CLIPS

* deffunction Definir funciones
* defglobal Definir variables globales
* Con reglas
  + deftemplate Para definir plantillas
  + deffacts Para definir hechos
  + defrule Para definir reglas

**Tipos de datos**

* Reales (float) 1.5,-0.7
* Enteros (integer) 1,-1
* Simbolos (symbols) Cualquier secuencia de caracteres que no empiece con un numero.Distingue entre mayúsculas y minúsculas
* Cadenas (Strings) “Deben estar entre comillas”

**Variables**

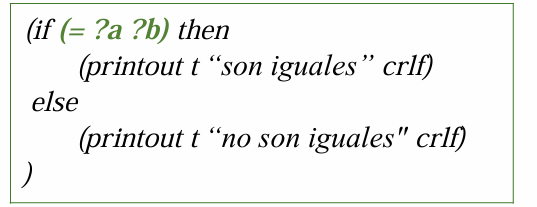
* **bind** se asigna un valor (equivalente a =)
* **variables simples para** guardar solamente un único valor, usaremos ?nombre\_var
  + **(bind ?var 15)**
* **Variables multicampo** para guardar mas de un valor (lista), usaremos $?nombre\_var
  + El comando **create$** sirve para crear un multicampo
    - (bind $?var2 hola 89 adios)
    - (bind $?var3(create$ hola 89 adios))
* **Variables globales** deben nombrarse delimitadas por \* y se usa el constructor defglobal
  + **(defglobal ?\*var1\* = 7)**
  + **(defgloabal ?\*var2\* = (create$ hola adios 5))**

**Entrada y salida**

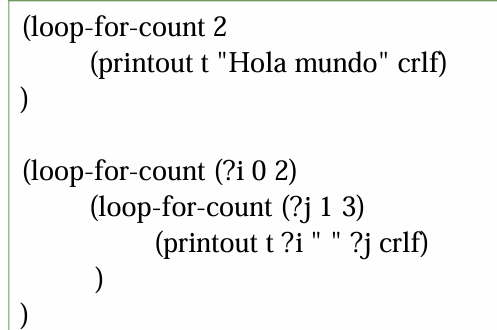
* **Entrada** la función **(read)** permite a CLIPS leer información proporcionada por el usuario.El programa se detiene a la espera de que se teclee el dato
  + **(bind ?var1(read))**
* La función read lee solo hasta el primer espacio en blanco o salto de linea
* Para leer una serie de datos separados por un espacio,se usa **(readline)**,esta función retorna una cadena (STRING)
* Para convertir un String a un conjunto de símbolos usaremos la función **explode$**
  + **(bind ?var1 (explode$(readline)))**
* **Salida** el comando **printout** permite imprimir por pantalla como por fichero
  + **(printout t “Hola mundo” crlf)**
    - **crlf** salto de línea

**Estructuras de control**

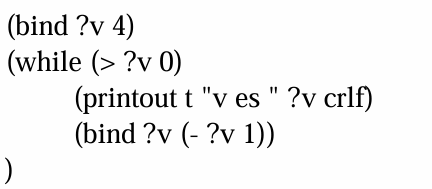
* **Sentencia condicional** 
  + (if <condicion> then
  + (<accion>)
  + [else(<accion>)])

****

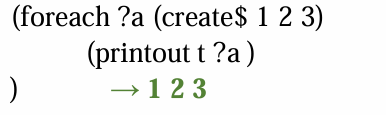
* **Sentencia repetitiva**
  + (loop-for-count (<var><inicio><final>)[do]<accion>)

****

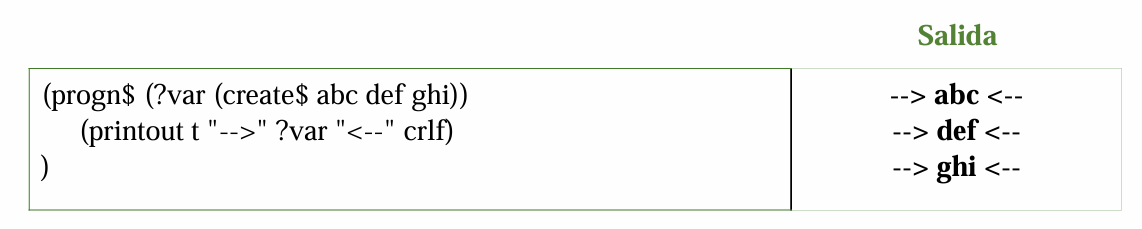
* + (while (<condicion>)[do](<accion>))

****

* + **(**foreach<var><varMulticampo>(<accion>)**)**

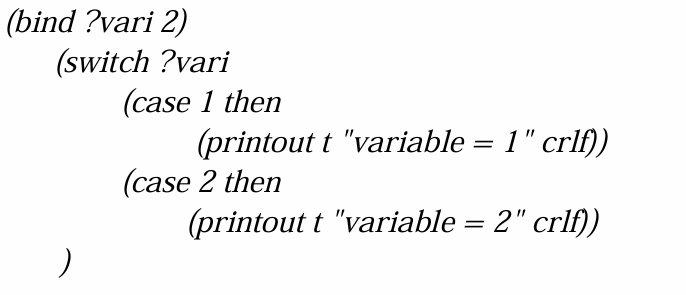
****

* **Ejemplo funciones multicampo**
  + **Progn** Similar a foreach
    - Permite realizar un conjunto de acciones sobre cada campo de un multicampo

****

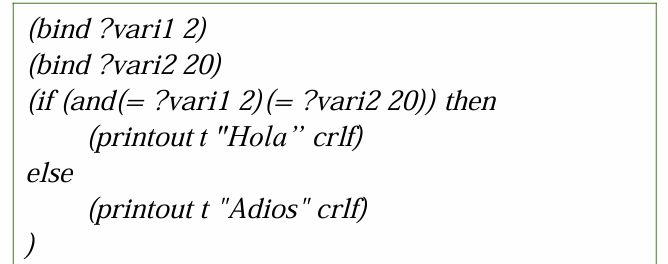
**Estructuras de control**

* **Switch**
  + (switch <expresión\_test>
  + (case <comparacion> then <accion>)
  + …
  + (case <comparacion> then <accion>)
  + [(default <accion>)]
  + )

****

**Operadores lógicos y matemáticos**

* **Operadores lógicos**  and,or,no

****

* **Operadores matemáticos** abs,div,float,integer,max,min,+,-,\*,/
  + **(abs <numero>)** devuelve el valor absoluto del numero
  + **(div <numero><numero>+)** devuelve la división (entero)
  + **(float <numero>)** devuelve el numero en float (Casting)
  + **(integer <numero>)** devuelve el numero en entero (Casting)
  + **(max|min <numero><numero>+)** devuelve el numero maximo o minimo
  + **(<op> <numero><numero>)**
    - Donde <op>:
      * + suma todos los elementos
      * \* producto de todos los elementos
      * -,/ resta o divide el primer elemento con los demás (/ devuelve float)
* **Mas funciones matemáticas** exp,log,mod,sqrt,\*\*,random
  + (exp <numero>) devuelve e elevado a la potencia del numero
  + (log <numero>) devuelve el logaritmo en base e del numero
  + (log10 <numero>) devuelve el logaritmo en base 10 del numero
  + (mod <numero><numero>) devuelve el resto de la división de los números
  + (pi) devuelve el valor de pi
  + (round <numero>) devuelve el valor del argumento al entero mas cercano
  + (sqrt <numero>) devuelve la raíz cuadrada del argumento
  + (\*\* <numero><numero>) devuelve el primer numero elevado a la potencia del segundo
  + (random <inicio><fin>) devuvle un numero entero aleatorio dentro de ese rango
  + **Trigonométricas (<f><expresión numerica>)** 
    - Donde f sin|cos|tan|cot|sec|tanh|sinh|…

**Strings/símbolos**

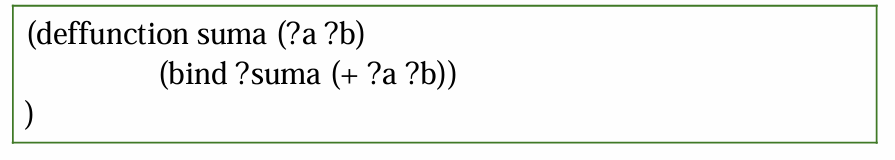
* **Funciones con Strings y/o Simbolos**
  + (lowcase|upcase <String|Simbolo>) convierte a minúsculas|mayúsculas el símbolo o string
  + (str-cat <String|Int|Simbolo>\*) concatena todos los parámetros y devuelve un String
  + (sym-cat <String|Int|Simbolo>\*) concatena todos los parámetros y devuelve un Simbolo
  + (str-compare <String|Simbolo>) devuelve el resultado de comparación
    - 0 si ambos parámetros son idénticos
    - 1 si el 1er string o símbolo >2do string o símbolo
    - -1 si el 1er string o simbolo<2do string o símbolo
  + (srt-index <String1><String2>) devuelve el índice en el que se encuentra Str1 en Str2 (puede usarse con simbolos)
* **Mas funciones con strings**
  + (str-length<String|simbolo>) devuelve la longitud
  + (sub-string<inicio><fin><String|Sim>) devuelve una subcadena (String) entre inicio y fin
  + (eval <String>) trata de convertir en función el string
    - (eval “(+ 3 4)”)
  + (string-to-field <String|Simbolo>) convierte una cadena en un campo

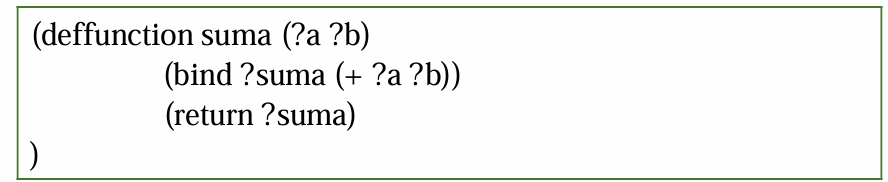
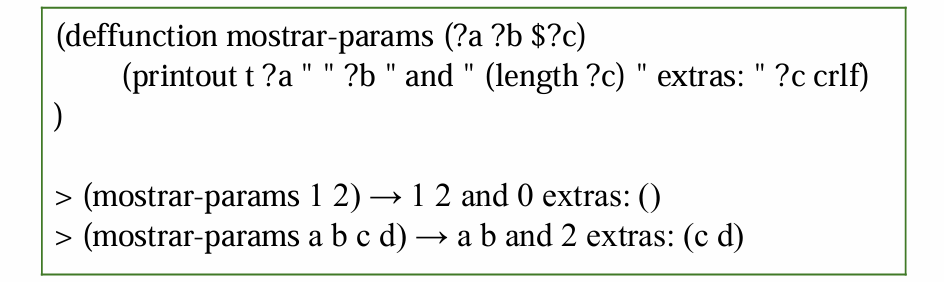
**Elementos multicampo**

* **Manipulación de valores multicampo** Todas devuelven lo creado o lo modificado
  + **(**create$ <expresion>\***)** crea un valor multicampo
    - **(create$ (+ 34) xyz “texto”)**
  + (delete$ <expresiónMulticampo><inicio><final>) borra todos los campos en ese rango
    - (delete$ (create$ a b c d e )3 4) ->(abe)
  + (explode$<string>) crea un valor multicampo a partir de los campos del string
    - (explode$ “1 2 abc \” el coche\””) -> (1 2 abc “el coche”)
  + (first$ <expresionMultiC>) Devuelve el primer campo en un multicampo
    - (first$ (create$ a b c d )) -> (a)
  + (implode$ <expresionMultiC>) Devuelve un string formado por los campos
    - (implode$ (create$ a b c d)) -> “a b c d”
  + (insert$ <expresionMultiC><pos><expresionMulti|Simple>)inserta en esa posicion
    - (insert$(create$ a b c d)1 x) ->(x a b c d)
  + (lenght$ <expreisonMultiC>) Devuelve la cantidad de campos
    - (lengtht$ (create$ a b c d)) ->4
  + (member$ <expresionSimple><expresionMultiC>) Devuelve la posición
    - (member$ z (create$ a b c d)) -> false
    - (member$ c (create$ a b c d)) -> 3
  + (nth$ <pos><expresionMultiC>) Devuelve el campo de esa posición
    - (nth$ 3(create$ a b c d)) ->c
    - (nth$ 9(create$ a b c d))->nil
  + (replace$ <expresionMultiC><inicio><fin><reemplazaMultiC|Simple>) reemplaza en ese rango
    - (replace$ (create$ a b c)2 3 x) ->(a x)
  + (rest$ <expresionMultiC>) Devuelve un valor multicampo quitando el primer element
    - (rest$ (create$ 1 2 3 4)) -> (2 3 4)
  + (subseq$ <expresionMultiC><inicio><fin>) -> devuelve un valor mutlicampo de ese rango
    - (subseq$ (create$ a b c d)3 4) -> c d
  + (subset <expresionMultiC><enEstaExpresionMultiC>) -> devuelve si el primer multicampo es parte de otro
    - (subset (create$ 1 2)(create$ 1 2 3))->true

**Funciones definidas por usuario**

* Se utiliza el constructor deffunction
  + (deffunction <nombre> [<comentario>] (<parametro>[parámetroComodin])
  + <accion>\*)



* Las funciones devuelven la ultima expresión.Se puede utilizar (return expresion)
  + 
* **list-deffunctions** permite listar las funciones definidas
* otro ejemplo
* solamente admite un multicampo y siempre ha de ser el ultimo parámetro
* ¿Cómo pasar mas de una variable multicampo?
  + En la función definimos una variable simple ?a pero en la llamada pasamos una variable multicampo

