FUNCIONES DE CAMPOS MÚLTIPLES

create$	**crea un valor de campo múltiple** (create$ <expression>*) (create$ (+ 3 4) (* 2 3) (/ 8 4))
nth$	**devuelve el n-ésimo valor del campo** (nth$ <integer-expression> <multifield-expression>) CLIPS> (nth$ 3 (create$ a b c d e f g)) c
member$	**devuelve la posición del campo <expression> dentro del valor de campos múltiples** (member$ <expression> <multifield-expression>) CLIPS> (member$ blue (create$ red 3 "text" 8.7 blue)) 5 CLIPS> (member$ 4 (create$ red 3 "text" 8.7 blue)) FALSE CLIPS> (member$ (create$ b c) (create$ a b c d)) (2 3)
subsetp	**busca un subconjunto de valores dentro del valor de campos múltiples, no tiene en cuenta el orden de los valores** (subsetp <multifield-expression> <multifield-expression>) CLIPS> (subsetp (create$ hammer saw drill) (create$ hammer drill wrench pliers saw)) TRUE CLIPS> (subsetp (create$ wrench crowbar) (create$ hammer drill wrench pliers saw)) FALSE
delete$	**borra el rango especificado de valores** (delete$ <multifield-expression> <begin-integer-expression> <end-integer-expression>) CLIPS> (delete$ (create$ hammer drill saw pliers wrench) 3 4) (hammer drill wrench) CLIPS> (delete$ (create$ computer printer hard-disk) 1 1) (printer hard-disk)
delete-member$	**borra valores específicos dentro del valor multicampo (borra todos las instancias de valores repetidos)** (delete-member$ <multifield-expression> <expression>+) CLIPS> (delete-member$ (create$ a b a c) b a) (c) CLIPS> (delete-member$ (create$ a b c c b a) (create$ c b a)) (a b c)
explode$	**construye un valor de campos múltiples a partir de una cadena** (explode$ <string-expression>) CLIPS> (explode$ "hammer drill saw screw") (hammer drill saw screw) CLIPS> (explode$ "1 2 abc 3 4 \"abc\" \"def\"") (1 2 abc 3 4 "abc" "def") CLIPS> (explode$ "?x ~)") ("?x" "~" ")")
implode$	**construye una cadena de texto a partir de un valor de campos múltiples** (implode$ <multifield-expression>) CLIPS> (implode$ (create$ hammer drill screwdriver)) "hammer drill screwdriver wrench pliers saw" CLIPS> (implode$ (create$ 1 "abc" def "ghi" 2))

	"1 "abc" def "ghi" 2" CLIPS> (implode$ (create$ "abc def ghi")) ""abc def ghi""
Subseq$	**Extracción de una sub-secuencia dentro de un valor de campos múltiples** Extrae un rango especificado y devuelve un nuevo valor de multicampo con la subsecuencia (subseq$ <multifield-value> <begin-integer-expression> ; <end-integer-expression>) CLIPS> (subseq$ (create$ hammer drill wrench pliers) 3 4) (wrench pliers) CLIPS> (subseq$ (create$ 1 "abc" def "ghi" 2) 1 1) (1)
Replace$	**Reemplazar campos** (replace$ <multifield-expression> <begin-integer-expression> <end-integer-expression> <single-or-multi-field-expression>+) CLIPS> (replace$ (create$ drill wrench pliers) 3 3 machete) (drill wrench machete) CLIPS> (replace$ (create$ a b c d) 2 3 x y (create$ q r s)) (a x y q r s d)
Replace-member$	Reemplazar valores específicos contenidos en un valor multicampo (replace-member$ <multifield-expression> <substitute-expression> <search-expression>+) Example CLIPS> (replace-member$ (create$ a b a b) (create$ a b a) a b) (a b a a b a a b a a b a) CLIPS> (replace-member$ (create$ a b a b) (create$ a b a) (create$ a b)) (a b a a b a)
insert$	**Insertar campos en una determinada posición** (insert$ <multifield-expression> <integer-expression> ; debe ser igual o mayor que 1 <single-or-multi-field-expression>+) CLIPS> (insert$ (create$ a b c d) 1 x) (x a b c d) CLIPS> (insert$ (create$ a b c d) 4 y z) (a b c y z d) CLIPS> (insert$ (create$ a b c d) 5 (create$ q r)) (a b c d q r)
first$** **rest$	**Primero de los campos** CLIPS> (first$ (create$ a b c)) (a) CLIPS> (first$ (create$)) () **Resto de los campos menos el primero** CLIPS> (rest$ (create$ a b c)) (b c) CLIPS> (rest$ (create$)) ()
length$	**Determinar nº de campos** (length$ <multifield-expression>) CLIPS> (length$ (create$ a b c d e f g)) 7

FUNCIONES CON CADENAS

str-cat	**Concatenación para devolver una cadena** (str-cat <expression>*) CLIPS> (str-cat "foo" bar) "foobar"
sym-cat	**Concatenación para devolver un símbolo** (sym-cat <expression>*)
sub-string	**Extraer una subcadena** (sub-string <integer-expression> <integer-expression> <string-expression>) CLIPS> (sub-string 3 8 "abcdefghijkl") "cdefgh"
str-index	**Devuelve posición de una cadena dentro de otra** (str-index <lexeme-expression> <lexeme-expression>) CLIPS> (str-index "de f" "abcdefghi") 4 CLIPS> (str-index "qwerty" "qwertypoiuyt") 1 CLIPS> (str-index "qwerty" "poiuytqwer") FALSE
eval	**Evaluación de una función** (eval <string-or-symbol-expression>) ** uso restringido de variables locales, no funciona con constructores del tipo defrule, deffacts, etc. CLIPS> (eval "(+ 3 4)") 7 CLIPS> (eval "(create$ a b c)") (a b c)
build	**Evaluación de un constructor** (build <string-or-symbol-expression>) CLIPS> (clear) CLIPS> (build "(defrule foo (a) => (assert (b)))") TRUE CLIPS> (rules) foo For a total of 1 rule.
upcase **lowcase**	**Convertir a Mayúsculas** CLIPS> (upcase "This is a test of upcase") "THIS IS A TEST OF UPCASE" CLIPS> (upcase A_Word_Test_for_Upcase) A_WORD_TEST_FOR_UPCASE **Convertir a minúsculas** CLIPS> (lowcase "This is a test of lowcase") "this is a test of lowcase" CLIPS> (lowcase A_Word_Test_for_Lowcase) a_word_test_for_lowcase
str-compare	**Comparación de 2 cadenas** (str-compare <string-or-symbol-expression>

	<string-or-symbol-expression>) ** devuelve 0 si son iguales, <0 si el primer argumento es menor que el segundo, >0 en caso contrario CLIPS> (< (str-compare "string1" "string2") 0) TRUE ; since "1" < "2" in ASCII character set CLIPS> (str-compare "abcd" "abcd") 0
str-length	**Longitud de una cadena** (str-length <string-or-symbol-expression>) CLIPS> (str-length "abcd") 4 CLIPS> (str-length xyz) **3**
check-syntax	**Comprobación de la sintaxis para realizar una llamada a una función o a un constructor** (check-syntax <construct-or-function-string>) **devuelve: FALSE si no hay errores ni warnings MISSING-LEFT-PARENTHESIS, EXTRANEOUS-INPUT-AFTER-LAST-PARENTHESIS, etc. CLIPS> (check-syntax "(defrule example =>)") FALSE CLIPS> (check-syntax "(defrule foo (number 40000000000000) =>)") (FALSE "[SCANNER1] WARNING: Over or underflow of long integer. ") CLIPS> (check-syntax "(defrule example (3) =>)") (" [PRNTUTIL2] Syntax Error: Check appropriate syntax for the first field of a pattern. ERROR: (defrule MAIN::example (3 " FALSE)
string-to-field	**Convertir una cadena a un campo** (string-to-field <string-or-symbol-expression>) CLIPS> (string-to-field "3.4") 3.4 CLIPS> (string-to-field "a b") a