



Universidad de Granada

[decsai.ugr.es](http://decsai.ugr.es)

# Inteligencia Artificial

Seminario 1

## Agentes Conversacionales



DECSAI

**Departamento de Ciencias de la  
Computación e Inteligencia Artificial**

- 1. Método evaluación de las prácticas**
- 2. Introducción de Agentes Conversacionales**
- 3. Presentación de la práctica 1**
- 4. AIML**

# Evaluación de la parte de prácticas

## En Convocatoria Ordinaria

Asistencia y participación en las clases (0%)  
Calificación de varias prácticas/pruebas (100%):

Ahora NO se tiene en cuenta la asistencia para la nota!

Práctica 1	Agente conversacional	25%
Práctica 2	Resolución de problemas con agentes reactivos/deliberativos	25%
Práctica 3	Resolución de un juego basado en técnicas de búsqueda	25%
Examen de problemas		25%

No es obligatorio entregar todas las prácticas!!!

Es necesario alcanzar un 3 para hacer media con la teoría!

# Temporización Inicial Aproximada

Semana	Planificación de clases de Prácticas/Problemas		Comentarios
	Prácticas	Relaciones de Problemas	
01-mar	<b>Seminario 1 (AIML) / Práctica 1 (Agentes Conversacionales)</b>		Sin clase el lunes
08-mar		RP1	
15-mar	Seguimiento/dudas P1		
22-mar	Seguimiento/dudas P1		
29-mar	-----	-----	Semana Santa
05-abr	<b>Seminario 2 (Belkan) / Práctica 2 (Agentes Reactivos y Deliberativos)</b>		Sin clase el lunes
12-abr		RP2	
19-abr	Seguimiento/dudas P2		
26-abr	Seguimiento/dudas P2		
03-may	Seguimiento/dudas P2		
10-may	<b>Seminario3 / Práctica 3 (Juegos)</b>		
17-may		RP3	
24-may	Seguimiento/dudas P3		
31-may	Seguimiento/dudas P3		Sin clase jueves y viernes
07-jun	Seguimiento/dudas P3		Sin clase jueves y viernes

# Agentes Conversacionales

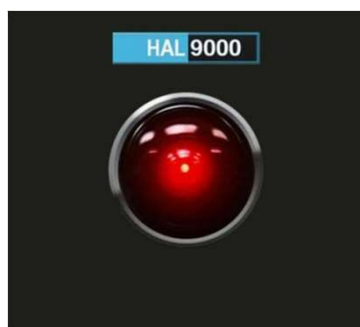
## Introducción



Alan **Turing** establece un **test** para determinar la inteligencia de un sistema artificial basado en la **capacidad de mantener la coherencia durante una conversación** con un ser humano.



- El cine ha ayudado a establecer esta vinculación entre I.A. y sistemas capaces de conversar con seres humanos.



2001 Una Odisea en el espacio  
(1969)



Star Wars  
(1977)

Ava



Ex Machina  
(2015)

En estos últimos años han surgido una gran cantidad de sistemas basados en **agentes conversacionales** con la intención de **facilitar o ayudar a los seres humanos a realizar algunas tareas.**

Son conocidos como **asistentes**.





# Práctica 1

Definir agentes  
conversacionales

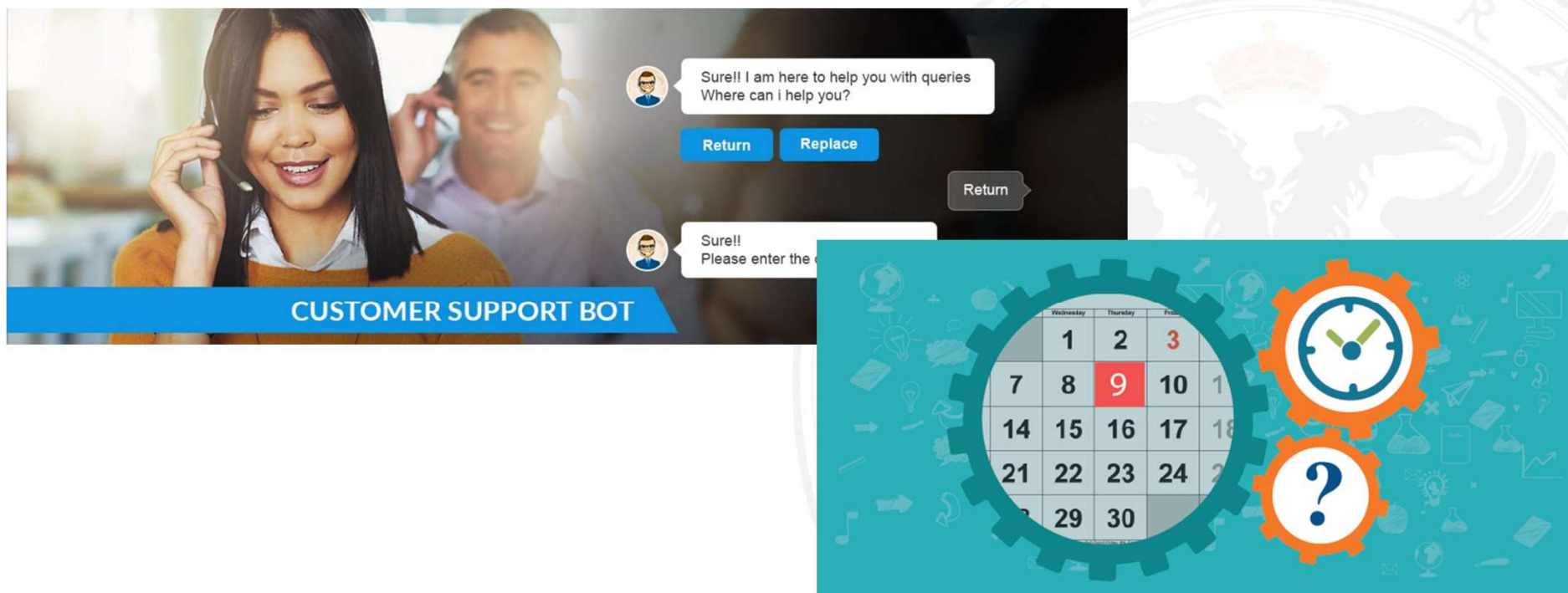




- **Familiarizarse con** una aplicación concreta de Inteligencia Artificial (IA), **los agentes conversacionales**,
- **aprender un lenguaje de representación de conocimiento diseñado específicamente para estas aplicaciones** (el lenguaje AIML 2.0),
- **aprender a usar un intérprete** (program-ab) **de este lenguaje**.

Se pide construir el conocimiento necesario en AIML 2.0 para:

1. Contestar algunas de las **preguntas básicas sobre información de fechas.**
2. Implementar un bot para la **gestión de citas en un servicio de atención al cliente.**



Sobre el primero de ellos, **se pide que el agente sea capaz de responder correctamente a las 5 preguntas que aparecen en el guion de prácticas**. Así, el asistente responderá como lo haría una persona a preguntas sobre información de fechas.

El objetivo de esta parte no es sólo que *el agente* responda como haríamos a las preguntas propuestas, sino que *ante pequeñas alteraciones en la forma de preguntar, sea capaz de reconocer qué es lo que debe responder*. **Es importante que el agente tenga esta capacidad, en otro caso la práctica no se considera válida.**

Las preguntas que aparecen en el guion son:

**P1: En qué estación estamos**

**P2: En qué fase del día estamos.**

**P3: Qué día de la semana es <hoy/manyana/pasado manyana>.**

**P4: Qué fecha será dentro de una semana.**

**P5: Qué fecha será el próximo [Lunes Martes ... Domingo]**

Para facilitar el uso de las operaciones sobre gestión de fechas, en el material de la práctica (dentro del directorio aiml del bot) se proporciona el fichero AIML ***dates\_es.aiml***.

El contenido de este fichero no está documentado, pero es fácil de interpretar una vez conocido el seminario y tutorial de AIML de las sesiones de prácticas.

El objetivo en esta parte es aproximarse a cómo se representa el conocimiento en AIML, mediante la escritura de las reglas necesarias, basándose en las reglas escritas en este fichero.

### Agente para la gestión de citas: Planteamiento del escenario

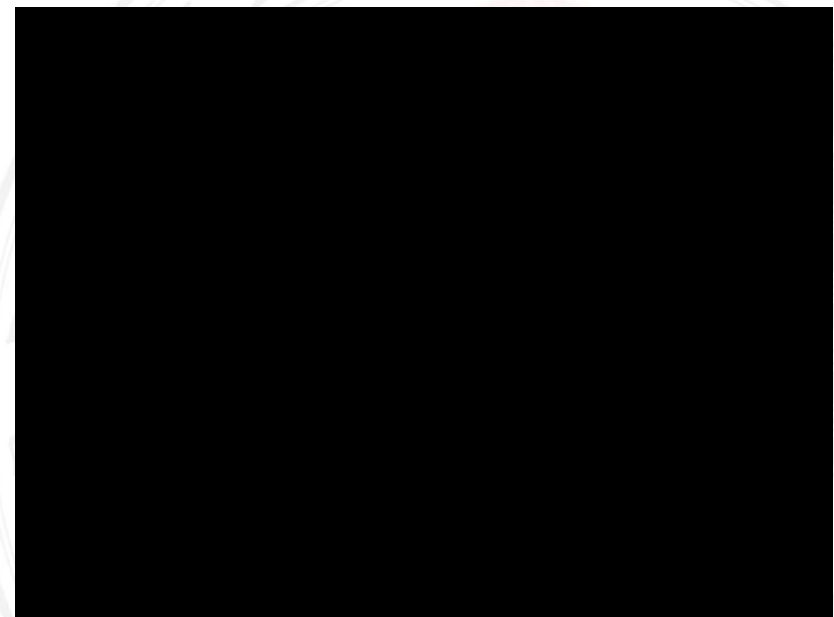
En este segundo escenario:

- **Un usuario contacta por teléfono con una clínica dental para concertar una cita.**
- **El asistente contesta y trata de ayudarle a determinar una cita (día y hora) consultando su base de datos de ocupación.**
- **El asistente informa de forma amigable sobre qué días tiene disponibles.**
- El usuario puede tener claro o no qué día quiere la cita y el bot tiene que actuar en consecuencia.

**Google Assistant Haircut Appointment  
Call | Google I/O 18**

<https://www.youtube.com/watch?v=yDI5oVn0RgM>

Otro ejemplo interesante: <https://aeon.co/videos/whats-it-like-to-chat-with-an-ai-that-mimics-you-uncanny-conversations-with-replika>



Vamos a asumir que el bot parte de una representación de una agenda de eventos como un diccionario AIML llamado **eventos.txt** en el que se guarda la información sobre la fecha, hora y nombre del evento (en general aparecerá como “NOLIBRE” o “LIBRE”). Por ejemplo, la siguiente tabla muestra un evento en el que el intervalo de las 06:00 a las 09:00 del día 23 de Febrero del 2021 está “Ocupado” (marcado como “NOLIBRE”).

[illegible]



Este fichero es un ejemplo de cómo representar un diccionario (*map*) en un fichero de texto según AIML

- **Clave:** la fecha en que se produce el evento (en formato *dd\_mm\_aa*)
- **Valor:** una lista de 24 posiciones que indican los intervalos de 1 hora de cada día.
  - Esta lista comienza en la hora 00:00, siguiendo por, 01:00, 02:00, etc.
  - Ejemplo: **la hora 17:00 corresponde a la posición 18** de esa lista.

**Es importante tener en cuenta que** el usuario habla castellano y tiene una representación de fechas habitual en España que sigue el formato *"dd de MMMMMMMMM del yy"*.

Finalmente, observar que la **representación de fechas almacenadas como claves en el diccionario es diferente de la usada en la conversación con el humano.**

## EVENTOS.TXT

[illegible]





- Aseguraos de que en la configuración local de vuestro sistema operativo el formato de fecha es el “español de España” (es\_ES).
- LINUX
  - Cambiar el formato fecha en el entorno local con \$LC\_TIME  
[https://www.linuxtotal.com.mx/index.php?cont=info\\_admon\\_007](https://www.linuxtotal.com.mx/index.php?cont=info_admon_007)
  - se trata de asignar un valor a esa variable, pero para eso hay que tener instalados o generados los ficheros de “ language settings”. Y esto se hace siguiendo estas instrucciones [https://www.thomas-krenn.com/en/wiki/Configure\\_Locales\\_in\\_Ubuntu](https://www.thomas-krenn.com/en/wiki/Configure_Locales_in_Ubuntu)
- WINDOWS
  - <https://www.windowscentral.com/how-change-date-and-time-formats-windows-10>

### EVENTOS.TXT

```

13_02_21:LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE NOLIBRE LIBRE LIBRE LIBRE NOLIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE
14_02_21:LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE
15_02_21:LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE
16_02_21:LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE
17_02_21:LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE
18_02_21:LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE
19_02_21:LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE NOLIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE
20_02_21:LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE
21_02_21:LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE
22_02_21:LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE
23_02_21:LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE NOLIBRE NOLIBRE NOLIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE LIBRE

```

Considerando esta información de partida:

Vamos a considerar 3 etapas en el desarrollo del agente conversacional:

**(Nivel 1) Operaciones de consulta de las citas en un día y determinar cita usando fechas descritas en un formato simple**

P.ej: Quisiera una cita para el 12 de Febrero del 21.

**(Nivel 2) Determinar cita usando una descripción de fechas más elaborada**

P.ej: Quisiera una cita para el próximo lunes por la tarde.

**(Nivel 3) Mejora de las respuestas proporcionadas por el agente conversacional**

P.ej: Ante la pregunta “Quiero cita para pasado mañana” el bot respondería:

POR LA MANYANA TENGO LIBRE DESDE LAS **NUEVE** HASTA LAS **ONCE**, Y POR LA TARDE TENGO LIBRE A LA **UNA**, DE LAS **TRES** A LAS **CINCO** Y DE LAS **SIETE** A LAS **OCHO**

### Ejemplo de conversación en Nivel 1

Humano: Quisiera una cita para el 20 de febrero del 21

Robot: Muy bien voy a comprobarlo, espere un momentico....

*Puede que ese día esté ocupado en cuyo caso el bot contestaría*

Lo siento no puedo está ocupado, ¿desea otro día?

H: Sí (o No)

R: *<el bot contesta dependiendo de la respuesta del cliente, preguntando por un nuevo día o terminando la conversación>*

[...]

*Puede que ese día queden algunos huecos libres, en cuyo caso el bot contestaría:*

*Si tiene huecos libres por la mañana y por la tarde*

R1: Pues por la mañana tengo los siguientes huecos libres 10:00 y 11:00 , y por la tarde 14:00 15:00 y 18:00

*Si tiene huecos solo por la mañana*

R2: Pues por la mañana tengo los siguientes huecos libres 10:00 y 11:00 y toda la tarde ocupada

*Si tiene huecos solo por la tarde*

R3: Pues la mañana la tengo ocupada y por la tarde los siguientes huecos libres 14:00 15:00 y 18:00

*Si tiene la mañana libre*

R4: Pues la mañana está libre y por la tarde ...

*Si tiene la tarde libre*

R5: *<respuesta análoga>*

*¿Quiere alguno en concreto?*

***En este contexto el usuario puede responder de distinta forma, bien determina la hora o le da igual.***

H: Sí a las 09:00

*<el bot contesta adecuadamente, si la hora no es correcta debe informar al usuario de que la hora es errónea y volver a preguntar si quiere alguna hora concreta>*

R: *la respuesta debe ser amigable, por ejemplo, "La hora es incorrecta, recuerda que <informar sobre huecos libres>"*

*< en otro caso incluye la cita en la agenda y finaliza la conversación o pide si el usuario quiere otra>*

H: Me da igual

R: *<en este caso el bot rellena el siguiente hueco libre e informa al humano de la decisión>*

### Tareas a realizar en el nivel 1

1. **Escribir las reglas necesarias para poder realizar las siguientes consultas:**
  - a. Resolución temporal del día. **Dado un día** representado como "dd de MMMMMMMMM del yy"
    - i. **informar sobre si el día está libre o no** (un día está libre si tiene al menos una franja horaria no ocupada, es decir, con la palabra "LIBRE" entre las 08:00 y las 20:00, en otro caso está ocupado),
    - ii. **devolver la lista de franjas horarias libres en un día**, es decir, una secuencia de números representando la posición de las horas libres que hay entre dos horas de un mismo día. Por ejemplo 11 12 15 16 19 representaría que hay horas libres en ese día a las 10:00 11:00 14:00 15:00 y 18:00,
    - iii. **devolver una lista de franjas libres solo por la mañana** (las horas de la mañana son de 08:00 a 12:00 ambas inclusive),
    - iv. **devolver una lista de franjas libres solo por la tarde** (las horas de la tarde son desde las 13:00 hasta las 20:00 ambas inclusive).
2. **Escribir las reglas necesarias para que el bot pueda entablar una conversación con un usuario que** desea agendar una cita en una fecha concreta, especificada en un formato sencillo, en un contexto en el que el cliente **tiene claro qué día quiere la cita**.

### Tareas a realizar en el nivel 1: funcionalidades mínimas exigidas

**Funcionalidad mínima.** Para poder superar este nivel, al menos debe implementarse una regla para cada una de las siguientes funciones.

**Informar del estado de un día**

Entrada: un día representado como dd de MMMMMMMMM del yy

Salida: "SI" si el día está libre, "NO" si no lo está.

Ejemplo:

Comando en la terminal del bot LIBRE 07 de JULIO del 21

Salida en la terminal: SI (o NO)

**Informar de las franjas libres en un día**

Entrada: un día representado como dd de MMMMMMMMM del yy

Salida: una lista de horas en formato HH:MM separadas por espacios. O la cadena "**EMPTYLIST**" en caso de que la lista esté vacía

Ejemplo:

Comando: HORASLIBRES 07 de JULIO del 21

Salida: 09:00 14:00 15:00 18:00.

**Informar de las franjas libres por la mañana en un día**

Entrada: un día representado como dd de MMMMMMMMM del yy

Salida: una lista de horas en formato HH:MM separadas por espacios. O la cadena "**EMPTYLIST**" en caso de que la lista esté vacía

Ejemplo:

Comando: HLMANYANA 07 de JULIO del 21

Salida: 08:00 11:00

**Informar de las franjas libres por la tarde en un día**

Entrada: un día representado como dd de MMMMMMMMM del yy

Salida: una lista de horas en formato HH:MM separadas por espacios. O la cadena "**EMPTYLIST**" en caso de que la lista esté vacía

Ejemplo:

Comando: HLTARDE 07 de JULIO del 21

Salida: 14:00 15:00 18:00.

## Material proporcionado para facilitar el trabajo

- ***dates\_ES.aiml***: *operaciones básicas para la gestión de fechas*. El contenido de este fichero no está documentado, por lo que se recomienda entender el contenido de las operaciones implementadas en él mediante experimentación. En resumen, *el alumno probará el funcionamiento de las funciones en la terminal*.
- ***utilidades.aiml***: que consiste en una *librería de utilidades que pueden facilitar la implementación de la prácticas*. Las operaciones incluidas en este fichero están documentadas en el documento **Anexo3\_UtilidadesAIML.pdf** proporcionado en el material de esta práctica.
- ***utilidades\_2020.aiml***: que consiste en una *librería de utilidades específicas para esta práctica*, construidas sobre *Utilidades.aiml* y *destinadas a simplificar el trabajo del alumno*. Las operaciones incluidas en este fichero están documentadas en el documento **Anexo4\_UtilidadesAIML\_2021.pdf** proporcionado en el material de esta práctica.

### Ejemplo de conversación en el Nivel 2

H: Quisiera una cita para **pasado mañana**.

H: Quisiera una cita para **el próximo lunes por la tarde**.

H: Quiero una cita para **mañana por la mañana**.

*La dinámica de la conversación es similar al nivel 1, la funcionalidad está centrada en poder interpretar una descripción de fecha más elaborada.*

La siguiente gramática describe todas las posibles sentencias: **la calificación máxima se obtiene si se pueden interpretar todas las descripciones que genera la gramática**, pero no es obligatorio interpretarlas todas. **Un repertorio mínimo de 7 posibles descripciones se considera suficiente.**

```
<DESCRIPCION FECHA>::= <ESPECIFICA DIA> <COMPLEMENTO>
<ESPECIFICA DIA>::= <FECHA CONCRETA> | HOY | MAÑANA | PASADO MAÑANA | PROXIMO <DIA SEMANA> |
SIGUIENTE <DIA SEMANA>
<FECHA CONCRETA>::= 13 de Febrero de 2020.
<DIA SEMANA>::= LUNES | MARTES | ... | VIERNES
<COMPLEMENTO>::= POR LA <FIN COMPLEMENTO> |
                   <FIN COMPLEMENTO> | A LAS <HORA>
<FIN COMPLEMENTO>::= MAÑANA | TARDE
<HORA>::= HH:00
```



## Tareas a realizar en el nivel 2

**Funcionalidad mínima.** Para poder superar este nivel, al menos debe implementarse una regla para cada una de las siguientes funciones.

**Informar del estado de un día especificado con <ESPECIFICA DIA>**

Entrada: un día representado como <ESPECIFICA DIA>

Salida: "SI" si el día esta libre, "NO" si no lo está.

Ejemplo:

Comando en la terminal del bot LIBRE PROXIMO LUNES

Salida en la terminal: SI (o NO)

**Informar de las franjas libres en un día especificado con <ESPECIFICA DIA>**

Entrada: un día representado como <ESPECIFICA DIA>

Salida: una lista de horas en formato HH:MM separadas por espacios. O un valor apropiado en caso de que la lista esté vacía

Ejemplo:

Comando: HORASLIBRES PASADO MANYANA

Salida: 09:00 14:00 15:00 18:00.

**Informar de las franjas libres por la mañana en un día especificado con <ESPECIFICA DIA>**

Entrada: un día representado como <ESPECIFICA DIA> POR LA MANYANA

Salida: una lista de horas en formato HH:MM separadas por espacios. O un valor apropiado en caso de que la lista esté vacía

Ejemplo:

Comando: HORASLIBRES MANYANA POR LA MANYANA

Salida: 08:00 11:00

**Informar de las franjas libres por la tarde en un día especificado con <ESPECIFICA DIA>**

Entrada: un día representado como <ESPECIFICA DIA> POR LA TARDE

Salida: una lista de horas en formato HH:MM separadas por espacios. O un valor apropiado en caso de que la lista esté vacía

Ejemplo:

Comando: HORASLIBRES HOY POR LA TARDE

Salida: 13:00 18:00

**Informar de las franjas libres a partir de una hora dada en un día especificado con <ESPECIFICA DIA>**

Entrada: un día representado como <ESPECIFICA DIA> A PARTIR DE LAS <HH:MM>

Salida: una lista de horas en formato HH:MM separadas por espacios. O un valor apropiado en caso de que la lista esté vacía

Ejemplo:

Comando: HORASLIBRES ELPROXIMO JUEVES A PARTIR DE LAS 14:00

Salida: 13:00 18:00

### Tareas a realizar en el Nivel 3:

**Adaptar la respuesta al usuario de una forma amigable, extendiendo y mejorando las respuestas que se han proporcionado en los niveles anteriores**

Por ejemplo, si consideramos la siguiente lista de horas  
09:00 10:00 11:00 13:00 15:00 16:00 17:00 19:00 20:00

Observamos que tiene un intervalo de horas consecutivas de [09:00 a 11:00], un intervalo de un único valor a las [13:00], un intervalo de [15:00 a 17:00] y otro de [19:00 a 20:00]).

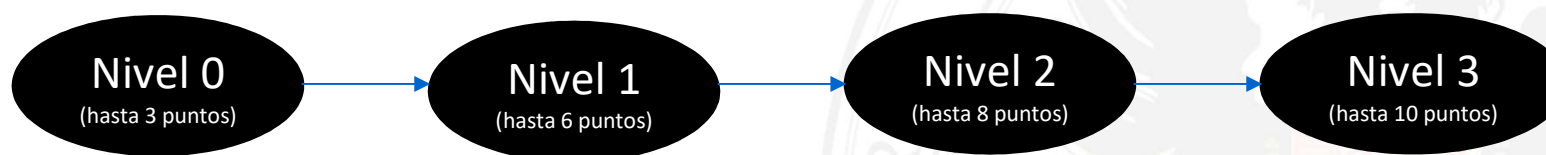
El bot debería responder POR LA MANYANA TENGO LIBRE DESDE LAS **NUEVE** HASTA LAS **ONCE**, Y POR LA TARDE TENGO LIBRE A LA **UNA**, DE LAS **TRES** A LAS **CINCO** Y DE LAS **SIETE** A LAS **OCHO**.

1. Determinar qué intervalos de horas consecutivas hay en la lista
2. Traducir cada hora a su correspondiente adjetivo numeral cardinal.
3. Contemplar la posibilidad de que cuando el usuario nos indique la hora, pueda responder:
  1. A las 09:00
  2. A las NUEVE DE LA MANYANA
  3. A las OCHO DE LA TARDE

## EVALUACIÓN

Hemos diseñado un modelo de evaluación que se adapta al nivel de implicación con el que el alumno quiere enfrentarse a esta práctica. **Cada nivel aumenta el grado de dificultad del anterior, y también la calificación que se puede obtener.**

**El proceso de evaluación estará basado en un formulario de autoevaluación** que estará disponible después de entregar la práctica, que deberá rellenar y enviar el alumno. Este formulario de autoevaluación juega el papel del proceso de defensa de la práctica, sin perjuicio de que el profesor pueda requerir una cita online con algún alumno en concreto.



**NIVEL 0 (hasta 3 puntos):** Es el mínimo que hay que entregar para que la práctica sea considerada para su evaluación.

- Lo evalúa el profesor, es necesario superar este nivel para pasar el siguiente. Los criterios se especifican en el guion.
- **El agente debe ser capaz de responder a las preguntas básicas sobre información de fechas.**

**P: En qué estación estamos.**

R: estamos en invierno/primavera/otoño/verano

**P: En qué fase del día estamos.**

R: Ahora estamos en la <manyana/tarde/noche>

**P: Qué día de la semana es <hoy/manyana/pasado manyana>.** Observar que el intérprete de AIML no acepta la “ñ”, adoptamos el convenio de sustituirla por “ny” en toda la práctica.

R: <hoy/manyana/pasado manyana> es [Lunes ... Domingo]

**P: Qué fecha será dentro de una semana.**

R: Dentro de una semana será 23 de Febrero del 2021

**P: Qué fecha será el próximo [Lunes Martes ... Domingo]**

R: El próximo [Lunes Martes ... Domingo] será 23 de [Enero ... Diciembre] del 2021.

**NIVEL 1 (hasta 6 puntos):** Consiste en hacer el Nivel 0 y las **consultas sobre disponibilidad de día y la conversación para agendar una fecha simple**.

**El profesor aportará un fichero de eventos predefinido** que servirá como base de datos inicial para las operaciones en este nivel y en los siguientes. **En el formulario de autoevaluación se pedirá al estudiante que haga uso del bot ante situaciones concretas** para comprobar que se realizan correctamente las operaciones de consulta y modificación de fechas que se han implementado, así como exponer al bot en una conversación básica en los términos descritos para el Nivel 1.

**El profesor comprobará si el contenido del cuestionario de autoevaluación coincide con las respuestas reales del bot, caso de no ser así la práctica se considerará directamente suspensa.**

Si realiza correctamente las tareas encomendadas, el alumno obtendrá una calificación de 6 puntos y puede pasar al siguiente nivel.

En otro caso, la calificación del alumno será de 3 puntos y el proceso de evaluación se dará por terminado.

**NIVEL 2 (hasta 8 puntos):** Consiste en hacer el Nivel 1 y la parte de **interpretación de descripciones de fechas complejas, según la gramática descrita** anteriormente.

El cuestionario incluirá consultas para comprobar **qué posibilidades de descripción de fecha ha implementado el alumno**, comprobando el grado de completitud de implementación de la gramática si fuera necesario mediante preguntas directas al alumno. En todo caso, durante la conversación con el bot, el agente deberá responder con coherencia. Si ese es el caso se añade:

- **UN** punto si se han implementado todas las funcionalidades descritas para el Nivel 2.
- **Una puntuación entre 0 y 1** dependiendo del grado de completitud de la implementación de la gramática.

**Si responde correctamente y con coherencia a las consultas, ha implementado todas las funcionalidades y ha implementado la gramática en su totalidad, obtendrá un 8 en la calificación y puede pasar al nivel 3.**

En otro caso, la calificación del alumno será de 6 puntos y el proceso de evaluación se dará por terminado.



**NIVEL 3 (hasta 10 puntos):** Consiste en **hacer el Nivel 2 y la extensión para mejorar la respuesta a las preguntas**, además de mantener una conversación en términos similares a los Niveles 1 y 2.

Se pedirá en el cuestionario de autoevaluación que se haga uso del bot para comprobar que se han realizado correctamente las tareas del Nivel3 y que se sigue una conversación coherente.

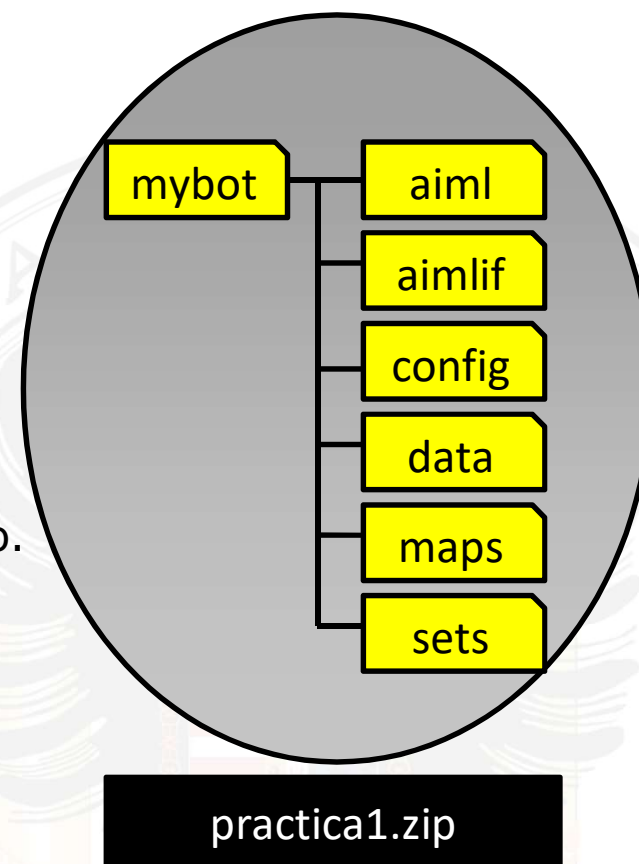
Si el funcionamiento es correcto, la calificación máxima en este nivel es de 2 puntos que se añadirán a la obtenida hasta el Nivel 2.



## ¿Qué hay que entregar?

Se ha de entregar un archivo comprimido zip llamado "**practica1.zip**" que contiene la estructura en directorios tomando la carpeta "**mybot**" como la raíz, donde en la carpeta

1. "aiml" se encuentran los archivos **aiml** que describen el conocimiento del agente conversacional hasta el nivel que ha deseado realizar,
2. "sets" los **sets** que necesita para que funcione correctamente su agente,
3. "maps" los **maps** que necesita para que funciones correctamente su agente,
4. "aimlif" los archivos **csv** que se hayan generado.

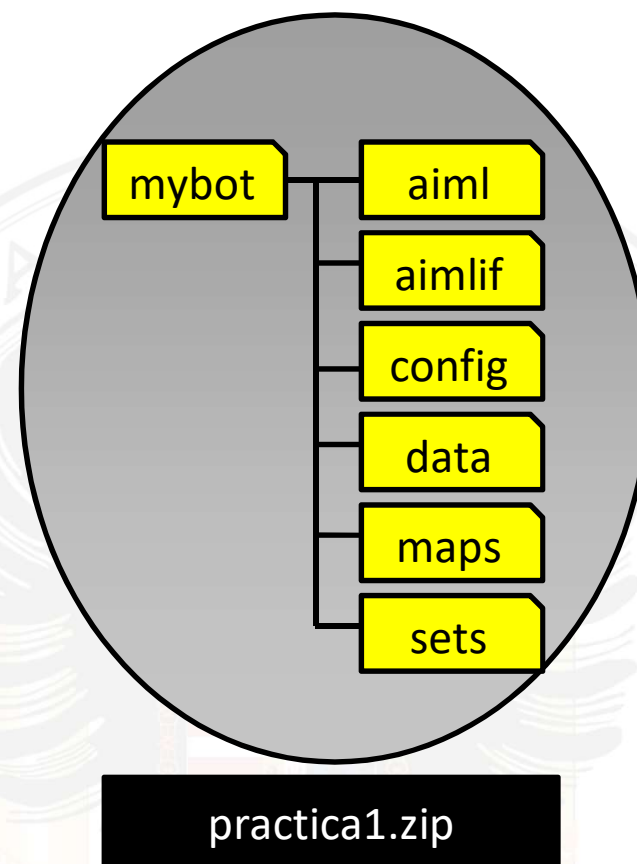


## ¿Qué hay que entregar?

IMPORTANTE: dentro del directorio aiml tienen que aparecer los siguientes archivos:

- **Nivel0.aiml**: para las reglas del Nivel 0.
- **Nivel1.aiml**: para las de Nivel 1.
- **Nivel2.aiml**: para las de Nivel 2.
- **Nivel3.aiml**: para las de Nivel 3.

Nota: es muy importante repetir que se entrega el contenido COMPLETO del directorio “mybot”, pero en ningún caso el directorio del “program-ab” completo, si una entrega no se ajusta a este requisito se penalizará correspondientemente.



## Consideraciones Finales

1. Las prácticas son **INDIVIDUALES**.
2. Entre la entrega y la defensa, se procederá a pasar un **detector de copias** a los ficheros entregados por los alumnos de todos los grupos de prácticas independientemente del grupo de teoría al que pertenezca.
3. Los alumnos asociados con **las prácticas que se detecten copiadas**, ya sea durante el proceso de detección de copia o durante la defensa de la práctica, **tendrán automáticamente suspensa la asignatura** y deberán presentarse a la convocatoria extraordinaria.
4. **Tiene la misma penalización el que copia como el que se deja copiar.** Por eso razón, para prevenir que sucedan estas situaciones, os aconsejamos que en ningún caso paséis vuestro código a nadie y bajo ninguna circunstancia.
5. El objetivo de la defensa es evaluar lo que vosotros habéis hecho, por consiguiente, quién asiste tiene que saber justificar cada cosa de la que aparece en su código. La justificación no apropiada de algún aspecto del código implica considerar la práctica copiada. **CONSEJO: no metáis nada en vuestro código que no sepáis explicar.**
6. El alumno que entrega una práctica y **no se presenta al proceso de defensa** tiene una **calificación de cero en la práctica**

## Desarrollo Temporal

- **Semana del 1 de Marzo**
  - Presentación de la práctica
- **Clases del 8 de Marzo al 26 de Marzo**
  - Desarrollo de la práctica en clase
- **Entrega de la práctica:** 5 de Abril, antes de las 23:00
- **Entrega del formulario de autoevaluación:** del 6 al 8 de Abril hasta las 23:00

## El lenguaje AIML

1. Estructura básica de AIML
2. El intérprete "program-ab"
3. Wildcards o "comodines"
4. Variables
5. Reducción Simbólica (<srai>)
6. Sets y Maps
7. Contexto
8. Random, Estructura Condicional y Ciclos
9. Aprender

## El lenguaje AIML

Estructura  
Básica de  
AIML



AIML (Artificial Intelligence Markup Language)<sup>1</sup> es un **lenguaje basado en XML** para crear aplicaciones de inteligencia artificial orientado al **desarrollo de interfaces que simulan el comportamiento humano**, manteniendo una implementación mediante programas simples, siendo fácil de entender y de mantener.

- ¿Por qué usar AIML?
  - Es un lenguaje simple
  - Es "Open Source"
  - Se pueden encontrar una amplia diversidad de intérpretes para este lenguaje.

### Estructura básica:

```
<?xml version="1.0" encoding="UTF-8"?>
<aiml version="2.0">

<category>
<pattern>Esta es la pregunta</pattern>
<template>Esta es la respuesta</template>
</category>

</aiml>
```

<sup>1</sup> Inicialmente liberado en 2001. Última versión estable: 2018 <http://www.aiml.foundation/doc.html>  
 AIML formaba la base de A.L.I.C.E., sistema ganador del premio Loebner en 2000, 2001 y 2004.



## El lenguaje AIML

El Interprete  
“program-ab”

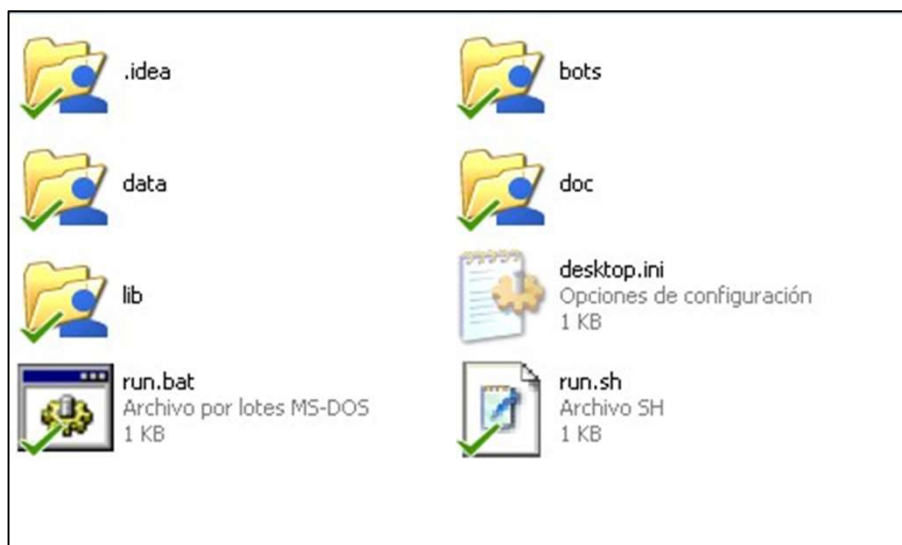


Program-ab es un intérprete para AIML 2.0 que es también "Open Source". Será el que usaremos en la práctica.

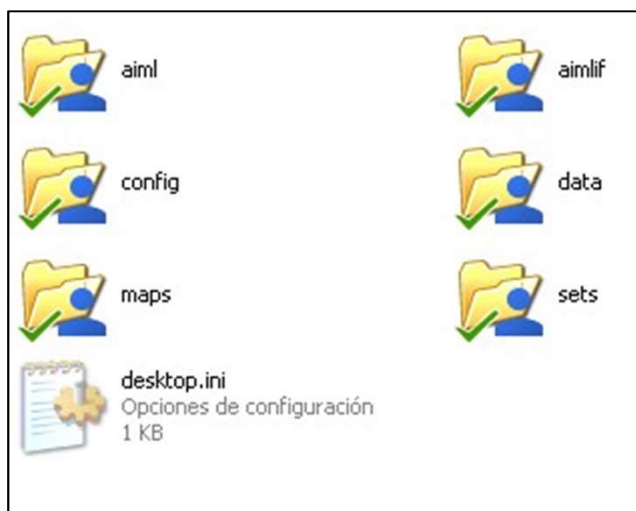
Es simple de usar, aquí aprenderemos a instalarlo.

1. Descargamos el archivo "program-ab.zip"
2. Se descomprime en una carpeta.

3. Accedemos a la carpeta "bots".
4. Accedemos a la carpeta "mybot".



Aquí aparece la estructura de un "bot"



- **aiml** carpeta donde se incluyen los archivos con extensión aiml.
- **aimlif** carpeta donde se creará la nueva información aprendida por el "bot". En este caso, la extensión de los ficheros es .csv
- **config** carpeta que contiene la información del "bot"
- **data** carpeta donde se almacena información temporal del intérprete. En nuestro caso, esta carpeta será ignorada.
- **sets** carpeta donde se almacenan los "set" que va a usar el intérprete.
- **maps** carpeta donde se almacenan los "map" que usará el intérprete.

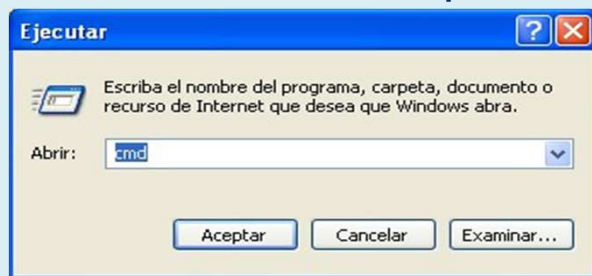
5. Accedemos a la carpeta ***aiml***
6. Creamos un fichero que llamaremos ***"primero.aiml"***.
7. Como editor podemos usar el *CodeBlocks* en *Windows* o el editor *atom* en *linux*, pero podría ser cualquier otro editor de texto.
8. Copiamos en el editor el siguiente texto

```
<?xml version="1.0"
  encoding="UTF-8"?>
<aiml version="2.0">

<!-- Primera regla -->
<category>
<pattern>Hola</pattern>
<template>Hola, que
  tal?</template>
</category>

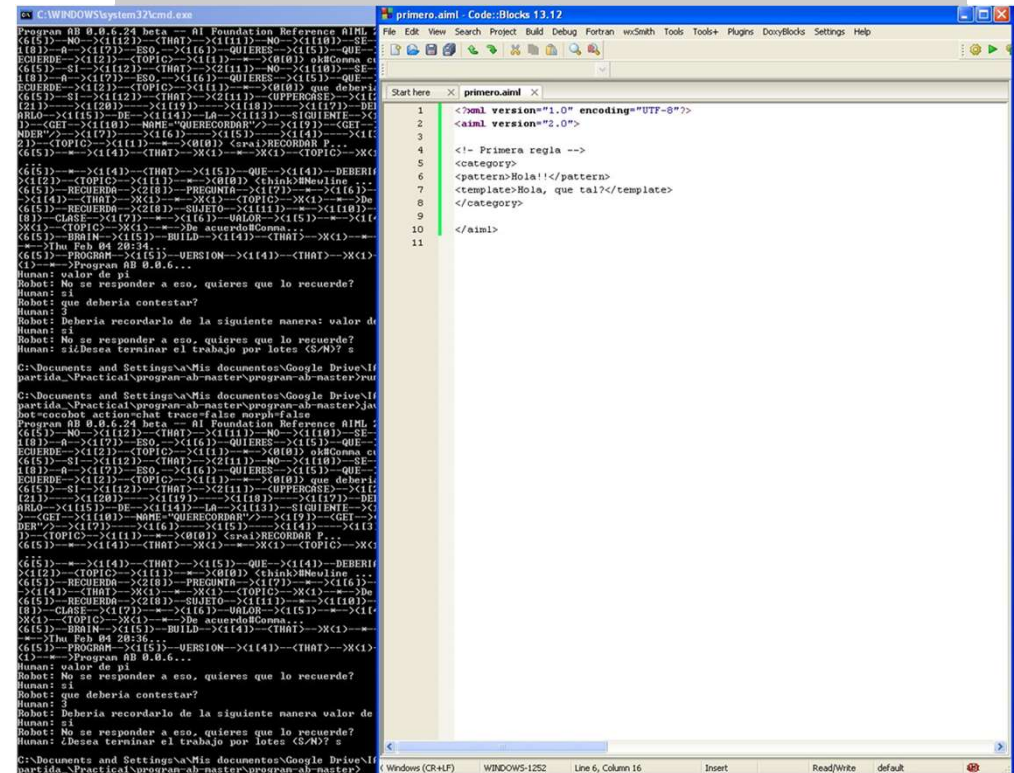
</aiml>
```

9. Ya que el editor y el intérprete no están integrados, haremos lo siguiente: Pulsaremos en "inicio -> ejecutar", ponemos "cmd" y le daremos a "Aceptar"



10. En el terminal que nos aparece nos movemos hasta el directorio raíz de **program-ab**, donde está el archivo "**run.bat**" (**windows**) o "**source run.sh**" (**linux**).

11. Y estructuramos la pantalla de la siguiente forma:



12. En la terminal ponemos "**run**" y pulsamos "**return**".

Nota: debéis tener instalado Java! <https://www.java.com/en/download/>



13. **Program-ab** cargará todos los archivos con extensión **aiml** que encuentre en la carpeta **aiml**. Cuando termine aparecerá "Human:" que es donde nosotros introducimos nuestra parte del diálogo con el bot. En esta caso pondremos "Hola!!", y en "Robot:" nos contestara "Hola, que tal?".

```
C:\Documents and Settings\A\Mis documentos\Google Drive\IA 2015-2016 Carpeta com
partida_\Practical\program-ab-master\program-ab>run

C:\Documents and Settings\A\Mis documentos\Google Drive\IA 2015-2016 Carpeta com
partida_\Practical\program-ab-master\program-ab>java -cp lib/Ab.jar Main bot=myb
ot action=chat trace=false morph=false
Program AB 0.0.6.24 beta -- AI Foundation Reference AIML 2.1 implementation
(3[5]--HOLA-->(1[4]--<THAT>-->X(1)--*-->X(1)--<TOPIC>-->X(1)--*-->Hola#Comma q
ue t...
(3[5]--BRAIN-->(1[5]--BUILD-->(1[4]--<THAT>-->X(1)--*-->X(1)--<TOPIC>-->X(1)-
*-->Mon Feb 08 11:43...
(3[5]--PROGRAM-->(1[5]--VERSION-->(1[4]--<THAT>-->X(1)--*-->X(1)--<TOPIC>-->X
(1)--*-->Program AB 0.0.6...
Human: Hola!!
Robot: Hola, que tal?
Human:
```

Antes de que **program-ab** haga una llamada al conocimiento contenido en los archivos **aiml**, realiza un **preprocesamiento** consistente en lo siguiente:

- *Eliminación de los signos de puntuación*, interrogación, admiración,...
- *Transformación a mayúsculas* de todo el contenido.
- *Extiende las contracciones* (esto heredado del proceso del inglés)

Así, para nuestra regla las siguientes entradas son equivalentes:

- Hola!!
- Hola
- hola!
- !!HOla!!
- Hola

**IMPORTANTE:** esta versión de AIML no reconoce las tildes ni la "ñ". Así, que no usaremos estos símbolos para definir las reglas!!



¿Qué ocurre si respondemos "Hola, que tal?" del robot con "Estoy bien".

```
C:\Documents and Settings\A\Mis documentos\Google Drive\IA 2015-2016 Carpeta com
partida_\Practical\program-ab-master\program-ab>run

C:\Documents and Settings\A\Mis documentos\Google Drive\IA 2015-2016 Carpeta com
partida_\Practical\program-ab-master\program-ab>java -cp lib/Ab.jar Main bot=myb
ot action=chat trace=false morph=false
Program AB 0.0.6.24 beta -- AI Foundation Reference AIML 2.1 implementation
<3[51]--HOLA--><1[41]--<THAT>-->X<1>--*-->X<1>--<TOPIC>-->X<1>--*-->Hola#Comma q
ue t...
<3[51]--BRAIN--><1[51]--BUILD--><1[41]--<THAT>-->X<1>--*-->X<1>--<TOPIC>-->X<1>--
*-->Mon Feb 08 11:43...
<3[51]--PROGRAM--><1[51]--VERSION--><1[41]--<THAT>-->X<1>--*-->X<1>--<TOPIC>-->X
<1>--*-->Program AB 0.0.6...
Human: Hola!!
Robot: Hola, que tal?
Human: Estoy bien
Robot: I have no answer for that.
Human: _
```

Como vemos, responde "I have no answer for that", es decir, "No tengo respuesta para eso".

El lenguaje tiene definida una regla por defecto (UDC) que ***si la entrada no se adapta con ningún <pattern>, devuelve esa respuesta.***

**¿Cómo funciona el proceso de encontrar la regla (*<category>* en aiml) que se dispara ante una entrada?**

El proceso consiste en tomar la entrada proporcionada por el usuario y **buscar las reglas cuyo patrón tiene adaptación sobre esa entrada.**

- **Si hay una única regla con adaptación, esa es la que se dispara.**
- **Si hay más de una regla**, se dispara **la de mayor prioridad** (más adelante hablaremos sobre esto).
- **Si no hay ninguna regla** con adaptación a la entrada **se disparará la regla por defecto**: "I have no answer for that".

**Ejercicio 1: Añade en primero.aiml las reglas necesarias para mantener la siguiente conversación con el bot.**

Human: Hola!

Robot: Hola, que tal?

Human: Yo bien, que tal tu?

Robot: Estoy genial!!! Me encanta conocer gente nueva.

Human: Genial!! Como te llamas?

Robot: Mi nombre es HALfonso

### Resolución Ejercicio 1:

Human: Hola

Robot: **Hola, que tal?**

Human: Yo bien, que tal tu?

Robot: **Estoy genial!!!**

Human: Como te llamas?

Robot: **Mi nombre es HALfonso**



```
<!-- Primera regla -->
<category>
<pattern>Hola</pattern>
<template>Hola, que
    tal?</template>
</category>
```



### Resolución Ejercicio 1:

Human: Hola

Robot: **Hola, que tal?**

Human: Yo bien, que tal tu?

Robot: **Estoy genial!!!**

Human: Como te llamas?

Robot: **Mi nombre es HALfonso**



```
<!-- Primera regla -->
<category>
<pattern>Hola</pattern>
<template>Hola, que tal?</template>
</category>

<!-- Segunda regla -->
<category>
<pattern>yo bien, que tal
tu</pattern>
<template>Estoy
genial!!!</template>
</category>
```

### Resolución Ejercicio 1:

Human: Hola

Robot: **Hola, que tal?**

Human: Yo bien, que tal tu?

Robot: **Estoy genial!!!**

Human: Como te llamas?

Robot: **Mi nombre es HALfonso**

```
<!-- Primera regla -->
<category>
<pattern>Hola</pattern>
<template>Hola, que tal?</template>
</category>

<!-- Segunda regla -->
<category>
<pattern>yo bien, que tal
tu</pattern>
<template>Estoy genial!!!</template>
</category>

<!-- Tercera regla -->
<category>
<pattern>como te llamas</pattern>
<template>
    Mi nombre es HALfonso
</template>
</category>
```

```

primero.aiml - Code::Blocks 13.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Pl
...
Start here X primero.aiml X primero.aiml X
1  <?xml version="1.0" encoding="UTF-8"?>
2  <aiml version="2.0">
3
4
5  <!-- regla 1 -->
6  <category>
7  <pattern>Hola</pattern>
8  <template>Hola, que tal?</template>
9  </category>
10
11 <!-- regla 2 -->
12 <category>
13 <pattern>yo bien, que tal tu</pattern>
14 <template>Estoy genial!!!</template>
15 </category>
16
17 <!-- regla 3 -->
18 <category>
19 <pattern> Como te llamas</pattern>
20 <template> Mi nombre es H&Lfonso</template>
21 </category>
22
23
24 </aiml>
25

```



## El lenguaje AIML

Wildcards  
“comodines”



Los wildcards o "comodines" permiten capturar varias entradas para una misma regla (o categoría).

Hay varios comodines:

- El comodín "\*": captura **una o más** palabras de la entrada

**<pattern>Hola \*</pattern>**

Captura entradas como:

Hola amigo

Hola, estoy aquí de nuevo

Hola Arturo

- El comodín "^": captura **cero o más** palabras de la entrada

**<pattern>Hola ^</pattern>**

Captura entradas como:

Hola

Hola, estoy aquí de nuevo

Hola Arturo

¿Qué ocurre si existen los dos siguientes patrones?

```
<pattern>Hola *</pattern>
```

```
<pattern>Hola ^</pattern>
```

**El patrón que contiene “^” tiene mayor prioridad** y por consiguiente, será esta la regla que se dispare.

En este caso concreto, **el patrón con el “\*” no se disparará nunca si existe el otro patrón.**

¿y qué ocurre si aparecen los siguientes 3 patrones

```
<pattern>Hola *</pattern>
```

```
<pattern>Hola ^</pattern>
```

```
<pattern>Hola  
amigo</pattern>
```

ante la entrada “**Hola amigo**”?

En este caso, **la adaptación exacta tiene mayor prioridad** que “^” y ésta, a su vez, mayor prioridad que “\*”.

Hay otros dos "comodines"

- El comodín "\_": captura **una o más** palabras de la entrada (como el \*)

**<pattern>Hola \_</pattern>**

- El comodín "#": captura **cero o más** palabras de la entrada (como el ^)

**<pattern>Hola #</pattern>**

**La única diferencia** con los anteriores **es la prioridad**, así el orden de prioridad de mayor a menor es el siguiente:

**Hola # > Hola \_ > Hola amigo > Hola ^ > Hola \***

A veces se desea definir un patrón que tenga mayor prioridad que “#” o “\_”. Para esos casos está el **símbolo “\$”, que indica que ese patrón tiene la mayor prioridad si la adaptación contiene esa palabra.**

```
<pattern>$Quien * Luis</pattern>
```

```
<pattern>_ Luis </pattern>
```

En este ejemplo, si en la entrada aparece “Quien”, el primer patrón tiene prioridad sobre el segundo.

**\$ no es un comodín, es sólo un marcador de prioridad.**

Los comodines pueden ser capturados dentro del **<template>** usando **<star/>**.

```
<category>
<pattern>Mi nombre es *</pattern>
<template>Hola <star/></template>
</category>
```

Human: Mi nombre es Rocio

Robot: **Hola Rocio**

Cuando hay más de un comodín se hace uso de **<star index="x"/>**, donde x indica la posición que ocupa el comodín desde el principio del patrón.

```
<category>
<pattern>Estudio * en *</pattern>
<template>En <star index="2"/>, yo también estudio
<star/></template>
</category>
```

Human: Estudio informatica en Granada

Robot: **En Granada, yo también estudio informatica**

**Ejercicio 2: Modifica o añade reglas a las reglas del ejercicio 1 para que incluyan comodines y pueda seguir la siguiente conversación, donde “...” representa que al menos aparece una palabra más en la entrada, y “..1..” representa que los comodines están vinculados.**

Human: Hola ...

Robot: Hola, que tal?

Human: Hola

Robot: Hola de nuevo, que tal?

Human: ... que tal tu?

Robot: Estoy genial!!!

Human: Fenomeno, me llamo ..1..

Robot: Que casualidad ..1.. yo tambien tengo nombre, me llamo HALberto

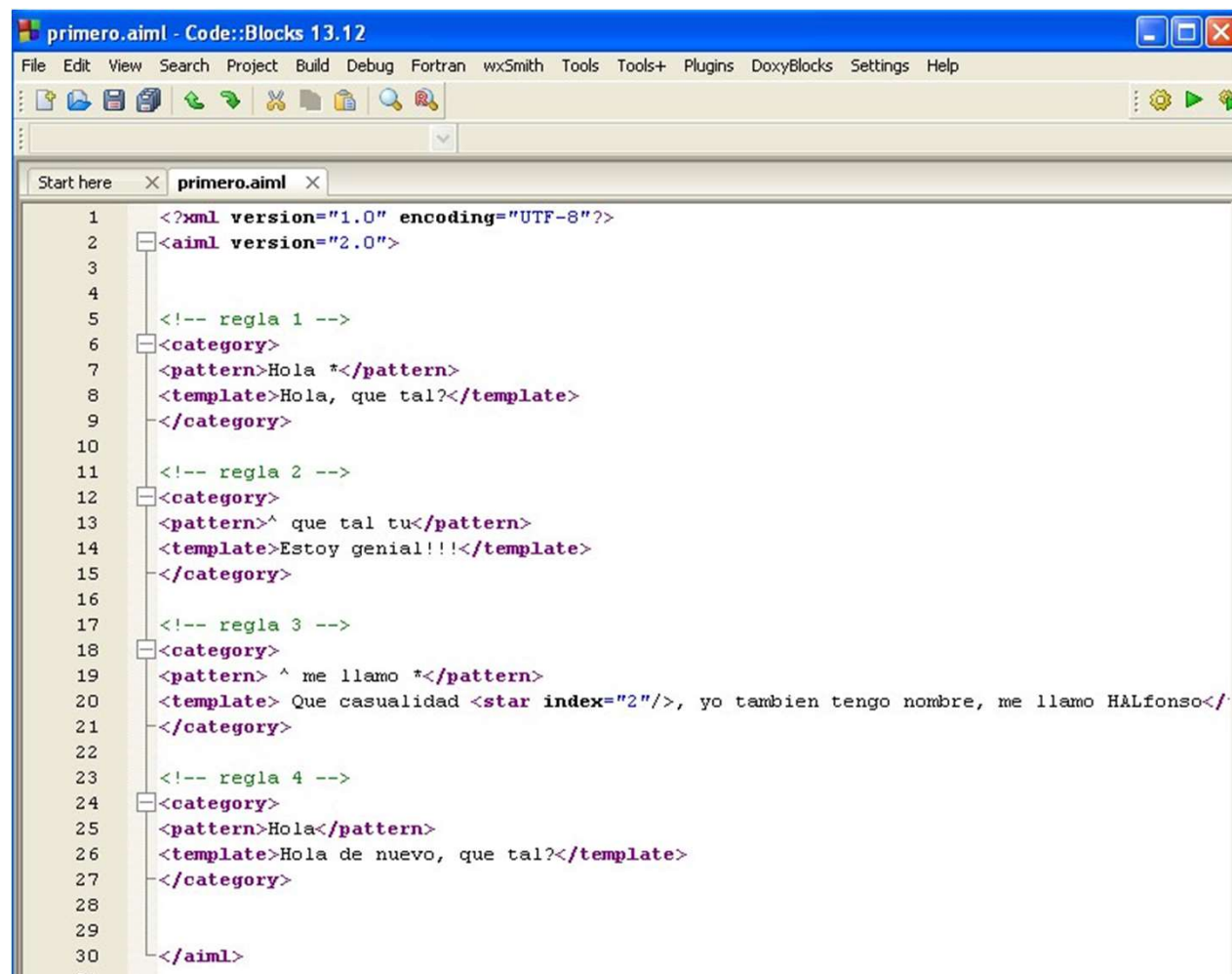


Cuidado con el uso de comas, junto a comodines...

## Resolución Ejercicio 2:

```
<category>
<pattern>_, que tal tu</pattern>
<template>Estoy genial!!!</template>
</category>
```

Human: buenos dias, que tal tu  
Robot: I have no answer for that.



```
primero.aiml - Code::Blocks 13.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Start here x primero.aiml x

1 <?xml version="1.0" encoding="UTF-8"?>
2 <aiml version="2.0">
3
4
5 <!-- regla 1 -->
6 <category>
7 <pattern>Hola *</pattern>
8 <template>Hola, que tal?</template>
9 </category>
10
11 <!-- regla 2 -->
12 <category>
13 <pattern>^ que tal tu</pattern>
14 <template>Estoy genial!!!</template>
15 </category>
16
17 <!-- regla 3 -->
18 <category>
19 <pattern> ^ me llamo *</pattern>
20 <template> Que casualidad <star index="2"/>, yo tambien tengo nombre, me llamo H&Lfonso</
21 </category>
22
23 <!-- regla 4 -->
24 <category>
25 <pattern>Hola</pattern>
26 <template>Hola de nuevo, que tal?</template>
27 </category>
28
29
30 </aiml>
```

## El lenguaje AIML

Variables



### En AIML hay 3 tipos de variables:

- **Propiedades del Bot**

Define la **información que quiere proporcionar el Bot sobre sí mismo**, y solo pueden ser asignadas por el BotMaster (el creador del Bot).

- **Predicados o Variables Globales**

En el lenguaje se denominan predicados, pero no tienen ningún tipo de asociación con el concepto de predicado en lógica. En realidad son variables globales. Dentro de AIML una variable es global cuando **su valor puede ser consultado o modificado fuera de una regla** (categoría).

- **Variables Locales**

Como su nombre indica son variables cuyo ámbito es **local a una regla (categoría)** y su valor no puede ser consultado fuera de dicha regla.

### Propiedades del Bot

Las propiedades del Bot viene definidas en el fichero "***properties.txt***" cuyo *path* es el siguiente: "***program-ab/bots/mybots/config/***"

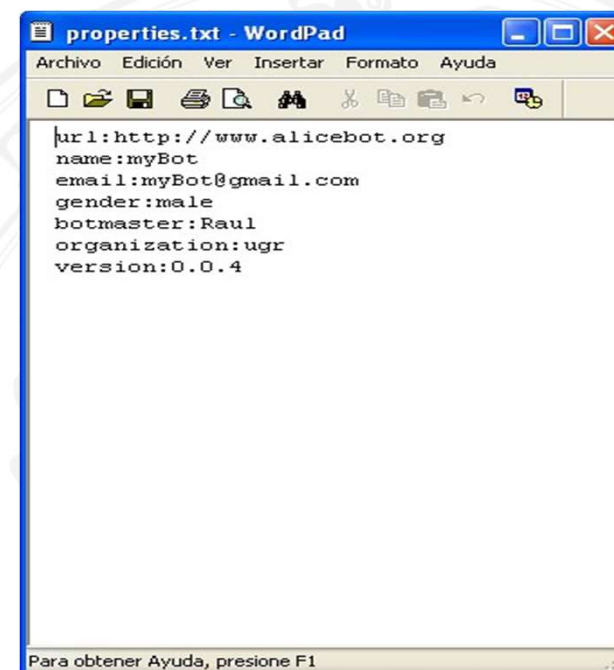
Por defecto vienen definidas las variables:  
url, name, email, gender, botmaster, organization, version.

Se puede añadir nuevas variables siguiendo el siguiente formato:

**<nombre variable>:<valor>**

Por ejemplo, añadamos los siguiente:

age:20  
job:estudiante



## Propiedades del Bot

Las propiedades del Bot viene definidas en el fichero "***properties.txt***" cuyo *path* es el siguiente: "***program-ab/bots/mybots/config/***"

Por defecto vienen definidas las variables:

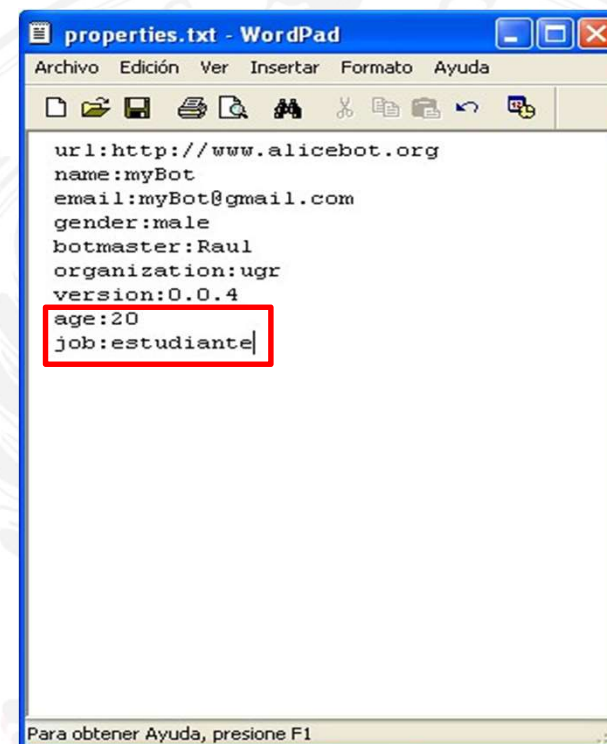
url, name, email, gender, botmaster, organization, version.

Se puede añadir nuevas variables siguiendo el siguiente formato:

<nombre variable>:<valor>

Por ejemplo, añadamos los siguiente:

age:20  
job:estudiante



```

url:http://www.alicebot.org
name:myBot
email:myBot@gmail.com
gender:male
botmaster:Raul
organization:ugr
version:0.0.4
age:20
job:estudiante
  
```

### Uso de las Propiedades del Bot

La sintaxis es

```
<bot name="x"/>
```

donde x representa la propiedad que se desea.

```
<category>
<pattern>Cual es tu edad</pattern>
<template>Tengo <bot name="age"/>
    anios</template>
</category>
```

Human: Cual es tu edad?

Robot: Tengo 20 anios

primero.aiml

```
29 <!-- regla 5 -->
30 <category>
31 <pattern>Cual es tu edad</pattern>
32 <template>Tengo <bot name="age"/> anios</template>
33 </category>
34
```

## Predicados o Variables Globales

La sintaxis es

```
<set name="x">value</set>
```

donde **x** representa el nombre de la variable.

Hay que tener en cuenta que AIML no tiene declaración de variables, así que hay que tener cuidado con el nombre que se le pone a las variables.

La sintaxis para acceder al valor de una variable global es  
donde x representa el nombre de la variable.

```
<get name="x"/>
```

**Las variables globales tienen sentido cuando el valor va a ser usado en varias reglas. Si no es así, las variables que deben usarse son las locales.**



## Predicados o Variables Globales

Vamos a modificar la regla 3 del fichero ***primero.aiml*** para almacenar el nombre del usuario. La versión original es:

### Regla Original

```

17 <!-- regla 3 -->
18 <category>
19   <pattern> ^ me llamo *</pattern>
20   <template> Que casualidad <star index="2"/>, yo tambien tengo nombre, me llamo HALfonso</
21 </category>

```

### Regla Modificada

```

17 <!-- regla 3 -->
18 <category>
19   <pattern> ^ me llamo *</pattern>
20   <template>
21     <set name="nombre"><star index="2"/></set>
22     <get name="nombre"/> es un bonito nombre.
23   </template>
24 </category>
25

```

## Predicados o Variables Globales

Vamos a incluir una nueva regla en ***primero.aiml*** para devolver el nombre del usuario.

```

38 <!-- regla 6 -->
39 <category>
40 <pattern>Cual es mi nombre</pattern>
41 <template>Tu nombre es <get name="nombre"/></template>
42 </category>
43

```

Y ahora probamos la siguiente secuencia.

Human: Cual es mi nombre?  
 Robot: Tu nombre es unknown  
 Human: Me llamo Raul  
 Robot: Raul  
       Raul es un bonito nombre.  
 Human: Cual es mi nombre?  
 Robot: Tu nombre es Raul

Una variable que se invoca sin haberle asignado un valor previamente devuelve siempre como valor "unknown"

## Variables Locales

La sintaxis es

```
<set var="x">value</set>
```

donde **x** representa el nombre de la variable.

La sintaxis para acceder al valor de una variable global es  
donde x representa el nombre de la variable.

```
<get var="x"/>
```

Las variables locales tiene como ámbito el **template** de la regla a diferencia de las variables globales.

```
44 <!-- regla 7 -->
45 <category>
46 <pattern>mi color favorito es el *</pattern>
47 <template>
48     <set var="color">star</set>
49     El <get var="color"/> es un color que no me gusta.
50 </template>
51 </category>
```

### El tag `<think>`

Tanto el acceso como la asignación de una variable provoca “eco” por pantalla. Así, si en el intérprete ponemos:

Human: Mi color favorito es el amarillo  
 Robot: **amarillo**  
**El amarillo es un color que no me gusta.**

Para evitar ese “eco”,  
 las asignaciones y acceso se  
 encierran en un par  
**`<think></think>`**

```
44 <!-- regla 7 -->
45 <category>
46 <pattern>mi color favorito es el *</pattern>
47 <template>
48     <set var="color"><star/></set>
49     El <get var="color"/> es un color que no me gusta.
50 </template>
51 </category>
```

```
44 <!-- regla 7 -->
45 <category>
46 <pattern>mi color favorito es el *</pattern>
47 <template>
48     <think><set var="color"><star/></set></think>
49     El <get var="color"/> es un color que no me gusta.
50 </template>
51 </category>
```

## El lenguaje AIML

Reducción  
Simbólica  
<srai>



Una herramienta muy importante en AIML es la reducción simbólica, ya que permite:

- **Simplificar las entradas** usando pocas palabras
- **Enlazar distintas entradas sinónimas** con un mismo *template*
- **Corregir errores ortográficos** por parte del usuario
- **Reemplazar expresiones coloquiales** por expresiones formales
- **Eliminar palabras innecesarias** en las entradas

En realidad la reducción simbólica es una **invocación recursiva a la propia base de conocimiento** (conjunto de categorías) con la intención de reducir el tamaño del conocimiento.

El tag asociado a esta operación es  
donde ... **representa un patrón a buscar.**

`<srai>...</srai>`



En nuestro fichero "primero.aiml", tenemos la regla 6 que responde a la pregunta "Cual es mi nombre?", pero hay muchas formas de expresarlo, por ejemplo "Dime mi nombre?", "Como me llamo?", "Te acuerda de cómo me llamo?", "Sabrías mi nombre?" para las cuales la respuesta es la misma, la que ofrece la regla 6.

Estas expresiones las podemos dividir entre aquellas que terminan por "mi nombre" y las que terminan con "como me llamo". Podría construir dos reglas nuevas con estos patrones cuyo *template* invoque a la regla 6.

```
<category>
<pattern>* mi nombre</pattern>
<template><srai>CUAL ES MI
NOMBRE</srai></template>
</category>

<category>
<pattern>^ como me llamo</pattern>
<template><srai>CUAL ES MI
NOMBRE</srai></template>
</category>
```



Insertamos estas dos nuevas reglas, la 8 y la 9

```

38 <!-- regla 6 -->
39 <category>
40 <pattern>Cual es mi nombre</pattern>
41 <template>Tu nombre es <get name="nombre"/></template>
42 </category>
43
44 <!-- regla 7 -->
45 <category>
46 <pattern>mi color favorito es el *</pattern>
47 <template>
48   <think><set var="color"><star/></set></think>
49   El <get var="color"/> es un color que no me gusta.
50 </template>
51 </category>
52
53 <!-- regla 8 -->
54 <category>
55 <pattern>* mi nombre</pattern>
56 <template><srai>CUAL ES MI NOMBRE</srai></template>
57 </category>
58
59 <!-- regla 9 -->
60 <category>
61 <pattern>^ como me llamo</pattern>
62 <template><srai>CUAL ES MI NOMBRE</srai></template>
63 </category>

```

Human: me llamo Raul  
 Robot: Raul  
 Raul es un bonito nombre ...  
 Human: como me llamo?  
 Robot: Tu nombre es Raul  
 Human: sabes mi nombre?  
 Robot: Tu nombre es Raul  
 Human: mi nombre?  
 Robot: I have no answer for that

Recordad que el comodín "\*" captura una o más palabras

**Ejercicio 3:** Aplica la reducción simbólica sobre la base de reglas actual de *primero.aiml* para contemplar los saludos más habituales.



## El lenguaje AIML

Sets y Maps

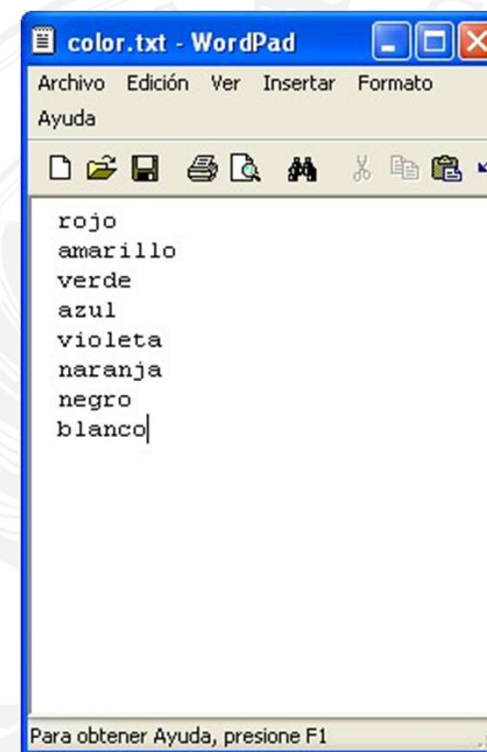


**Una de las aportaciones relevantes que se incluyen en la versión AIML 2.0 es el uso de Sets y Maps.**

**Un set es una lista de cadenas que se expresa sobre un fichero.** El nombre del fichero da la denominación del set y la extensión debe ser "txt". Este fichero debe estar ubicado en la carpeta "sets".

En la siguiente figura se muestra un ejemplo de set "color" donde se ilustra la sintaxis que debe tener el fichero.

La secuencia es un valor en cada línea del archivo txt.



**Los sets permiten hacer una reducción importante en el número de reglas.** Por ejemplo, si defino un set con la lista de colores, con dos únicas reglas puedo determinar si algo es un color o no.

```
<category>
<pattern>Es <set>color</set> un color</pattern>
<template>Si, <star/> es un color </template>
</category>

<category>
<pattern>Es * un color</pattern>
<template>No, <star/> no es un color</template>
</category>
```

**La secuencia <set>color</set> en el patrón, verifica si la entrada coincide con alguna de las palabras que aparecen en el fichero "color.txt".** Si es así, la regla se dispara. En otro caso, será la segunda regla la que se dispare.

**Set tiene mayor prioridad que "\*" y "^", pero menos que "\_" y "#"**

Si añadimos estas reglas a nuestro fichero ***primero.aiml***, y hacemos la secuencia de preguntas que se indican, se obtendrá:

```

65 <!-- regla 10 -->
66 <category>
67 <pattern>es <set>color</set> un color</pattern>
68 <template>Si, <star/> es un color.</template>
69 </category>
70
71
72 <!-- regla 11 -->
73 <category>
74 <pattern>es * un color</pattern>
75 <template>No, <star/> no es un color.</template>
76 </category>

```

Human: Es amarillo un color?

Robot: Si, amarillo es un color.

Human: Es rojo un color?

Robot: Si, rojo es un color.

Human: Es verde un color?

Robot: Si, verde es un color.

Human: Es lapiz un color?

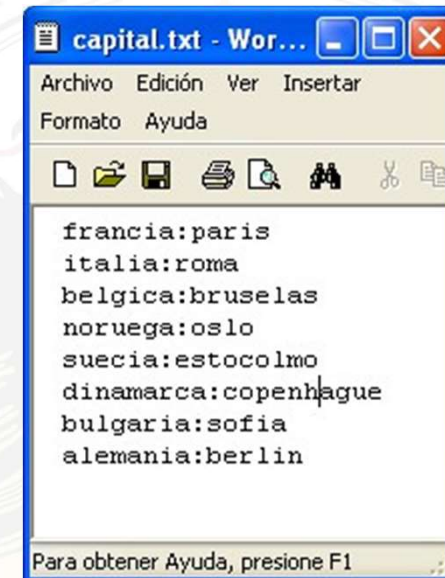
Robot: No, lapiz no es un color.

**Maps** representa el tipo de dato **diccionario** y, al igual que los *sets*, se codifica en un fichero independiente donde el nombre del fichero da nombre al *maps* y la extensión ha de ser “txt”. Este fichero debe estar alojado en la carpeta **maps**.

En cada línea del fichero se codifica una entrada con el siguiente formato:

```
cadena1:cadena2
```

Como ejemplo vamos a definir un *map* nombre “**capital**” para asociar a cada capital.





Vamos a definir una regla, que responda a cual es la capital de un país.

```
<category>
<pattern>Cual es la capital de *</pattern>
<template>
  La capital de <star/> es <map name="capital"><star/></map>.
</template>
</category>
```

La operación **<map name="capital">KEY</map>** devuelve el valor asociado a la clave **KEY**.

Añadimos esta regla al fichero primero.aiml.

```

79 <!-- regla 12 -->
80 <category>
81 <pattern>Cual es la capital de *</pattern>
82 <template>La capital de <star/> es <map name="capital"><star/></map>.</template>
83 </category>
84

```

Human: Cual es la capital de francia?

Robot: La capital de francia es paris.

Human: Cual es la capital de italia?

Robot: La capital de italia es roma.

Human: Cual es la capital de Cuba?

Robot: La capital de Cuba es unknown.

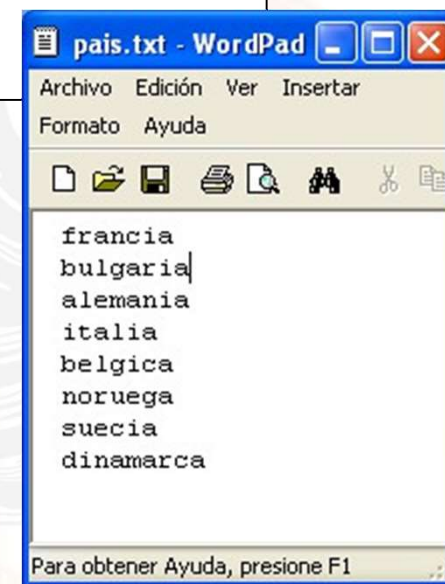
Para que no ocurra que no encuentre la respuesta, **se suele definir un set con las claves del *map***. En este caso, un set de países, y así podemos contemplar a través de 2 reglas, si sabemos o no la capital de un determinado país.

```

79 <!-- regla 12 -->
80 <category>
81 <pattern>Cual es la capital de <set>pais</set></pattern>
82 <template>La capital de <star/> es <map name="capital"><star/></map>.</template>
83 </category>
84
85 <!-- regla 13 -->
86 <category>
87 <pattern>Cual es la capital de *</pattern>
88 <template>No se cual es la capital de <star/>.</template>
89 </category>

```

Human: Cual es la capital de francia?  
 Robot: La capital de francia es paris.  
 Human: Cual es la capital de italia?  
 Robot: La capital de italia es roma.  
 Human: Cual es la capital de Cuba?  
 Robot: No se cual es la capital de Cuba



AIML tiene implícitamente definidos los siguientes sets y maps:

- `<set>number</set>`  
Números naturales
- `<map name="successor">`  
Dado un número natural "n" devuelve "n+1"
- `<map name="predecessor">`  
Dado un número natural "n" devuelve "n-1"
- `<map name="plural">`  
Devuelve el plural de un palabra en singular (sólo Inglés)
- `<map name="singular">`  
Devuelve el singular de un palabra en plural (sólo Inglés)

**Ejercicio 4:** Construye un fichero set, llamado "compi.txt" que contenga al menos el nombre de 5 compañeros de clase, y define 2 ficheros map, uno que asocie a cada uno de ellos su color de pelo y llámalo "pelo.txt" y el otro que le asocie su color de ojos y llámalo "ojos.txt".

Una vez hecho eso, construye un fichero llamado "ejer4.aiml" y define el conjunto de reglas necesario para responder a las preguntas sobre color de ojos y de pelo de un compañero.

## El lenguaje AIML

### Contexto



**El **contexto** es fundamental para que una conversación mantenga coherencia y tiene que ver con recordar cosas que el bot ha dicho previamente.**

En AIML hay 3 elementos para recordar el contexto:

- Los predicados o variables globales (vistas previamente)
- El tag <that>
- Un “set” predefinido en el lenguaje llamado “topic”



### El tag <that>

**El bot recuerda la última respuesta.** En base a ese respuesta puede alterar la respuesta a la siguiente pregunta.

El tag <that> se sitúa entre <pattern> y <template> siendo su sintaxis la siguiente:

#### Regla 1

```
<category>
<pattern>Si</pattern>
<that> TE GUSTA EL CAFE
</that>
<template>
  Lo prefieres solo o con leche
</template>
</category>
```

#### Regla 2

```
<category>
<pattern>^ cafe ^</pattern>
<template>
  Te gusta el cafe
</template>
</category>
```

Obviamente, para que se dispare la Regla 1, es necesario que justo antes en la conversación se haya disparado una regla como la Regla 2.

## El tag <that>

Añadimos estas dos reglas al fichero primero.aiml.

```

92 <!-- regla 14 -->
93 <category>
94 <pattern>^ cafe ^</pattern>
95 <template>Te gusta el cafe.</template>
96 </category>
97
98
99 <!-- regla 15 -->
.00 <category>
.01 <pattern>Si</pattern>
.02 <that>TE GUSTA EL CAFE</that>
.03 <template>Lo prefieres solo o con leche.</template>
.04 </category>

```

Human: esta mañana me tome un  
cafe

Robot: Te gusta el cafe.

Human: Si

Robot: Lo prefieres solo o con  
leche.

## El tag <that>

Otro ejemplo:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<aiml version = "1.0.1" encoding = "UTF-8"?>
  <category>
    <pattern>WHAT ABOUT MOVIES</pattern>
    <template>Do you like comedy movies</template>
  </category>

  <category>
    <pattern>YES</pattern>
    <that>Do you like comedy movies</that>
    <template>Nice, I like comedy movies too.</template>
  </category>

  <category>
    <pattern>NO</pattern>
    <that>Do you like comedy movies</that>
    <template>Ok! But I like comedy movies.</template>
  </category>
</aiml>
```

Human: Hi Alice! What about movies?  
 Robot: Do you like comedy movies?  
 Human: No  
 Robot: Ok! But I like comedy movies.

Ejemplo extraído de  
[https://www.tutorialspoint.com/aiml/aiml\\_that\\_tag.htm](https://www.tutorialspoint.com/aiml/aiml_that_tag.htm)

**<set name="topic"> / <topic name="x"></topic>**

**Esta variable global predefinida** en el lenguaje **permite agrupar las reglas** de manera que **estas solo se activan cuando la conversación se centra en un tema concreto**.

Por defecto, el valor de "topic" es "unknown". Como ha sido nuestro caso, a lo largo de todo este tutorial, ya que nunca le fijamos un valor.

Lo primero es definir un bloque de reglas sobre un tema, para ello se encierran las reglas entre un <topic name="tema"> ... </topic>.

```
<topic name="cafe">
<category> ..... </category>
.....
<category> ..... </category>
</topic>
```

Para fijar un tema, en el "template" de alguna regla se usa <set name="topic">

```
<template> te gusta el <set name="topic"> cafe
</set></template>
```

**<set name="topic"> / <topic name="x"></topic>**

El tag <topic> ayuda a utilizar categorías escritas dentro del contexto de una temática concreta.

### Ejemplo

```
<?xml version = "1.0" encoding = "UTF-8"?>
<aiml version = "1.0.1" encoding = "UTF-8"?>
  <category>
    <pattern>LET DISCUSS MOVIES</pattern>
    <template>Yes <set name = "topic">movies</set></template>
  </category>

  <topic name = "movies">
    <category>
      <pattern> * </pattern>
      <template>Watching good movie refreshes our minds.</template>
    </category>

    <category>
      <pattern> I LIKE WATCHING COMEDY! </pattern>
      <template>I like comedy movies too.</template>
    </category>

  </topic>
</aiml>
```

```
Human: let discuss movies
Robot: Yes movies
Human: Comedy movies are nice to watch
Robot: Watching good movie refreshes our minds.
Human: I like watching comedy
Robot: I too like watching comedy.
```

Ejemplo extraído de  
[https://www.tutorialspoint.com/aiml/aiml\\_topic\\_tag.htm](https://www.tutorialspoint.com/aiml/aiml_topic_tag.htm)

## El lenguaje AIML

Random,  
Estructura  
Condicional y  
Ciclos





**No responder exactamente de la misma forma ante la misma pregunta, o ante preguntas similares, ayuda al bot a dar la impresión de presentar un comportamiento más humano.**

El lenguaje AIML tiene el **tag <random>**, para aportar al conocimiento este comportamiento. La sintaxis es la siguiente:

```
<random>
  <li> .... </li>
  <li> ... </li>
  .....
  <li> ... </li>
</random>
```

### Ejemplo

```
<category>
  <pattern>hola *</pattern>
  <template>
    <random>
      <li> Hola! </li>
      <li> Buenas! Qué tal? </li>
    </random>
  </template>
</category>
```

Los pares <li>...</li> separan las distintas salidas posible. El comportamiento de esta sentencia, es que aleatoriamente elige entre uno de los pares <li></li> para ofrecerlo como salida.



### En AIML también hay una estructura condicional.

Esta estructura condicional funciona como el “switch” de C. Su sintaxis es la siguiente:

#### Para variables locales

```
<condition var = "x">  
<li value="x1"> .... </li>  
<li value="x2"> ... </li>  
.....  
<li> ... </li>  
</condition>
```

#### Para variables globales

```
<condition name = "x">  
<li value="x1"> .... </li>  
<li value="x2"> ... </li>  
.....  
<li> ... </li>  
</condition>
```

Los pares <li value></li> separan los distintos casos, y el último <li></li> se aplica cuando ninguno de los casos anteriores se cumple.

Veamos un ejemplo de uso

En nuestro fichero primero.aiml, nos dimos cuenta que si le pedíamos nuestro nombre en la regla 6 y aún no lo habíamos almacenado ya que no se había invocado a la regla 4, nos decía: “**Tu nombre es unknown**”.

Vamos a corregir la regla 6 de la siguiente manera: si ya se ha asignado valor a la variable “nombre” entonces que funcione como está ahora mismo, pero si no, que diga que aún no me has dicho tu nombre.

### Regla Original

```
38 <!-- regla 6 -->
39 <category>
40 <pattern>Cual es mi nombre</pattern>
41 <template>Tu nombre es <get name="nombre"/></template>
42 </category>
43
```

### Regla Modificada

```
38 <!-- regla 6 -->
39 <category>
40 <pattern>Cual es mi nombre</pattern>
41 <template>
42 <condition name="nombre">
43 <li value="unknown"> Aun no me has dicho tu nombre</li>
44 <li>Tu nombre es <get name="nombre"/></li>
45 </condition>
46 </template>
47 </category>
```

Human: como me llamo?

Robot: Aun no me has dicho tu nombre

Human: me llamo Raul

Robot: Raul

Raul es un bonito nombre...

Human: como me llamo?

Robot: Tu nombre es Raul

Otro elemento básico en un lenguaje de programación son los ciclos.

AIML tiene una forma muy peculiar para la construcción de ciclos.

**Son ciclos del tipo “*mientras condición, hacer un bloque de operaciones*”** y eso implica el uso implícito de la operación de condición.

Veámoslo con un ejemplo: Supongamos que queremos construir una regla que cuente hasta un determinado número.

```
<category>
<pattern>Cuenta hasta <set>number</set></pattern>
<template>

</template>
</category>
```

<category>

<pattern>Cuenta hasta <set>number</set></pattern>

<template>

</template>

</category>

Planteo la regla y defino como patrón responder a las consultas que son de la forma “Cuenta hasta n” siendo n un número.

```
<category>
<pattern>Cuenta hasta <set>number</set></pattern>
<template>
  <think>
    <set var="contador">1</set>
    <set var="salida">1</set>
  </think>
</template>
</category>
```

Lo que tengo que hacer es un bucle contador. Así, declaro una variable local “contador” para ir almacenando los distintos valores hasta llegar a *number*.

Además, declaro otra variable local “salida”, que va a ir almacenando la secuencia de números por los que va pasando contador. La idea es que salida se comporte como una cadena de caracteres.

Ambas variables se inicializan con el valor 1 y están incluidas en un bloque <think> para que no produzcan “eco” por pantalla.

```

<category>
<pattern>Cuenta hasta <set>number</set></pattern>
<template>
  <think>
    <set var="contador">1</set>
    <set var="salida">1</set>
  </think>
  <condition var="contador">
    <li><value><star/></value><get var="salida"/></li>
    <li></li>
  </condition>
</template>
</category>

```

Ahora planteo un comportamiento diferente en función de una condición.

La condición revisa el valor de la variable "contador".

El primer caso nos dice que **si el valor del comodín**, que en este caso es el número introducido en el patrón, **coincide con el valor de la variable "contador"**, entonces devuelva por el terminal el **valor de la variable salida**.

Hasta ahora habíamos visto <li value="x">, pero cuando hay un valor que no podemos encerrar entre comillas, se puede descomponer de la forma que aquí aparece, es decir,  
 <li><value>valor</value>.....</li>

```

<category>
<pattern>Cuenta hasta <set>number</set></pattern>
<template>
  <think>
    <set var="contador">1</set>
    <set var="salida">1</set>
  </think>
  <condition var="contador">
    <li><value><star/></value><get var="salida"/></li>
    <li>
      <think>
        <set var="contador">
          <map name="successor"><get var="contador"/></map>
        </set>
      </think>
    </li>
  </condition>
</template>
</category>

```

El caso anterior, es la condición de salida. Así que **este segundo caso corresponde al bloque del ciclo** en un lenguaje de programación convencional. En este caso, lo que tenemos que hacer es **incrementar el contador**. La forma que tiene AIML para hacer esto es hacer uso del map "successor".

Este grupo de acciones lo que hace es **calcular el sucesor del valor actual de "contador"** y el resultado de ese cálculo lo vuelve a almacenar en la variable "contador".



```

<category>
<pattern>Cuenta hasta <set>number</set></pattern>
<template>
  <think>
    <set var="contador">1</set>
    <set var="salida">1</set>
  </think>
  <condition var="contador">
    <li><value><star/></value><get var="salida"/></li>
    <li>
      <think>
        <set var="contador">
          <map name="successor"><get var="contador"/></map>
        </set>
        <set var="salida"><get var="salida"/> <get var="contador"/></set>
      </think>
    </li>
  </condition>
</template>
</category>

```

Además, de incrementar “contador”, actualizo la variable “salida”, en este caso, mediante una concatenación de cadenas.

Se puede observar que el nuevo valor de “salida” es su anterior valor al que se le concatena el valor de la variable “contador”. No se ve bien, pero es importante, hay un espacio en blanco entre el get de “salida” y el get de “contador”, para separar los números.

```

<category>
<pattern>Cuenta hasta <set>number</set></pattern>
<template>
  <think>
    <set var="contador">1</set>
    <set var="salida">1</set>
  </think>
  <condition var="contador">
    <li><value><star/></value><get var="salida"/></li>
    <li>
      <think>
        <set var="contador">
          <map name="successor"><get var="contador"/></map>
        </set>
        <set var="salida"><get var="salida"/> <get var="contador"/></set>
      </think>
      <loop/>
    </li>
  </condition>
</template>
</category>

```

Termino el caso con el comando `<loop/>`, que indica que se revise el valor de la última condición examinada. En nuestro caso la única condición que hay.

Si hubiera varias condiciones anidadas, el ciclo implica a la más interna.

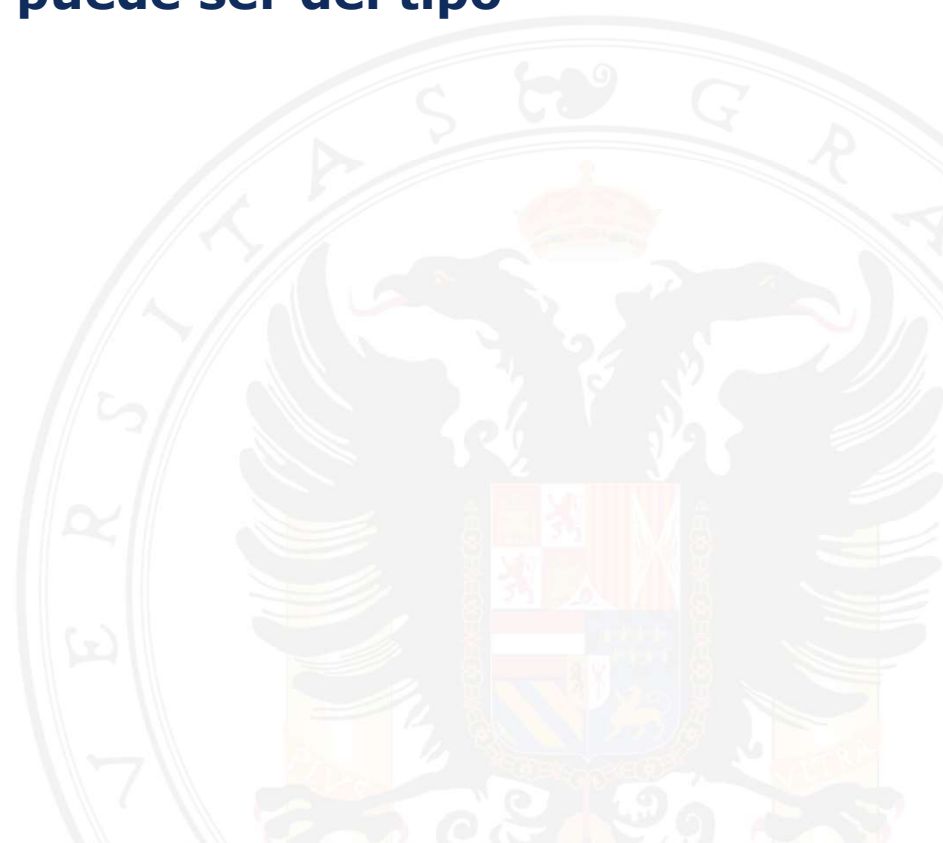
Así quedaría la regla al final.

```

08 <!-- regla 16 -->
09 <category>
10 <pattern>Cuenta hasta <set>number</set></pattern>
11 <template>
12 <think>
13 <set var="contador">1</set>
14 <set var="salida">1</set>
15 </think>
16 <condition var="contador">
17 <li><value>star/</value><get var="salida"/></li>
18 <li>
19 <think>
20 <set var="contador">
21 <map name="successor"><get var="contador"/></map>
22 </set>
23 <set var="salida"><get var="salida"/> <get var="contador"/></set>
24 </think>
25 <loop/>
26 </li>
27 </condition>
28 </template>
29 </category>

```

**Ejercicio 5:** Toma el conocimiento desarrollado en el ejercicio 4 que responde al color de ojos y pelo de algunos compañeros y añade las reglas necesarias para que dado un color concreto de pelo o de ojos, te devuelva los nombres de los compañeros que coinciden con esa propiedad. La pregunta puede ser del tipo “Que compañeros son rubios?”



## El lenguaje AIML

Aprender



Una de las particularidades más importante de AIML es que da la posibilidad al bot de **aprender de los usuarios**.

Para eso se usan dos tag **<learn>** y **<learnf>**.

Ambos se usan de la misma manera, y la única diferencia entre ambos es **si lo que aprende solo se usa en la conversación actual**, en ese caso, **se usa <learn>** o **si lo aprendido se desea que se mantenga como parte de la base de reglas**, en cuyo caso **hay que usar <learnf>**.

Aquí explicaremos el uso de **<learn>**, asumiendo que se hace igual en el caso de querer usar **<learnf>**.

Una aclaración, cuando se usa **<learnf>** el botMaster pierde un tanto el control del bot y puede que este aprenda "cosas malas"

Aprenderemos su uso con un ejemplo:

Las reglas 12 y 13 del fichero primero.aiml, definen el comportamiento para responder cual es la capital de un determinado país. La 12 en el caso afirmativo de que el país esté en el **set país** y la 13 indicando que no lo sabemos.

Vamos a intentar complementar estas dos reglas con una adicional que, si no sabemos el país o su capital, la aprendamos.

En concreto, la entrada que vamos a permitir es la siguiente:

La capital de \* es \*

Lo que tendremos que hacer es lo siguiente:

1. Verificar si el país está en set país en cuyo caso invocamos a la regla 12.
2. Verificar si es una capital que ya habíamos aprendido antes.
3. Si no es ninguna de las dos anteriores, pasamos a aprender la nueva regla.



Las reglas 12 y 13 son las siguientes:

```

79 <!-- regla 12 -->
80 <category>
81 <pattern>Cual es la capital de <set>pais</set></pattern>
82 <template>La capital de <star/> es <map name="capital"><star/></map>.</template>
83 </category>
84
85 <!-- regla 13 -->
86 <category>
87 <pattern>Cual es la capital de *</pattern>
88 <template>No se cual es la capital de <star/>.</template>
89 </category>

```

Voy a transformar el **template** de la regla 13 para que devuelva "No lo se". Esto lo hago simplemente por comodidad para la nueva regla.

```

84 <!-- regla 12 -->
85 <category>
86 <pattern>Cual es la capital de <set>pais</set></pattern>
87 <template>La capital de <star/> es <map name="capital"><star/></map>.</template>
88 </category>
89
90 <!-- regla 13 -->
91 <category>
92 <pattern>Cual es la capital de *</pattern>
93 <template>No lo se</template>
94 </category>

```

**<category>**

**<pattern>**la capital de \* es \***</pattern>**

**<template>**

**<template>**

**</category>**

Propongo la estructura básica de la regla y fijo el patrón con dos comodines, el primero recoge el país y el segundo la capital de dicho país.

```
<category>
<pattern>la capital de * es *</pattern>
<template>
  <think>
    <set var="cap"><srai>CUAL ES LA CAPITAL DE <star/></srai></set>
  </think>
</template>
</category>
```

Añado en el **template** una invocación a la regla 12 o 13. Esto devolverá "NO LO SE" si se dispara la regla 13.

En otro caso, devuelve "LA CAPITAL DE ...".

El valor devuelto se almacena en la variable local "cap", y como veis está incrustado en un bloque <think> para que no presente "eco" por pantalla.

```

<category>
<pattern>la capital de * es *</pattern>
<template>
  <think>
    <set var="cap"><srai>CUAL ES LA CAPITAL DE <star/></srai></set>
  </think>
  <condition var="cap">
    <li value="NO LO SE">
    </li>
  </condition>
</template>
</category>

```

Ahora propongo una condición para determinar si el bot sabe la respuesta.

Si no lo sabe, la variable "cap" contendrá "NO LO SE". Así, que este es el caso en el que quiero aprender la información proporcionada por el usuario.

```

<category>
<pattern>la capital de * es *</pattern>
<template>
  <think>
    <set var="cap"><srai>CUAL ES LA CAPITAL DE <star/></srai></set>
  </think>
  <condition var="cap">
    <li value="NO LO SE">
      <learn>
        <category>
          <pattern>CUAL ES LA CAPITAL DE <eval><star/></eval></pattern>
          <template>
            La capital de <eval><star/></eval> es <eval><star index="2"/></eval>
          </template>
        </category>
      </learn>
    </li>
  </condition>
</template>
</category>

```

Pues, aquí aparece la sintaxis de `<learn>`.

Como se puede observar, dentro del bloque *learn* aparece incrustada la estructura de una regla con su *category*, su *pattern* y su *template*.

Así, la nueva regla que se propone es que ante la entrada de “CUAL ES LA CAPITAL DE ...”, siendo ... un país concreto, la respuesta será “La capital de .1. es .2., donde .1. es un país concreto y .2. es una ciudad concreta.

El *tag* `<eval>` lo que hace es transformar el comodín por el valor concreto con el que se ha instanciado esa regla. Si no estuviera el `<eval>`, no instancia al valor concreto el comodín, sino que deja directamente el comodín.

```

<category>
<pattern>la capital de * es *</pattern>
<template>
  <think>
    <set var="cap"><srai>CUAL ES LA CAPITAL DE <star/></srai></set>
  </think>
  <condition var="cap">
    <li value="NO LO SE">
      <learn>
        <category>
          <pattern>CUAL ES LA CAPITAL DE <eval><star/></eval></pattern>
          <template>
            La capital de <eval><star/></eval> es <eval><star index="2"/></eval>
          </template>
        </category>
      </learn>
      Recordare que la capital de <star/> es <star index="2"/>.
    </li>
  </condition>
</template>
</category>

```

Antes de terminar el caso, y una vez que he propuesto la nueva regla a incluir en la base de reglas, saco un mensaje al usuario indicando que "Recordare que la capital de ..."

```

<category>
<pattern>la capital de * es *</pattern>
<template>
  <think>
    <set var="cap"><srai>CUAL ES LA CAPITAL DE <star/></srai></set>
  </think>
  <condition var="cap">
    <li value="NO LO SE">
      <learn>
        <category>
          <pattern>CUAL ES LA CAPITAL DE <eval><star/></eval></pattern>
          <template>
            La capital de <eval><star/></eval> es <eval><star index="2"/></eval>
          </template>
        </category>
      </learn>
      Recordare que la capital de <star/> es <star index="2"/>.
    </li>
  </condition>
  <li>
    Ya lo sabia.
  </li>
</template>
</category>

```

Por último, añado el caso en que la variable “cap” no tenga el valor “NO LO SE”. En esta situación, le indico al usuario que “Ya lo sabia”.



### Así quedaría la nueva regla

```

35 <!-- regla 17 -->
36 <category>
37 <pattern>la capital de * es *</pattern>
38 <template>
39   <think>
40     <set var="cap"><srai>CUAL ES LA CAPITAL DE <star/></srai></set>
41   </think>
42   <condition var="cap">
43     <li value="NO LO SE"> ←
44     <learn>
45       <category>
46       <pattern>
47         CUAL ES LA CAPITAL DE <eval><star/></eval>
48       </pattern>
49       <template>
50         La capital de <eval><star/></eval> es <eval><star index="2"/></eval>
51       </template>
52     </category>
53   </learn>
54   Recordare que la capital de <star/> es <star index="2"/>.
55 </li>
56 <li>
57   Ya lo sabia.
58 </li>
59 </condition>
60 </template>
61 </category>

```

**Ojo!** Si en el template de la regla 13 tenéis "No lo se." (con punto), el bot siempre respondería "Ya lo sabia."

### Ejercicio 6:

- a) Corrige la regla 17 para que detecte la no coincidencia entre el nombre de la capital que introduce el usuario con la que el bot tenía almacenada.**
  
- a) Toma el conocimiento obtenido en el ejercicio 5 que trata sobre el color de pelo y de ojos de compañeros, y añade las reglas necesarias para ante una afirmación del tipo “Luis tiene los ojos azules y el pelo moreno” aprenda esa información para responder a las preguntas sobre el color de pelo o de ojos de Luis.**

## El lenguaje AIML

### Algunos Enlaces

- [AIML 2.0 Working Draft](#)
- [A.L.I.C.E. The Artificial Intelligence Foundation](#)
- [Pandorabots](#)
- [AIML Quick Guide](#)
- [Build a Simple Virtual Assistant with AIML 2.0](#)