Si tenemos una plantilla
(deftemplate persona
(multislot nombre)
(slot edad))
El patrón (persona (nombre Juan)) emparejaría con el hecho
(persona (nombre Juan))

**TRUE**

---

En la siguiente regla:
(defrule encontrardatos
(datos 1 azul rojo)
=>
)
el elemento condicional patrón contiene sólo restricciones literales
**TRUE**

---

La siguiente regla
(defrule encontrardatos
(datos 1 azul rojo) => )
con los siguientes hechos no se activaría
f0 (initialfact)
f1 (datos 1.0 azul "rojo")
f2 (datos 1 azul)
f3 (datos 1 azul rojo)
f4 (datos 1 azul ROJO)
f5 (datos 1 rojo azul)
f6 (datos 1 azul rojo 6.9)
**FALSE**

---

El siguiente ECP (dato $? VERDE $?) no emparejaría con
todos los siguientes hechos:
(dato VERDE)
(dato VERDE rojo azul)
(dato rojo VERDE azul)
(dato rojo azul VERDE)
(dato VERDE azul VERDE)
**FALSE**

---

La siguiente regla dará error en tiempo de ejecución
(defrule prueba
=>
(printout t ?x crlf))
**TRUE**

---

Los elementos condicionales patrón utilizan los conectores lógicos & (and), |
(or), ~ (not)
y el orden de precedencia entre ellos es ~, &, |
**TRUE**

La siguiente regla muestra un elemento condicional patrón con una restricción predicado
(defrule r1
(datos ?x&:(numberp ?x))
=>
)
**TRUE**

---

En la siguiente regla:
(defrule encontrardatos
(datos 1 azul rojo)
=>
)
el elemento condicional patrón se comparará con hechos ordenados y su primer campo debe ser un símbolo.
**TRUE**

---

Con la siguiente plantilla
(deftemplate calificacion
(slot alumno (type STRING))
(slot asignatura (type LEXEME))
(slot nota (type NUMBER) (range 0 10)))
sería correcta la siguiente afirmación de hechos
CLIPS>(assert (calificacion(alumno "José López") (asignatura Lengua) (nota 11)))

**FALSE**

---

La ejecución de la orden (clear) elimina toda la información del entorno de CLIPS, concretamente:
1. Vacía la agenda
2. Vacía la lista de hechos
3. Vacía la base de conocimiento
4. Borra todas las definiciones hechas con constructores
**TRUE**

---

La ejecución de la orden (reset) lleva a cabo las siguientes acciones:
1. Borra la base de conocimiento y todas las activaciones de la agenda
2. Borra todos los hechos de la lista de hechos
3. Afirma el hecho (initial-fact)
4. Afirma todos los hechos definidos mediante constructores deffacts
**FALSE**

---

Además de con hechos definidos a partir de plantillas y hechos ordenados, la información en Clips se puede representar con variables globales y con objetos (instancias de clases)

**TRUE**

---

Con la siguiente plantilla
(deftamplate persona
(slot nombre (type LEXEME))
(slot edad (type INTEGER SYMBOL)))
sería correcto la orden
(assert (persona (nombre Juan) (edad 20.5)))
**FALSE**

---

La información el Clips se puede representar a través de hechos ordenador, hechos definidos a partir de plantillas y hechos mixtos

**FALSE**

---

Al definir los campos de una plantilla, CLIPS siempre obliga que se indique va a tratas de un campo monovalor o de un campo multivalor

**TRUE**

---

Los siguientes hechos son hechos ordenados. En el hecho f-0 el tipo de datos del segundo campo es cadena de caracteres y en el hecho f-1 el segundo campo es de tipo símbolo
f-0 (libro "El Quijote")
f-1 (arbol encina)

**TRUE**

---

(deftamplate datos
(slot w (default ?NONE))
(slot x (default ?DERIVE))
(slot y (default (gensym*)))
(slot z (default-dynamic (gensym*))))
Con la siguiente plantilla sería correcto la orden
CLIPS> (assert(datos))

**FALSE**

---

Con la siguiente plantilla
(deftamplate calificacion
(slot alumno (type STRING))
(slot asignatura (type LEXEME))
(slot nota (type NUMBER) (range 0 10)))
Sería correcta la siguiente afirmación de hechos:
CLIPS> (assert (calificacion(alumno "Juan Pérez")
(asignatura Lengua)
(nota 7.5)))
**TRUE**

---

Con la siguiente plantilla
(deftemplate persona
(slot nombre (type LEXEME))
(slot edad (type INTEGER SYMBOL)))
Sería correcto la siguiente orden de clips
CLIPS> (assert (persona (nombre "Juan") (edad treinta)))
**TRUE**

---

El siguiente código corresponde a un hecho no ordenado o definido a través de una plantilla
(coche (marca Ford) (modelo focus) (color gris))
**TRUE**

---

En la siguiente regla
(defrule rl
(hombre Socrates)
(assert (mortal ?x))
(hombre Socrates) representa un hecho ordenado

**FALSE**

---

La orden (clear) borra completamente la memoria de trabajo de clips tanto de la base de hechos como la base de conocimiento.

**TRUE**

---

En las siguientes frases:
1) Todos los hombre son mortales
2) Sócrates es un hombre
La primera corresponde a conocimiento y en un formalismo de representación de sistema basado en reglas (SBR) se representará a través de una regla
La segunda frase corresponde a una afirmación o a un hecho y en un SBR podemos representarlo a través de un hecho.
A través de la programación orientada a objetos estas dos frases podrían trasladarse a un programa que permitiera inferir automáticamente que Sócrates es mortal. El mecanismo de inferencia que se utilizaría sería el de la herencia

**TRUE**

---

Para que una regla se active y pase a la agenda se deben de satisfacer todos los elementos condicionales del antecedente de la regla

**TRUE**

---

La orden (reset) carga los hechos de los constructores deffacts en la base de hechos y prepara al sistema para la ejecución
**TRUE**

---

En un ciclo de ejecución del motor de inferencia todas las reglas que estén activadas en la agenda se disparan y sus acciones son ejecutadas.

**FALSE**

---

En el siguiente programa de clips:
(deffacts hl
(hombre Socrates)

(mortal ?x)
(printout t ?x " es mortal" crlf )
(defrule r2
(hombre ?x)

(assert (mortal ?x))
El motor de inferencia ejecutaría primero la regla r 1 y después la regla r2

**FALSE**

---

En el siguiente programa de clips:
(deffacts hl
(hombre Socrates)
(defrule rl
(hombre ?x)
=>
(assert (mortal ?x))
El mecanismo de inferencia se denomina comparación de patrones. El ingeniero debe de programar este mecanismo para cada programa de clips que desee ejecutar.

**FALSE**

---

El código del siguiente programa es correcto:
(deffacts hl ;Constructor de hechos
(n 0) ; Hecho ordenado
(defrule r 1
?f<-(n ?x) ; Elemento condicional patrón (ECP)
; A la variable ?x se le ligará valores de los hechos que emparejen
• A la variable ?f se le liga la dirección de hecho con el que el ECP
(test (< ?x 10)); Elemento condicional test
(printout t "n= " ?y crlf) ;Acción de imprimir
(assert (n (+ ?x 1)); Afirmación de un hecho nuevo (n resultado-de-la-suma) (retract ?f)
Elimina el hecho cuya dirección está en la variable ?f

**FALSE**

---

La siguiente figura muestra un entorno de clips. El programa escrito permite al sistema inferir que Socrates es mortal.

**TRUE**

---

En el siguiente código
(deftemplate persona
(slot nombre)
(multislot apellidos)
(slot estudiantesdad)
)
(deffacts hechos
(persona (nombre Juan))
(persona (nombre Pedro))
)
(defrule r1
(persona (nombre ?x ))
=>
(printout t ?x crlf)
)

**FALSE**

---

En el siguiente programa la regla se activará sólo una vez
(defrule numeros
(numeros ?n)
(exists (num ?))
=>
(printout t "Números. " ?n crlf)
)
(deffacts hechos
(numeros 3)
(num 2)
(num 4)
(num 5)
)

**TRUE**

---

En el siguiente código
(deftemplate clase
(slot estudiantes)
(slot profesor)
)
( deffacts datos
(clase (estudiantes 30) (profesor "Marta Ramírez"))
(clase (profesor "Marta Ramírez")(estudiantes 30) )
)

se crea dos hechos definidos a partir de una plantilla pero presenta un error ya que era necesario especificar que el slot estudiantes es de tipo numérico y el slot profesor es de tipo cadena de caracteres

**FALSE**

---

En terminología de CLIPS, podemos decir que el siguiente hecho es ordenado a pesar de que sus elementos no lo están ni de menor a mayor ni de mayor a menor:
(vector 3 1 4 7 2)
**FALSE**

---

El elemento condicional and puede combinarse con el elemento condicional not pero no con el or, ya que produciría una situación inválida para el motor de inferencia

**FALSE**

---

La siguiente regla asigna un nuevo valor a una variable global x si encuentra un hecho de tipo valor con valor igual al actual de la variable x:
(defrule r-global-mal
(valor ?y&:(= ?y ?*x*) )
=>
(bind ?*x* 3) )
**TRUE**

---

El comando modify no permite modificar los valores del siguiente hecho:
(edad-cliente 25)

**TRUE**

---

Dada la siguiente regla:
(defrule comprobar-ventanas
(exists (ventana (estado abierta)) )
=>
(printout t "Aviso: cierre la ventana abierta" crlf) )
y los siguientes hechos afirmados:
(ventana (ubicacion S1) (estado abierta))
(ventana (ubicacion S2) (estado cerrada))
(ventana (ubicacion P1) (estado abierta))
podemos afirmar que la regla se activará sólo una vez.

**TRUE**

---

La siguiente regla muestra en el antecedente un elemento condicional and
(defrule ejemplo1-2
(datos-B (valor ~rojo&~verde)) =>
)

**FALSE**

---

Las siguientes reglas se activan para el mismo conjunto de hechos:
(defrule r-check01
(estado (j3 ?c31) ) (estado (j3 ?c32&:(= ?c31 ?c32)))
=> )
(defrule r-check02
(estado (j3 ?c31) ) (estado (j3 ?c32))
(test (= ?c31 ?c32))
=> )

**TRUE**

---

Mediante el comando deffacts de CLIPS podemos crear plantillas con campos univaluados (slot) y multivaluados (multislot).

**FALSE**

---

La siguiente regla:
(defrule r-check
(forall (sensor (id ?id) (tipo fuego))
(verificacion (id ?id) (estado OK)) )
=>
(assert (sensores verificados)) )
se activa para la siguiente base de afirmaciones:
(verificacion (id 5) (estado OK))
**TRUE**

---

Dada la siguiente regla:
(defrule cuatro-patas
(or (animal perro) (animal gato))
=>
(printout t "Animal de cuatro patas" crlf))
y los siguientes hechos:
(animal perro)
(animal gato)
podemos afirmar que la regla se activará sólo una vez

**FALSE**

---

Sea la siguiente regla
(defrule r-parar
(logical (semaforo rojo))
=>
(assert (parar)) )
y los siguientes hechos afirmados: (semaforo rojo)
Si después de dispararse la regla, la cual creará el hecho (parar), se elimina el hecho (semaforo rojo), automáticamente se borra el hecho (parar).

**TRUE**

---

La siguiente regla se activará una vez y proporcionará el 4 como el número mayor
(deffacts hechos (numero 2) (numero 3) (numero 4) )
(defrule numero-mayor
(numero ?x)
(not (numero ?y&:(> ?y ?x)))
=>
(printout t ?x " es el mayor número" crlf))

**TRUE**

---

En la siguiente plantilla, el campo x se inicializa automáticamente a un valor nulo si no se indica ninguno durante la afirmación del hecho:
(deftemplate dato
(slot x (default ?NONE)) )

**TRUE**

---

Aunque los módulos nos permiten agrupar hechos y reglas, la agenda en CLIPS se comparte entre todos los módulos existentes para evitar duplicidad de información
FALSE
Con la siguiente regla podemos encontrar los hechos que terminan con el número 23:
(defrule r-find-23
($?fact 23)
=>
(printout t "Encontrado el número 23 en " $?fact crlf) )
Por ejemplo,
(val 23)
(x 23)
(es-numero 23)

**FALSE**

---

En la sintaxis del Elemento Condicional test, <???????????> indica que ahí debe de ir otros Elementos Condicionales:
<EC-test> ::= (test <???????????>)

**FALSE**

El siguiente EC patrón es correcto e indica que emparejará cuando el color de ojos sea marrón o negro:
(ojos ?x&marron|negro)

**TRUE**

---

Los módulos en CLIPS nos permiten agrupar constructores de forma modular

TRUE
El siguiente código es incorrecto
(deftemplate persona
(slot nombre)
(multislot apellidos)
(slot edad)
)
(deffacts
(persona (nombre Juan))
(persona (nombre Pedro))
)

**TRUE**

---

En la siguiente plantilla que representa un equipo de fútbol, la longitud máxima del nombre de los jugadores de 15 caracteres:
(deftemplate equipo
(slot nombre (type STRING))
(multislot jugadores (type STRING) (cardinality 1 15)) )
**FALSE**

---

Sea la siguiente regla
(defrule r-pasar
(exists (semaforo verde))
=>
(assert (pasar)) )
y los siguiente hechos afirmados:
(semaforo verde)
Sí después de dispararse la regla, la cuál creará el hecho (pasar), se elimina el hecho (semaforo verde), automáticamente se borra el hecho (pasar).

**FALSE**

---

Las siguientes reglas se activan para el mismo conjunto de hechos:
(defrule r-check01
(estado (j3 ?c31) ) (estado
(j3 ?c32&:(= ?c31 ?c32)))
=> )
(defrule r-check02

(estado (j3 ?c32) ) (estado
(j3 ?c32))
(test (= ?c31 ?c32))
=> )

**TRUE**

---

Los modulos en CLIPS nos permiten agrupar constructores de forma modular
**True**

---

En la siguiente regla basta con que en la base de hechos estén los hechos (tengo mantequilla) o (tengo aceite)

(defrule posibles-tostadas
(tengo pan)
(or (tengo mantequilla)
(tengo aceite))
=>
(assert (desayuno tostadas))
)

**FALSO**

---

En el antecedente de una regla
**(los 8 tipos y hay diferentes… )**

---

La regla
(defrule para (logical (semáforo rojo)) => (assert (parar)))

**Que si existe el hecho (semáforo rojo) entonces se inserta el hecho (parar), y si se borra el hecho (parar) entonces se borra automáticamente (semáforo rojo)**

---

El elemento condicional
(test (= 2 2))

**Correcto, comprobando el valor devuelto por una función**

---