero con definiciones
(open rutafichero)	Abre un fichero dando su manejador (setq infile (open ‘ruta)) Esto crea una variable manejador
(print X manejador)	Escribe un átomo en el fichero
(read manejador)	Lee un átomo del fichero
(close manejador)	Cierra el fichero.

Otras funciones:

(setq var val)	Mete val en var, y devuelve var
(length L)	Nº de elementos de L
(time N M)	N * M
(eval X)	Lo contrario que (quote X)

(mapcar ‘nombre ‘args) = ((nombre arg1) (nombre arg2) …)
(mapcar + ‘(1 2 3) ‘(4 5 6)) = (5 7 9)

(apply funcion arg) = Aplica la función a los elementos indicados
(apply + (1 2 3)) = (+ (+ 1 2) 3)

Lisp

Introduccion

```
Comentarios    -> ; texto de una sola línea
ob_list        -> lista global del interprete
Tipos de datos -> listas de atomos, y atomos
Lista vacía    -> ()

Valores condicionales:
     T   = cierto
     NIL = falso

NIL también equivale a una lista vacía.
```

Funciones

```
(defun nombre (parametros) sentencia)
(define (nombre parametros) sentencia)

(parametros) = (var var2 ... varN)
```

Sentencias de control

```
(if condicion sentenciaTrue sentenciaFalse)

(cond (condicion sentencia)
      ...
      (T sentencia))

(prog (variables)
      sentencia1
      ...
      sentenciaN)

(progn sentencia1 ... sentenciaN)

(return valor)
```

Operaciones

```
(gensym x)     -> Genera un nuevo atomo.
(intern x)     -> Añade a 'ob_list' un atomo.
(remob x)      -> Borra un atomo de 'ob_list'

(atom x)       -> ¿Es 'x' un átomo?
(numberp x)    -> ¿Es 'x' un número?
(oddp x)       -> ¿Es 'x' impar?
(null x)       -> ¿Es 'x' nulo?
```

```
(zerop x)       -> ¿Es 'x' cero?
```

Operadores aritméticos

```
(+ a b)         -> a + b
(- a b)         -> a - b
(* a b)         -> a * b
(/ a b)         -> a / b
(add1 a)        -> a + 1
(sub1 a)        -> a - 1
(rem a b)       -> resto de a / b
(mod a b)       -> a modulo b
(max x ... z)   -> valor máximo
(min x ... z)   -> valor mínimo
```

Operadores lógicos

```
(and a b)       -> a y b
(or a b)        -> a o b
(not a)         -> a negado
```

Operadores de comparación

```
(equal LA LB)   -> L1 == L2
(eq a b)        -> a == b
(greaterp a b)  -> a > b
(lessp a b)     -> a < b
(= a b)         -> a == b
(> a b)         -> a > b
(< a b)         -> a < b
(/= a b)        -> a != b
(>= a b)        -> a >= b
(<= a b)        -> a <= b
```

Operaciones sobre listas

```
(quote x)       -> Da x sin evaluar.
'x              -> Da x sin evaluar.
(eval x)        -> Evalua x.
(car L)         -> Da el primer elemento de la lista.
(cdr L)         -> Da la lista L sin el primer elemento.
(cons x L)      -> Da una lista con x como primer elemento
                   de L.
                   (cons 'a '(b c)) = (a b c)
(list x ... z) -> Da una lista formada por los parametros.
                   (list '(a b) 'c '(d e)) = ((a b) c (d e))
(append L1 L2) -> Da una lista con la fusión de L1 y L2.
                   (append '(a b) '(c d)) = (a b c d)
```

Operaciones de entrada y salida

```
(load ruta)     -> Carga un fichero con definiciones.
(print x)       -> Escribe 'x' en pantalla.
(read)          -> Lee un atomo del teclado.
(open ruta)     -> Abre un fichero dando su manejador.
                   (setq fd (open 'ruta))
(read fd)       -> Lee un atomo del fichero.
(print x fd)    -> Escribe 'x' en el fichero.
(close fd)      -> Cierra un fichero.
```

Operaciones matemáticas

```
(arctan x)      -> Arcotangente de x en radianes.
(cos x)         -> Coseno de x en radianes.
(exp x)         -> e elevado a x.
(float x)       -> Convierte x en un número real.
(ln x)          -> Logaritmo neperiano de x.
(round x)       -> Da x como un entero redondeado.
(sin x)         -> Seno de x en radianes.
(sqrt x)        -> Raíz cuadrada de x.
(trunc x)       -> Da x como un entero truncado.
```

Otras operaciones

```
(setq var val) -> var = val
(length L)     -> Número de elementos en la lista.

(mapcar funcion arg1 ... argn)
   (mapcar + '(1 2 3) '(4 5 6)) -> (5 7 9)

(apply funcion args)
   (apply + '(1 2 3)) -> (+ (+ 1 2) 3) -> 6
```