

---

Grado en Ingeniería Informática, Universidad de Córdoba

Sistemas Inteligentes

CLIPS

# Tema 7: Funciones y funciones genéricas

Aurora Esteban Toscano  
aestebant@uco.es

José Manuel Alcalde Llergo  
i72allj@uco.es

Abril de 2023



- Aprender a implementar funciones en CLIPS.
- Aprender a implementar funciones genéricas en CLIPS.
- Conocer las principales funciones y acciones del sistema en CLIPS.



Las funciones en CLIPS agrupan una secuencia de acciones, o expresiones, a ejecutar de forma secuencial mediante una única llamada a función. Típicamente reciben una entrada en forma de parámetros de entrada y producen una salida en base a dichos parámetros.

## Definición

```
(deffunction <nombre> [comentario]
  (<parámetro regular>* [parámetro comodín])
  <acción>*
)
<parámetro regular>:: <variable mono-campo>
<parámetro irregular>:: <variable multi-campo>
```



- Las funciones reciben una lista de cero o más parámetros regulares: **obligatorios**.
- Las funciones pueden recibir parámetros adicionales recogidos en el parámetro comodín: **opcionales**.
- Las funciones deben tener un nombre único y declararse antes de ser llamadas.
- Las funciones devuelven el resultado de evaluar su última acción y FALSE si no tienen ninguna acción.
- Las funciones pueden llamarse a sí mismas: útil para implementar un **comportamiento recursivo**.



```
CLIPS> (deffunction formatea-args
```

```
  (?a ?b $?c)
```

```
  (printout t "Tengo dos parametros obligatorios: " ?a " y " ?b
```

```
  ". Ademias de " (length$ $?c) " opcionales: " $?c crlf)
```

```
)
```

```
CLIPS> (formatea-args hola adios)
```

```
Tengo dos parametros obligatorios: hola y adios. Ademias de 0 opcionales: ()
```

```
CLIPS> (formatea-args hola adios "buenas tardes" 33)
```

```
Tengo dos parametros obligatorios: hola y adios. Ademias de 2 opcionales: ("buenas tardes" 33)
```

```
CLIPS>
```



```
CLIPS> (deffunction factorial
  (?a)
  (if (or (not (integerp ?a)) (< ?a 0)) then
    (printout t "Factorial error: " ?a crlf)
  else
    (if (= ?a 0) then 1
      else (* ?a (factorial (- ?a 1)))
    )
  )
)
CLIPS> (factorial 4)
24
CLIPS> (factorial -3)
Factorial error: -3
CLIPS> (factorial adios)
Factorial error: adios
```



- (list-deffunctions): muestra todas las funciones definidas en el sistema.
- (ppdeffunction <función>): muestra la definición de una función dada.
- (undeffunction <función>): borra la función dada.



Las funciones genéricas es la forma que tiene CLIPS de programar funciones que variarán su comportamiento dependiendo de los parámetros con los que se llamen: **sobrecarga de funciones**.

- Las funciones genéricas se componen de una **cabecera** y de una lista de **métodos**: cada método tiene una lista de parámetros diferentes.
- Declaración de la cabecera:
  - Explícitamente: usando el constructor `defgeneric`.
  - Implícitamente: definiendo al menos un método de la función mediante `defmethod`.
- Los métodos tienen un comportamiento similar a las funciones "normales" de CLIPS, con la salvedad de estar agrupados en una misma *familia* y las ventajas:
  - Pueden sobrescribir o ampliar funciones del sistema.
  - Pueden tener restricciones sobre los parámetros que reciben.





## Cabecera

```
(defgeneric <nombre> [comentario])
```

## Métodos

```
(defmethod <nombre> [índice] [comentario]
  (<restriccion mono-campo>* [restriccion multi-campo])
  <acción>*
)

<restriccion mono-campo> :: <variable mono-campo> | (<variable
  mono-campo> <tipo>* [consulta])
<restriccion multi-campo> :: <variable multi-campo> | (<variable
  multi-campo> <tipo>* [consulta])
<tipo> :: INTEGER | FLOAT | NUMBER | SYMBOL | STRING | LEXEME
<consulta> :: <variable global> | <llamada a función>
```



- Identificación de un método:
  - Nombre + lista de parámetros.
  - Nombre + índice.
- A cada método se le asigna un índice único dentro de una función genérica.
- Si se define un método con el mismo nombre y parámetros que otro existente, se sobrescribe el más antiguo.
- Reemplazar un método por otro con una definición de parámetros distinta: especificar el índice del método a reemplazar al definir el nuevo.
- Restricciones sobre parámetros:
  - De *tipo*: limita los tipos de datos admisibles.
  - De *consulta*: comprobación lógica sobre el argumento que debe devolver distinto de FALSE.
  - Las restricciones se evalúan de izquierda a derecha.
  - Si una restricción no se cumple, dejan de comprobarse las que siguen.



- Ningún parámetro: `(defmethod m1 ())`
- Un parámetro de cualquier tipo: `(defmethod m1 (?a))`
- Un parámetro entero: `(defmethod m1 ((?a INTEGER)))`
- Un parámetro entero o símbolo: `(defmethod m1 ((?a INTEGER SYMBOL)))`
- Un parámetro entero y par: `(defmethod m1 ((?a INTEGER (evenp ?a))))`
- Un parámetro entero o símbolo y otro de cualquier tipo: `(defmethod m1 ((?a INTEGER SYMBOL) ?b))`



```
CLIPS> (defmethod + ((?a STRING) (?b STRING))  
  (str-cat ?a ?b))  
CLIPS> (defmethod + ((?a STRING) (?b NUMBER))  
  (printout t "No se que hacer con eso." crlf))  
CLIPS> (list-defmethods)  
+ #SYS1 (NUMBER) (NUMBER) ($? NUMBER)  
+ #2 (STRING) (STRING)  
+ #3 (STRING) (NUMBER)  
CLIPS> (+ "Hola " "Mundo")  
"Hola Mundo"
```

- **Ejercicio:** ¿Qué pasará si se llama a la función con (+ <número> <cadena>)?
- **Ejercicio:** ¿Cómo se extendería la función para que si se llama con dos símbolos los devuelva concatenados con un espacio en medio?



*Generic dispatch* o *ejecución genérica*: proceso por el que CLIPS decide qué método de la función genérica ejecutar según la llamada realizada.

- Un método es aplicable a una llamada de función genérica si:
  - 1 Su nombre coincide con el de la función genérica.
  - 2 Acepta al menos tantos argumentos como se pasan en la llamada.
  - 3 Cada argumento satisface las restricciones de parámetro correspondientes.
- La precedencia entre dos métodos se determina mediante sus restricciones de parámetro: el método con las restricciones más específicas tiene más precedencia.



```
CLIPS> (defmethod m1 () m1-1)
CLIPS> (defmethod m1 (?a) m1-2)
CLIPS> (defmethod m1 ((?a INTEGER)) m1-3)
CLIPS> (defmethod m1 ((?a INTEGER SYMBOL)) m1-4)
CLIPS> (defmethod m1 ((?a INTEGER STRING)) m1-5)
CLIPS> (defmethod m1 ((?a INTEGER (evenp ?a))) m1-6)
CLIPS> (m1)
m1-1
CLIPS> (m1 2.5)
m1-2
CLIPS> (m1 25)
m1-3
CLIPS> (m1 hola)
m1-4
CLIPS> (m1 22)
m1-6
```



- (list-defgenerics): muestra todas las funciones genéricas definidas en el sistema.
- (ppdefgeneric <función>): muestra la definición de una función genérica dada.
- (undefgeneric <función>): borra la función genérica dada, incluyendo todos sus métodos.
- (list-defmethods <función>): muestra los métodos definidos para una función genérica.
- (ppdefmethod <función> <índice>): muestra la definición de un método de una función genérica.
- (undefmethod <función> <índice>): borra un método de una función genérica dada.



CLIPS predefine múltiples funciones que pueden usarse para las acciones que se programen tanto en funciones del usuario como en funciones genéricas.

- Funciones predicado
- Funciones matemáticas
- Funciones multi-campo
- Funciones de cadena
- Funciones de entrada/salida
- Funciones procedurales

## OjO

Recuerda que tanto las funciones como los métodos de las funciones genéricas pueden contener varias acciones en su cuerpo. Devolverán el resultado de su última acción o FALSE si no hay acciones.





- Tipo de dato: (integerp | floatp | numberp | symbolp | stringp | lexemep <expresión>)
- Es multi-campo: (multifieldp <expresión>)
- Son misma expresión o distinta: (eq | neq <expresión> <expresión>+)
- Número par o impar: (evenp | oddp <expresión>)
- Comparaciones numéricas: (= | <> | < | <= | > | >= <expresión> <expresión>+)
- Lógica booleana: (not <expresión>), (and | or <expresión>+)



- Suma, resta, multiplicación o división: (+ | - | \* | / <expresión> <expresión>+)
- División entera: (div <expresión> <expresión>)
- Resto o módulo de la división: (mod <expresión> <expresión>)
- Raíz cuadrada: (sqrt <expresión>)
- Potencia: (\*\* <expresión> <expresión>)
- Redondeo: (round <expresión>)
- Valor absoluto: (abs <expresión>)
- Máximo o mínimo de un conjunto: (max | min <expresión>+)



- Crear un multi-campo: (create\$ <expresión>\*)
- Longitud de un multi-campo: (length\$ <expresión multi>)
- Obtener  $n$ -ésimo: (nth\$ <posición> <expresión multi>)
- Es miembro: (member\$ <expresión> <expresión multi>)
- Es subconjunto: (subsetp <expresión multi> <expresión multi>)
- Extraer subconjunto: (subseq\$ <expresión multi> <posición ini> <posición fin>)
- Extraer primero: (first\$ <expresión multi>)
- Todos menos el primero: (rest\$ <expresión multi>)



- Transformar una cadena en multicampo: (explode\$ <cadena>)
- Tranformar un multicampo en cadena: (implode\$ <expresión multi>)
- Insertar: (insert\$ <expresión multi> <posición ini> <expresión>+)
- Borrar: (delete\$ <expresión multi> <posición ini> <posición fin>)
- Reemplazar: (replace\$ <expresión multi> <posición ini> <posición fin> <expresión>+)



- Concatenar en una cadena: (str-cat <expresión>+)
- Concatenar en un símbolo: (sym-cat <expresión>+)
- Longitud de una cadena o símbolo: (str-length <expresión>)
- Extraer cadena: (sub-string <posición ini> <posición fin> <cadena>)
- Comparación lógica de cadenas o símbolos: (str-compare <expresión> <expresión>)



- Abrir un fichero: (`open <fichero> <nombre> <modo>`)
  - Posibles modos: "r"(lectura), "w"(escritura), "r+"(lectura y escritura), "a"(añadir al final),
- Cerrar un fichero: (`close <nombre>`)
- Escribir en un fichero: (`printout <nombre> <expresión>+`)
  - Escritura por consola: (`printout t <expresión>+`)
- Leer de un fichero: (`read <nombre>`)
  - Leer desde la consola la entrada del usuario (`read`)
- Leer una línea de un fichero: (`readline <nombre>`)



- Definir una variable: (bind <variable> <expresión>)
- Sentencia condicional: (if <expresión> then <acción>\* [else <acción>\*])
- Sentencia condicional múltiple: (switch <expresión prueba> (case <expresión comprobación> then <acción>\*)+ [(default <acción>\*)])
- Bucle mientras: (while <expresión> [do] <acción>\*)
- Bucle para: (loop-for-count <rango> [do] <acción>\*)  
<rango>:: <idx fin> | (<variable> <idx fin>) | (<variable> <idx ini> <idx fin>)
- Salir de una función: (return [<expresión>])
- Salir de un bucle: (break)

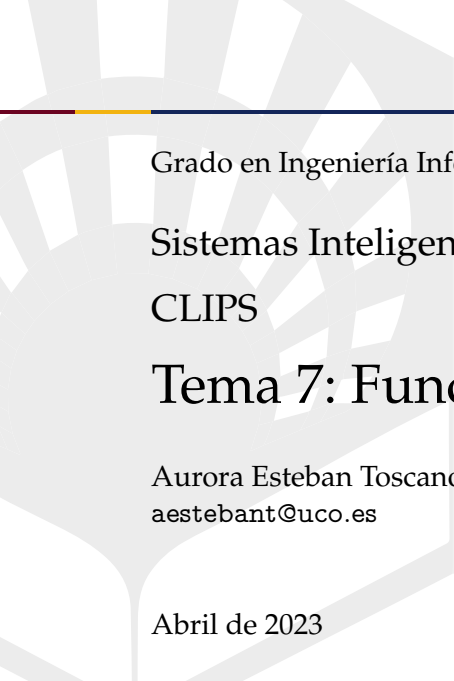


- Estudia un ejemplo de uso de bucles en funciones en el fichero `bucles.clp` disponible en Moodle.
- Estudia un ejemplo de petición de datos por pantalla en el fichero `transformaciontemporal.clp` disponible en Moodle.





- 1 Crear una función genérica para extraer  $N$  elementos de una cadena o de un valor multicampo. La función recibirá como parámetros en los dos casos el índice del primer y último elemento a extraer y como tercer parámetro la cadena o multicampo sobre los que realizar la extracción.
- 2 Crear una función que, utilizando recursividad, calcule el  $n$ -ésimo número par (considerando el 2 como primer par). La función recibe un parámetro, la posición que se desea obtener.
- 3 Crear una función que, utilizando recursividad, calcule el  $n$ -ésimo número impar (considerando el 1 como primer impar).
- 4 Crear una función que, utilizando recursividad, calcule la suma de los  $n$  primeros números pares.
- 5 Crear una función que, utilizando recursividad, calcule el  $n$ -ésimo término de la sucesión de números triangulares.



---

Grado en Ingeniería Informática, Universidad de Córdoba

Sistemas Inteligentes

CLIPS

# Tema 7: Funciones y funciones genéricas

Aurora Esteban Toscano  
aestebant@uco.es

José Manuel Alcalde Llergo  
i72allj@uco.es

Abril de 2023