Contenidos

- El lenguaje CLIPS: símbolos, valores, variables
- Hechos (facts)
  - **A) Hechos ordenados**
  - **B) Hechos no ordenados**: deftemplate
- La memoria de trabajo
  - Hechos iniciales: deffacts
  - Etiquetas temporales
- Reglas. Patrones y acciones
- Operaciones de Entrada y Salida interactivas
- Utilidades auxiliares. Instrucciones de control.
- Estrategias de resolución de conflictos
- Estructura básica de un programa en CLIPS
- Materiales de referencia

El lenguaje CLIPS

- **CLIPS** es un lenguaje que permite **construir sistemas basados en reglas**
  - **C language integrated production system**
  - Su sintaxis es tipo Lisp: listas de símbolos
  - Entre símbolos puede haber cualquier número de espacios en blanco
- Se distingue entre mayúsculas y minúsculas
- Los símbolos
  - Pueden incluir: a-z A-Z 0-9 $ * . = + / < > _ ? #
  - No pueden empezar por 0-9 o caracteres reservados para usos especiales ($? &)
- Símbolos especiales:
  - nil, TRUE, FALSE, crlf
- Comentarios
  - **Líneas que empiezan por ;**
  - Si el comentario abarca varias líneas ***/* comentario */***

Valores

- Los valores en CLIPS pueden ser:
  - Símbolos
    - Juan amarillo respuesta22 _ejemplo
  - Números
    - 56 47.8 5654L 6.0E4
  - Cadenas
    - “esto es un ejemplo”
  - Listas de símbolos, números, cadenas
    - (a b c d) (+ 3 5) (“Pregunta 1 “ “Nombre”) () (eq (3 5))

Variables

- Las variables empiezan por ?

  Ejemplos: ?pregunta ?nombre ?edad
  - Las variables no son tipadas aunque los valores lo sean
  - Para asignar un valor a una variable se usa la función bind

  (bind ?edad 18)
- Variables multivaluadas: empiezan por $?
  - **$?**apellido
- Variables globales. Se definen con la función
  - (defglobal ?*variable* = valor-por-defecto)

  (defglobal ?*x* = 7)

A) – Hechos ordenados

- Secuencia de literales separados por espacios
  - Codifican la información según la posición
  - El primer literal suele representar una relación entre los restantes
  - Los restantes son como atributos pero sin nombre (solo valores)

```
(convenio)
(alumnos Juan Luis Pedro)
(lista-de-la-compra pan leche arroz)
```

- Para incluirlos en la Base de Hechos **se asertan** (no se declaran)

```
(assert (alumnos Juan Luis Pedro))
(assert (temperatura 25) )
```

- Para que exista encaje o *matching* con el LHS de una regla
  - Los literales deben estar en **el mismo orden** que en la regla
  - Se usan para manejar poca información

B) - Hechos no ordenados: deftemplate

- Define **un tipo** de hecho que tiene varios slots (atributos) con nombre

```
(deftemplate persona
      (slot nombre (type SYMBOL))
      (multislot apellidos (type SYMBOL))
      (slot edad (type NUMBER)(default (+ 10 15)))
      (slot estado (type SYMBOL)(allowed-values soltero
                  libre casado viudo)(default soltero)))
```

- Para cada slot se puede definir:
  - el tipo: type
    - (valores posibles: SYMBOL, FLOAT, INTEGER, STRING, NUMBER)
  - Valor por defecto: default (admite cualquier operación)
  - Valores permitidos: allowed-values
  - Slot multivaluados: multislot
- Se necesita establecer cada hecho con **assert** o **deffacts**

Gestión de la Memoria de Trabajo (MT)

- **(deffacts …)** define un conjunto de hechos iniciales que se cargan en la MT al hacer (reset)
- **(assert <hecho>)** añade hecho a la MT

```
(assert (temperatura 25) )
```

- **(retract <índice-hecho>)** elimina hecho de la MT
- **(facts)** lista los hechos existentes en la MT
- **(clear)** elimina todos los hechos de la MT
- **(reset)**
  - elimina todos los hechos de la MT y las activaciones de la agenda
  - añade **initial-fact** y los hechos definidos con **deffacts**
  - añade las variables globales con su valor inicial
  - selecciona el módulo main

Crear hechos iniciales con deffacts

- Ejemplo

```
(deffacts alumnos "mi clase"    /* -- hechos iniciales --*/
  (persona (nombre Pepe)(apellidos Gomez Garcia))
  (persona (nombre Juan)(edad 25)))
```

→ **No olvidar hacer:**

```
(reset) /*- borra todos los hechos de MT, añade los deffacts */
(facts) /*---- lista los hechos actuales en la M.T. --*/
f-0   (MAIN::initial-fact)
f-1   (MAIN::persona (nombre Pepe) (edad 25) (estado soltero) (apellidos Gomez Garcia))
f-2   (MAIN::persona (nombre Juan) (edad 25) (estado soltero) (apellidos ))
```

- Si quiero volver a ejecutar deffacts, debo ejecutar reset de nuevo

Etiquetas temporales

- Son índices relativos al orden de creación de hechos
- f - 0 es el initial-fact, creado automáticamente por CLIPS

```
CLIPS> (facts)
f-0     (initial-fact)
f-1     (persona (nombre juan) (edad 18) (sexo h) (estatura 180) (peso 70) (
f-2     (persona (nombre maria) (edad 18) (sexo m) (estatura 170) (peso 60)
f-3     (persona (nombre pepe) (edad 25) (sexo h) (estatura 190) (peso 80) (
f-4     (persona (nombre luisa) (edad 18) (sexo m) (estatura 170) (peso 60)
f-5     (plato (nombre macarrones) (tipo primero) (mayoritario cereales) (ca
f-6     (plato (nombre sopa) (tipo primero) (mayoritario cereales) (calorias
f-7     (plato (nombre consome) (tipo primero) (mayoritario caldo) (calorias
f-8     (plato (nombre menestra) (tipo primero) (mayoritario verdura) (calor
f-9     (plato (nombre filete) (tipo segundo) (mayoritario carne) (calorias
f-10    (plato (nombre merluza) (tipo segundo) (mayoritario pescado) (calori
f-11    (plato (nombre flan) (tipo postre) (mayoritario huevos) (calorias 20
f-12    (plato (nombre manzana) (tipo postre) (mayoritario fruta) (calorias
For a total of 13 facts.
CLIPS>
```


Reglas

- Sintaxis:

```
(defrule <nombre-regla>      ;; para eliminar : undefrule
[<documentación opcional>]   ;; Se pone entre " "
[(declare (salience <num>))]  ;; prioridad de ejecución
(patrón 1)
(patrón 2)
...
(patrón N)
=>
(acción 1)
(acción 2)
...
(acción M)
)
```

- Ver el contenido de una regla **`(ppdefrule calcular-precio)`**
- Una regla sin LHS se ejecuta solo cada vez que se ejecute el **reset**.
  - **`Reglas de inicialización`**

Parte izquierda de las reglas: Patrones

- Las condiciones de la parte izquierda de una regla están implícitamente conectadas con and. Si necesitamos un or entonces hay que dividir la regla en dos.
- La parte izquierda de las reglas suelen incluir patrones variables:
  - Variables (?edad)
  - Variables anónimas (comodines, no importa su valor)
    - ? Se equipara con un valor
    - $? Se equipara con múltiples valores
  - Expresiones con variables y conectivas lógicas
    - not (~), and (&), or (|)
  - Test de expresiones lógicas (test (< ?x 18))
  - Condiciones complejas precedidas de :
    - (persona (edad ?x&: (> ?x 18)))

Parte derecha de las reglas: Acciones

- Son acciones implícitamente conectadas con and
- Tipos de acciones:
  - Crear un hecho (assert)
  - Eliminar un hecho (retract)
  - Modificar un hecho (modify)
  - Llamar a una función
  - Asignar un valor a una variable (bind)
  - Entrada / Salida (printout, read, readline)
  - Parar la ejecución (halt)

Ejemplo sin variables

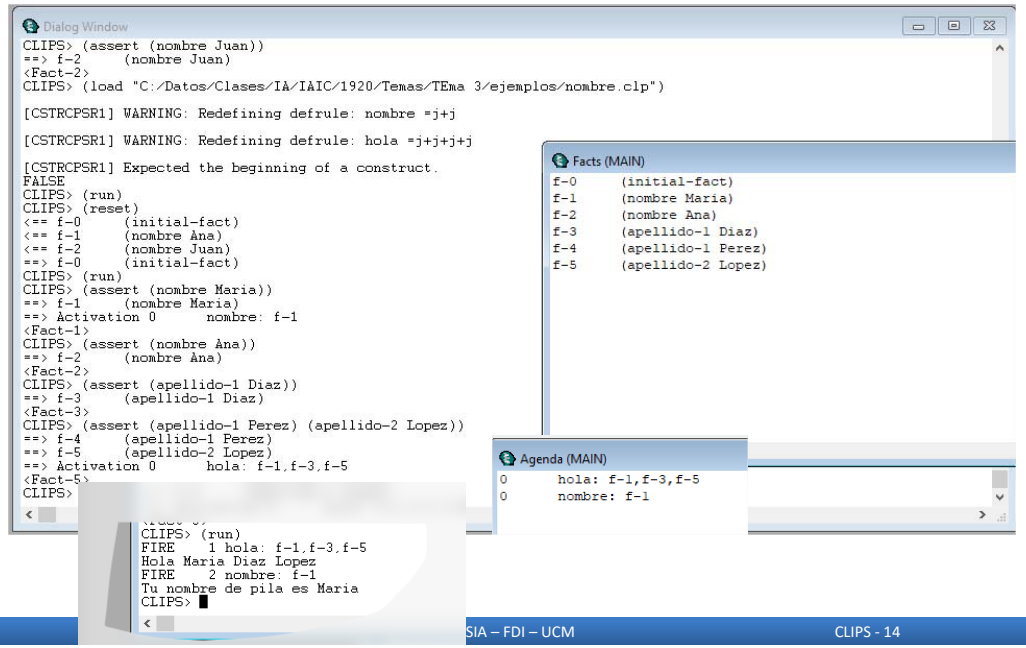
```
(defrule nombreMaria
  (nombre Maria)
  =>
  (printout t "Tu nombre de pila es Maria" crlf))
```

```
(assert (nombre Ana))
(assert (nombre Juan))
(assert (apellido-1 Diaz))
(assert (apellido-1 Perez) (apellido-2 Lopez))
(assert (apellido-1 Martinez))
```

```
(assert (nombre Maria))
```

```
(defrule Hola
        (nombre Maria)
        (apellido-1 Diaz)
        (apellido-2 Lopez)
        =>
        (printout t "Hola Maria Diaz Lopez" crlf))
```


Toma de contacto de la práctica 4



Regla con variables

- Permite usar una regla con diferentes objetivos:
  - busco: que cumplan cierto estado y tengan menos de 30 años
- Al repetir **?est** fuerza que coincida su valor en los hechos que equiparen

```
(deftemplate busca (slot estado))  ;; un hecho para indicar qué busco
(defrule busca-candidato
    (busca (estado ?est)) ;; persona y busca han de tener el mismo ?est
    ?candidato <- (persona (nombre ?nom) (edad ?ed) (estado ?est))
                      ;;  asigno un hecho a la variable ?candidato
    (test (< ?ed 30))  ;; solo ejecuto regla si cumple test
  =>
    (modify ?candidato (estado libre)) ;; cambia el estado de candidato
    (assert (tengo candidato))
    (printout t "mi candidato es: " ?nom " estado anterior: " ?est crlf) )

(reset)
(assert (busca (estado soltero))) ;; decido buscar candidatos solteros
(run)
```


Ejemplo modificación hecho

- Ejecución condicional : para poner apellidos a quien no tenga

```
(defrule poner-apellidos ;; solo equiparan personas sin apellidos
   ?persona <-(persona (nombre ?nombre) (apellidos))
=>
   (printout t crlf "Introduce apellidos para " ?nombre ": " )
   (modify ?persona (apellidos (read))))
```

Busco los hechos que encajan y me quedo con su índice

```
CLIPS> (facts)
f-0     (initial-fact)
f-1     (persona (nombre juan) (edad 18) (sexo h) (estatu
f-2     (persona (nombre maria) (edad 18) (sexo m) (estat
f-3     (persona (nombre pepe) (edad 25) (sexo h) (estatu
f-4     (persona (nombre luisa) (edad 18) (sexo m) (estat
f-5     (plato (nombre macarrones) (tipo primero) (mayori
f-6     (plato (nombre sopa) (tipo primero) (mayoritario
f-7     (plato (nombre consome) (tipo primero) (mayoritar
f-8     (plato (nombre menestra) (tipo primero) (mayorita
f-9     (plato (nombre filete) (tipo segundo) (mayoritari
```

Operaciones de Entrada y Salida interactivas - I

- Permite introducir un hecho entre comillas

```
(defrule inserta-hecho   ; no tiene LHS: se dispara si (reset) + (run)
        =>               ; para escribir un texto al disparar regla
     (printout t "Escribe un hecho como cadena" crlf)
     (assert-string (read)))  ; para leer un hecho
(reset)
(run)


Escribe un hecho como cadena        ; el ordenador escribe esto
“(persona (nombre NEO)) ”           ; el usuario escribe eso entre “”
                                    ; el sistema añade el hecho:
;;   (persona (nombre NEO) (apellidos ) (edad 25) (estado soltero))
;; los atributos apellidos, edad y estado están definido en el template
   persona
;; si tiene valor por defecto se pone
```


Operaciones de Entrada y Salida interactivas – I I

- Permite introducir una línea y construir un hecho concatenando paréntesis

```
(defrule lee-linea
        =>
      (printout t "Introduce datos." crlf)
      (bind ?cadena (readline))
      (assert-string (str-cat "(" ?cadena ")")))

Introduce datos                     ; el ordenador escribe esto
persona                             ; el usuario escribe eso
                                    ; el sistema añade el hecho:
; (persona (nombre "sin nombre") (apellidos ) (edad 25) (estado soltero))

Lo que hemos hecho es:
(assert (persona))

Y todos los atributos tienen el valor por defecto
```


Listas

- Una lista es una secuencia ordenada de valores
- Creación de listas: función create$
  - (bind ?ejemplolista (create$ a b c d e))
- Manejo de listas:
  - (nth$) devuelve el enésimo elemento
  - (first$) devuelve el primer elemento
  - (rest$) devuelve la lista sin su primer elemento

Funciones

```
(deffunction calculaCalorias (?peso ?altura ?edad ?sexo ?actividadFisica)
(if (= ?sexo h) then
(bind ?resultado (* (- (+ 66 (* 13.7 ?peso) (* 5 ?altura)) (* 6.8 ?edad))
?actividadFisica))
else
(bind ?resultado (* (- (+ 665 (* 9.6 ?peso) (* 1.8 ?altura)) (* 4.7 ?edad))
?actividadFisica)))
(return ?resultado))
```

- Ejemplos de funciones predefinidas:
  - (facts) lista todos los hechos presentes en la memoria de trabajo
  - (rules) lista todas las reglas que haya en la base de reglas
  - (watch) activa los mecanismos de depuración en CLIPS

Otras utilidades auxiliares

Listar las templates definidas (nativas y definidas por el usuario)

```
(list-deftemplates)
```

Trazar lo que va pasando (watch , unwatch)

```
(watch facts)(watch rules)(watch activations)(watch all)
```

Comprobar el tipo de una expresión o valor

```
(numberp <exp>) (stringp <exp>) (integerp <exp>)
```


Instrucciones de control (representación no declarativa)

- (if <exp> then <accion>* [elif <exp> then <accion>*]* [else <accion>*])
- (while <exp> [do] <accion>*)
- (foreach <var> <lista> <accion>*)
- (declare salience <num>)
  - definiendo esta propiedad dentro de una regla, se establece su prioridad. Cuanto mayor sea el número mayor será la prioridad de la regla.

Estrategias de resolución de conflictos

- El motor de inferencia de CLIPS disparará las reglas aplicables por orden decreciente de prioridad (salience).
- Si hay varias reglas aplicables con la misma prioridad (o no se han definido prioridades) CLIPS utilizará **por defecto** la estrategia LIFO (depth): disparar antes las reglas activadas más recientemente.
- La estrategia FIFO (breadth) dispara las reglas de igual prioridad en el orden en que han sido activadas. Primero las que antes se activaron.
- (set-strategy <estrategia>) permite seleccionar la estrategia que utilizará el intérprete de CLIPS (depth o breadth)

Estructura básica de un programa en CLIPS

```
; definición de plantillas
(deftemplate …)
…
; definición de hechos iniciales
(deffacts …)
…
; definición de reglas
(defrule …)
…
(reset)
(run)
```

https://www.csie.ntu.edu.tw/~sylee/courses/clips/rhs.htm

Funciones útiles

- Cadenas

```
(str-cat "use" "ful" "ness")
(sym-cat use ful ness)
(str-index "def" "abcdefghi")
(sub-string 4 6 "abcdefghijkl")
(upcase "This is a test of upcase")
(lowcase A_Word_Test_for_Lowcase)
```

- Multislot

```
create$
(create$) ;()
(create$ a b c d e f) ;(a b c d e f)
first$ (first$ (create$ a b c d) ) ;(a)
rest$ (rest$ (create$ a b c d) ) ;(b c d)
length$ (length$ (create$ a b c d) ) ;4
nth$ (nth$ 3 (create$ a b c d e f g)) ;c
member$ (member$ blue (create$ red 3 "text" 8.7 blue)) ;5
(member$ 4 (create$ red 3 "text" 8.7 blue)) ;FALSE
subsetp (subsetp (create$ hammer saw drill) (create$ hammer drill wrench pliers
saw)) ;TRUE
subseq$ (subseq$ (create$ hammer drill wrench pliers) 3 4) ;(wrench pliers)
explode$ (explode$ "a b c d") ;(a b c d)
implode$ (implode$ (create$ a b c d e)) ;"a b c d e"
expand$ (printout t (expand$ (create$ a b c))) ;abc (printout t (create$ a b c))
;(a b c)
insert$ (insert$ (create$ a e f) 2 (create$ b c d)) ;(a b c d e f)
delete$ (delete$ (create$ a b c d e f) 3 4) ;(a b e f)
replace$ (replace$ (create$ a b c d) 2 3 x y (create$ q r s)) ;(a x y q r s d)
```


Funciones útiles

```
(gensym*) ;return a unique word gen<number> each time it's called.
(setgen 5) ;to set starting number used by gensym
```

I/O Functions

read

```
(defrule player-select (phase choose-player) => (printout t "Who
moves first (Computer: c " "Human: h)? ") (assert (player-select
(read))))
```

open

```
(open "input.dat" data "r") ;"r" for read only ;"r+" for
read/write ;"a" for append only ;"w" for write only
```

close

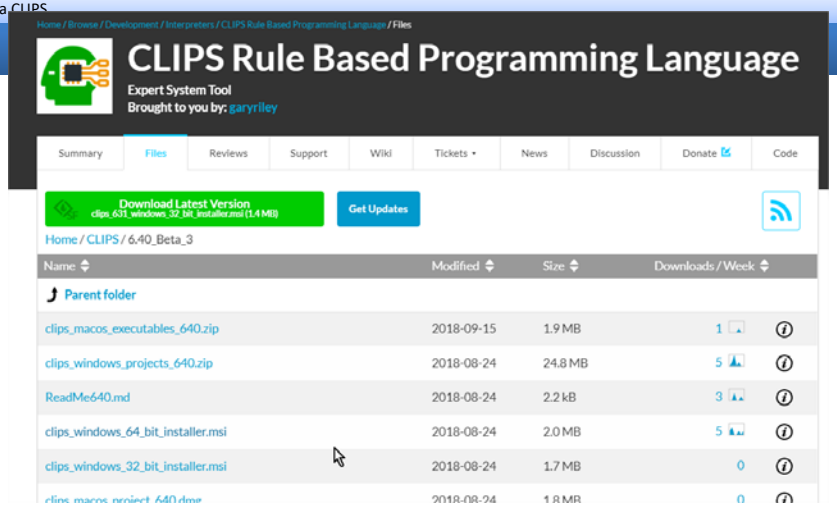
```
(open "example.dat" xmp "w") (printout xmp "green" crlf)
(printout xmp 7 crlf) (close xmp) (open "example.dat" xmp "r")
(read xmp) (read xmp) (close xmp)
```

format

```
(format t "Name: %-15s Age: %3d" "Bob Green" 35) (bind ?name
(format nil "Name: %-15s Age: %3d" "Bob Green" 35)) (printout t
?name)
```

readline

```
(readline xmp)
```

https://sourceforge.net/projects/clipsrules/files/CLIPS/6.40/

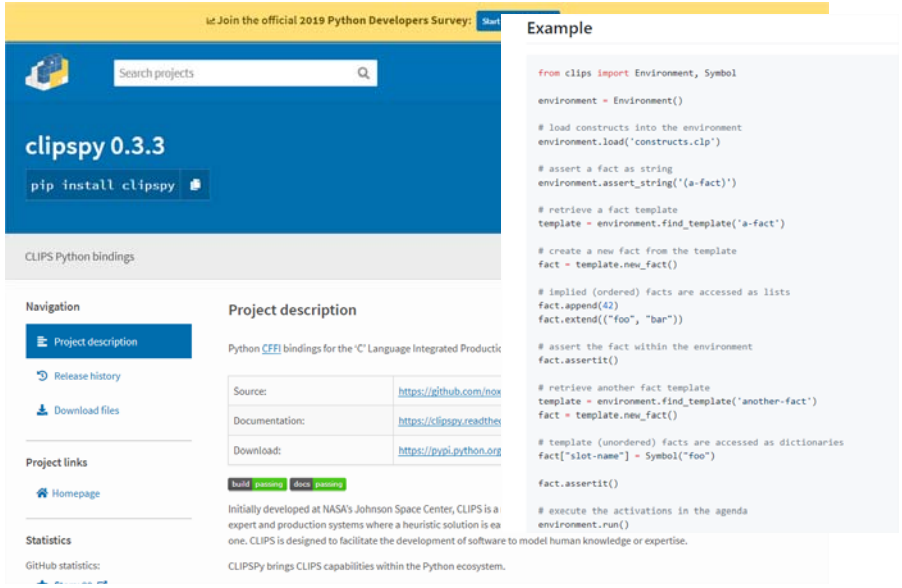
Agenda (MAIN)

```
0      ascendiente: f-12
0      hija: f-7
0      hermana: f-7,f-6
0      hermano: f-6,f-7
0      padre: f-6
0      madre: f-6
0      hijo: f-6
0      padre: f-5
0      madre: f-5
0      hijo: f-5
0      padre: f-4
0      madre: f-4
0      hijo: f-4
0      padre: f-3
0      madre: f-3
0      hija: f-3
```

Facts (MAIN)

```
f-0     (initial-fact)
f-1     (dd juan maria rosa m)
f-2     (dd juan maria luis h)
f-3     (dd jose laura pilar m)
f-4     (dd luis pilar miguel h)
f-5     (dd miguel isabel jaime h)
f-6     (dd pedro rosa pablo h)
f-7     (dd pedro rosa ana m)
f-8     (padre pedro ana)
f-9     (progenitor pedro ana)
f-10    (ascendiente pedro ana)
f-11    (madre rosa ana)
```

Load archivo
Visualizar la agenda y la MT



Materiales de referencia

- Documentación online sobre CLIPS
  - http://clipsrules.sourceforge.net/OnlineDocs.html
- Tutorial Universidad de Córdoba
  - http://www.uco.es/users/sventura/misc/TutorialCLIPS/Reglas.htm
- Otro tutorial
  - https://www.csee.umbc.edu/portal/clips/tutorial/