Repaso

¿Cómo es una aplicación en el paradigma lógico?
¿Por qué decimos que es declarativo? ¿En qué me ayuda el motor de inferencia?
¿Qué es una variable en Lógico? ¿Qué era en Imperativo?

Unificación vs. Asignación

Si yo hago

```
Z is X * 2.
```

Si Z es una variable sin ligar, se unificará el resultado de evaluar la expresión `X * 2` a la variable `Z`.
Esto sólo funciona si X tiene un valor, en caso contrario PROLOG dará error (todas las variables del lado derecho del *is* deben estar unificadas).
Técnicamente, PROLOG **no asigna** sino que **unifica**. La diferencia está en que si Z es una variable ligada a algún valor, se realizará una comparación.

Ejemplo: defino una cláusula

```
testDoble(Z, X):- Z is X * 2.
```

Veamos algunas consultas…

```
?- testDoble(4, 2).
true

?- testDoble(Incognita, 2).
Incognita = 4

?- testDoble(4, X).
```

Error, la expresión `X * 2` contiene a la variable `X` sin unificar

En cambio, si hago
`Z = X * 2`, estoy preguntando si `Z` (que se asume una variable ligada/unificada) es igual que `X * 2` (`X` también debe estar ligada). Las variables `X` y `Z` pueden unificarse cuando:

- yo le envío valores a los argumentos de un predicado
- en un predicado anterior llego a resolverlos y dejan de ser incógnita

Ojo: Tener en cuenta que la consulta

```
?- 4 = 2 * 2
```

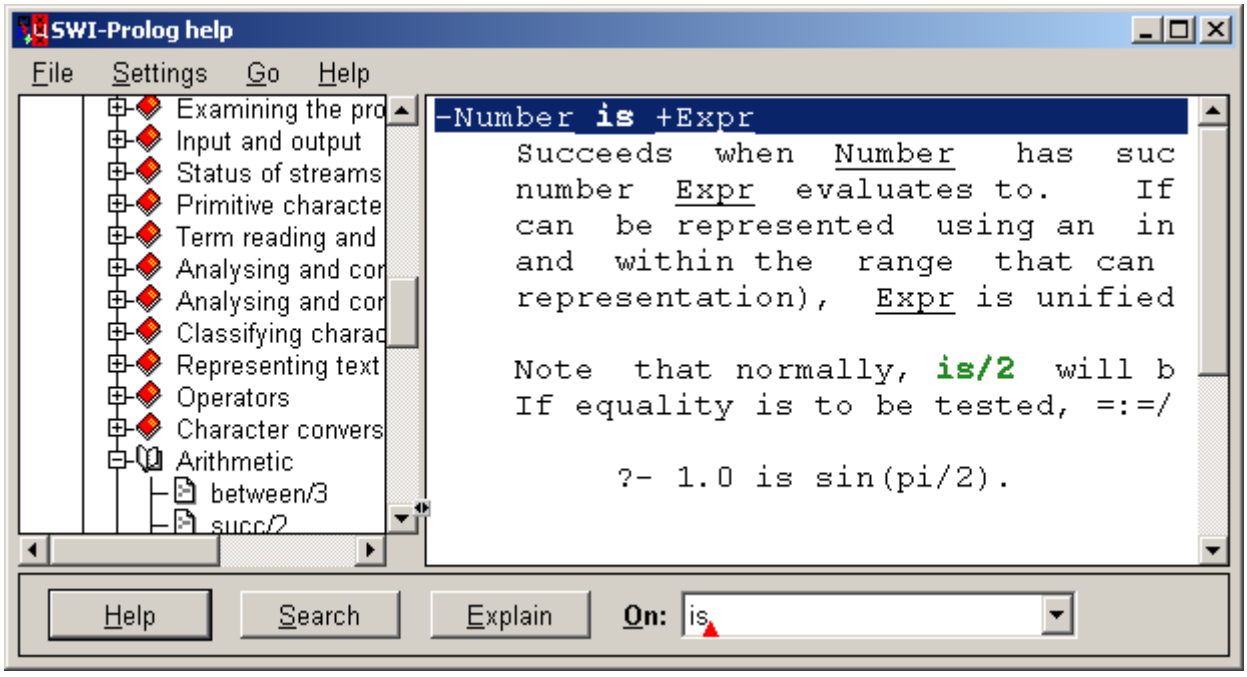
Nos devuelve
`false` ← PROLOG no evalúa la expresión `2 * 2`, entonces `2 * 2` no es exactamente igual a `4`. `4` sí es igual a `4`

Mientras que

```
?- 4 is 2 * 2
```

`true` ← `4` es "unificable" a `2 * 2`, pero no igual a `2 * 2` en el sentido lógico. El `is` evalúa la expresión del lado derecho, mientras que el = unifica términos directamente:

Pueden verlo en el manual online de Prolog: Help → online Manual…



Recordar que tanto
`X = X + 1` como
`X is X + 1` es una contradicción: nunca una variable puede ser igual a sí misma más uno.

Recursividad y algo de aritmética.

¿Qué es recursividad? Repasando la carpeta de Algoritmos y Estructuras de Datos… es una forma de definir algo en función de sí mismo. La recursividad en lógico consistirá en definir un predicado en términos de sí mismo. En otras palabras:

¿Qué elementos tenemos en lógico?	Predicados / cláusulas.
¿Qué es un programa lógico?	Un conjunto de cláusulas que definen relaciones entre individuos.
Entonces, ¿qué va a ser recursivo?	La definición de un predicado (más adelante, no va a ser lo único que tengamos).

Ejemplo bueno para apoyarse: predicado ancestro.

Buscamos primero la forma de contarlo *declarativamente:*

- *¿Quiénes son mis ancestros?*
  Mis padres…
  y los ancestros de mis padres.

- Observar que estoy definiendo ancestro en términos de ancestro.
  Recordemos la diferencia entre el castellano y la lógica - entre el "y" y el "o"

```
ancestro(Persona, Ancestro):- padre(Persona, Ancestro).
ancestro(Persona, Ancestro):- padre(Persona, Padre), ancestro(Padre, Ancestro).
```

Esto se lee: “ancestro es mi padre o bien el ancestro de mi papá”

Ejemplo para hacer en clase:

- Luisa y Juan son alumnos de Daniel.
- Ana es alumna de Luisa.
- Diana y Nahuel son alumnos de Nico.
- Tamara es alumna de Nahuel.
- A la fiesta pueden ir: Nico, Daniel, los alumnos de algún profesor que pueda ir y los lindos.
- Julián, Brad y Heidi son lindos.
- Brad y Rulo tienen pecas.

```
alumno(luisa, daniel).
alumno(juan, daniel).
alumno(ana, luisa).
alumno(diana, nico).
alumno(nahuel, nico).
alumno(tamara, nahuel).
lindo(julian).
lindo(brad).
lindo(heidi).
pecas(brad).
pecas(rulo).

puedeIr(nico).
puedeIr(daniel).
puedeIr(Persona):- alumno(Persona, Profesor), puedeIr(Profesor).
puedeIr(Persona):- lindo(Persona).
```

Aritmética

El factorial no es una función. Como siempre dentro del paradigma lógico, es una relación entre dos números: un número y su factorial.
A los predicados se les indica la aridad o cantidad de argumentos que recibe. En este caso, es **factorial/2**.

```
factorial(0, 1).
factorial(N, FN):-
      N1 is N – 1,
      factorial(N1, FN1),
      FN is N * FN1.
```

Cómo se lee: el factorial de N es N multiplicado por el factorial de N1 siendo que N1 es N - 1. La variable N1 la necesito ya que no puedo enviar expresiones como argumentos: no puedo hacer factorial (N – 1, FN1), primero hay que unificar esa expresión en una variable.

Cómo es el Pattern matching en Lógico
cuando llegamos al factorial de 0:
Matchea la primera cláusula: 0 = 0
¡Pero también matchea la segunda: N = 0!

En Lógico hay que agregar la restricción (N > 0) en la segunda cláusula, *no solamente por si quieren consultar el factorial de un número negativo*, sino también porque *factorial* es una relación, entonces el motor busca todas las soluciones posibles…

Recordar inducción: Caso base y caso recursivo. No olvidarse de pensar el caso base (es una de las contras que tiene la solución recursiva).

Inversibilidad

¿Qué diferencia hay entre una función y una relación? Una relación es más abarcativa, me permite ver más cosas: hay un concepto que se llama *inversibilidad* y es el que me permite hacer las siguientes preguntas:

¿Cuánto es el factorial de 4?
```
?- factorial(4, X)
```

¿Cuál es el número cuyo factorial da 6?
```
?- factorial (X, 6)
```

Y también puedo preguntar si el factorial de 3 es 6:
```
?- factorial(3, 6)
```

¿Es mejor o peor? Ni uno ni otro, son formas diferentes de hacerlo.

Ok, pero ¿qué es la inversibilidad?
Definimos **inversibilidad** como la propiedad de hacer consultas sin importarme si los argumentos están instanciados[1] o no. Esto es lo que le da mucho power al paradigma: puedo fijar algunas variables y dejar otras libres (como incógnitas) para ver qué respuestas tengo.

¿Puede un predicado de un solo argumento ser inversible?
Sí, revisando la definición de inversibilidad, quiero tener la posibilidad de dejar el argumento unificado o no: tenemos un predicado *pais* poliádico, donde el primer individuo es el nombre del país y el segundo es la población (en millones de habitantes).
```
pais(argentina, 40).
pais(china, 1000).
pais(holanda, 12).
pais(eeuu, 120).
```

"Un país poblado es aquel país cuya población excede los 100.000.000 de habitantes"
```
pais_poblado(Pais):- pais(Pais, Poblacion), Poblacion > 100.
```

¿Puedo preguntar `pais_poblado(china)` y `pais_poblado(X)`?
Y… la única restricción parece ser la comparación `Poblacion > 100`. Para poder comparar `Poblacion` con `100` necesito tener unificada la variable `Poblacion`. Ok, antes yo unifico `pais(Pais, Poblacion)`, que es un hecho. Entonces `Poblacion` siempre va a tener valores ligados. Entonces ➔ el predicado es inversible.

Límites a la inversibilidad

Sin embargo así como definimos el predicado factorial no es inversible para el primer argumento
☹

¿Por qué? Si quiero hacer:
```
factorial(X, 6)
```

[1] Recordemos que en la consulta factorial(3, X), el primer argumento está instanciado mientras que el segundo no (es una incógnita)

Al hacer `N1 is N - 1` *(reemplazando variables:* `N1 is` **`X`** `- 1`*)* aparece una incógnita a la derecha del operador `is` (la variable `X` que en la consulta se envió sin unificar), entonces los argumentos no están suficientemente instanciados (ese es el mensaje que arroja SWI – PROLOG) y el predicado no se puede resolver.

Otra definición de factorial que admite inversibilidad en el primer argumento

```
factorial(N, FN):- factorial(N, FN, 0, 1).

factorial(N, FN, N, FN).
factorial(N, FN, I, Result):- I1 is I + 1,
                              Result1 is I1 * Result,
                              factorial(N, FN, I1, Result1).
```

Esto permite consultar tanto
`factorial(2, X)` como
`factorial(X, 6).`

Restricciones típicas para que los predicados no sean inversibles:

- Comparaciones (>, <, >=, <=, =)
- Operaciones aritméticas (+, -, *)
- Operador *is, y sobre todo*
- la **negación**, que es el gran problema de la programación lógica.

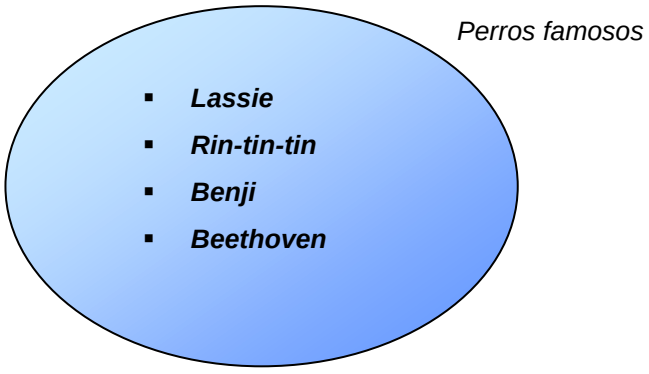
```
paisTranqui(X):- not(paisPoblado(X)).
?- paisTranqui(argentina)
Yes

?- paisTranqui(guineaEcuatorial)
Yes (Mmm… ¿cuándo puse yo la población de Guinea Ecuatorial?)

?- paisTranqui(X)
```

Me dice que no, ¿por qué? Porque el predicado **not** no es inversible. Not sólo funciona con predicados cuyas variables están unificadas.
`not(pais(X, 40))` funciona sólo si `X` está unificado a algún valor.

¿Por qué es así? Supongamos que yo tengo el Universo de los perros famosos que hicieron películas:



Y recordemos que todo lo que no está en la base de conocimientos no existe.

Entonces, ¿cómo se cuáles son los perros que no son famosos si no conozco los perros que forman parte de mi Universo?

No puedo pedirle al motor que me resuelva:
`perroComun(Perro):- not(perroFamoso(Perro)).`
si la variable Perro no está unificada.

Puedo preguntar `perroComun(lassie)` o `perroComun(sultan)`, pero no puedo preguntar `perroComun(X)`.
Por eso el predicado `perroComun` no es inversible.

¡Sólo sabe los perros famosos, no los que no son famosos!

Solución 1) `perroComun(Perro):- perro(Perro), not(perroFamoso(Perro)).`
O sea, si tengo el universo de perros y el universo de perros famosos, puedo determinar si un perro es común o famoso.

Solución 2) Agregar el complemento de los perros famosos y transformar la regla `perroComun` en un hecho. De esa manera transformo un predicado negativo en uno asertivo:

```
perroComun(chicho).
perroComun(sultan).
perroComun(copito).
```

La dificultad es que no siempre conozco el conjunto universal.

La inversibilidad de las cláusulas está relacionada con el concepto de declaratividad: si el predicado es inversible sólo tengo que encargarme que el motor haga su gracia. Grosso, ¿eh?

En las próximas clases vamos a seguir profundizando estas ideas.