

# TEMA V.

## Exemples de Llenguatges de Programació

21721-Llenguatges de Programació

# PROLOG

- Són les inicials de PROgramming in LOGic
- Introduït per Alain Colmerauer a principis dels 70
- Dissenyat pel processament d'informació simbòlica

# PROLOG

Programar en PROLOG consisteix en:

- Declarar un conjunt de fets coneguts
- Definir regles que descriguin les relacions entre els objectes
- Fer preguntes sobre els objectes i les seves relacions

# PROLOG

- Programar en PROLOG consisteix en dir-li el “**què**” és veritat i demanar-li que ho demostrï i tregui conclusions
- Als altres llenguatges es diu el què, “**com**” i quan ha de fer l'ordenador

# PROLOG

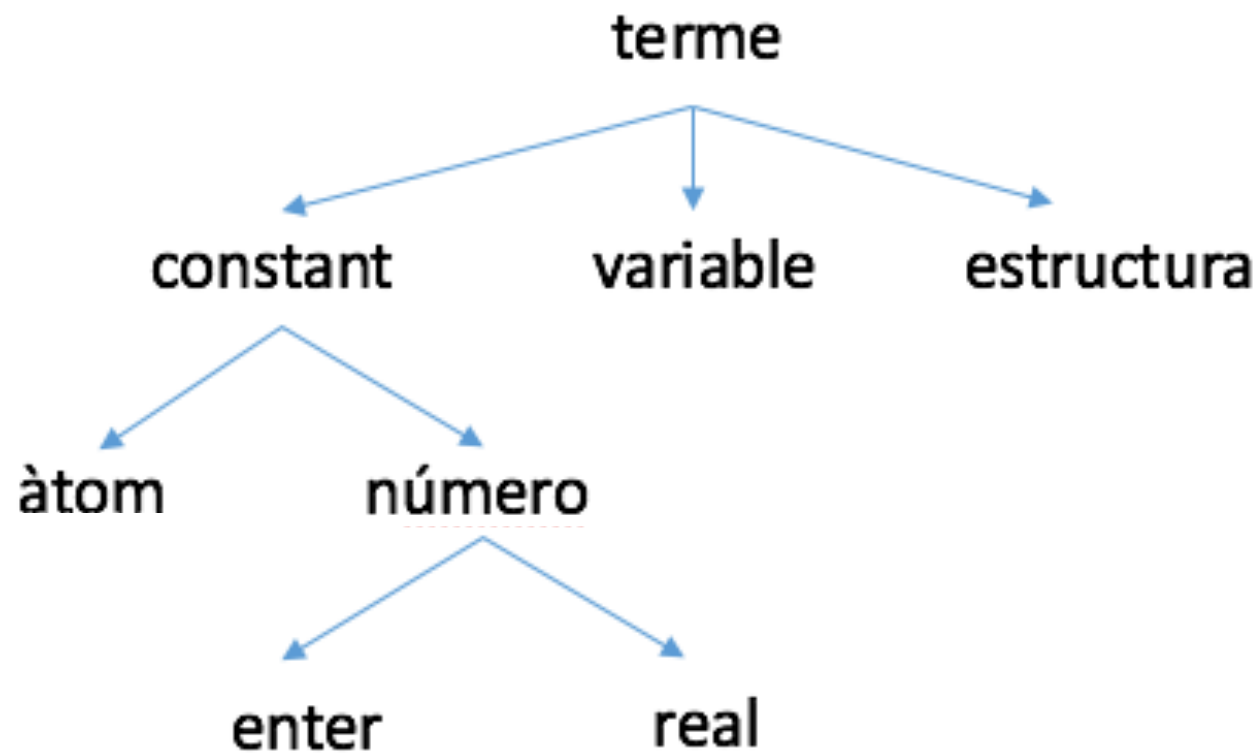
- Els coneixements es descriuen amb les clàusules de *Horn*, que es caracteritzen per tenir com a màxim un terme a la part de conclusió:

$$p \text{ :- } q_1, q_2, \dots, q_n.$$

- Que abans escrivíem com:

$$p_1, p_2, \dots, p_n \leftarrow q_1, q_2, \dots, q_n$$

# Els termes



# Conceptes bàsics: les constants

- S'utilitzen per donar nom als objectes concrets, a les seves relacions i a les seves propietats. Hi ha dos tipus de constants:
  - Els àtoms que són cadenes de lletres, dígit, guions i subratllats començant per **minúscula**. També podem tenir àtoms tancats entre cometes, en aquest cas no importa que comencin per minúscula
  - Numèrics: enters o reals

# Conceptes bàsics: els fets

- Serveixen per descriure propietats ja conegudes dels objectes i les seves relacions. Tant els noms dels objectes com de les relacions han de començar sempre en lletra minúscula
- Els fets es descriuen amb un nom de predicat i els objectes afectats com a arguments separats per comes i entre parèntesi
- **Acaben sempre en punt**



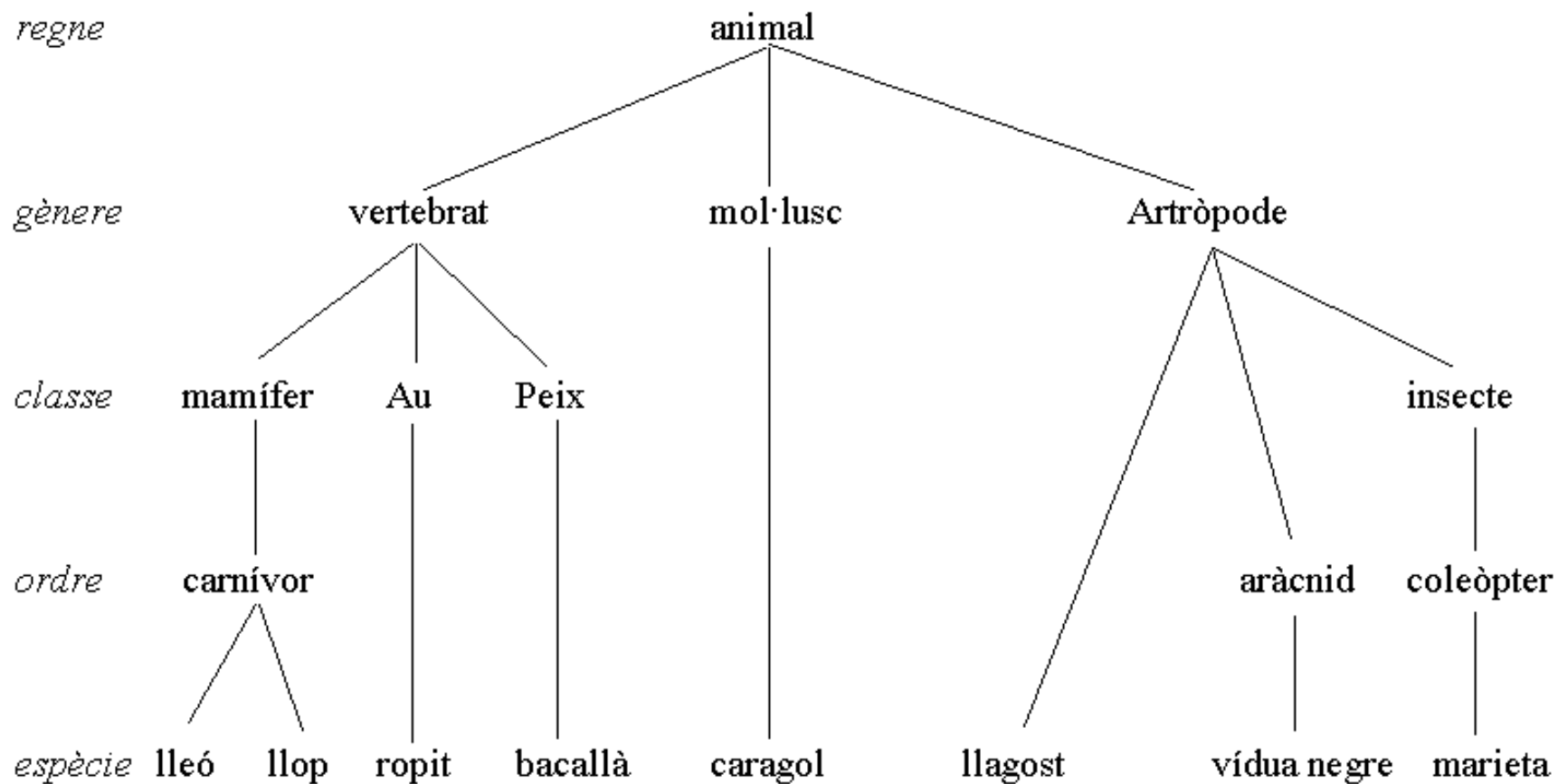
# Conceptes bàsics: els fets

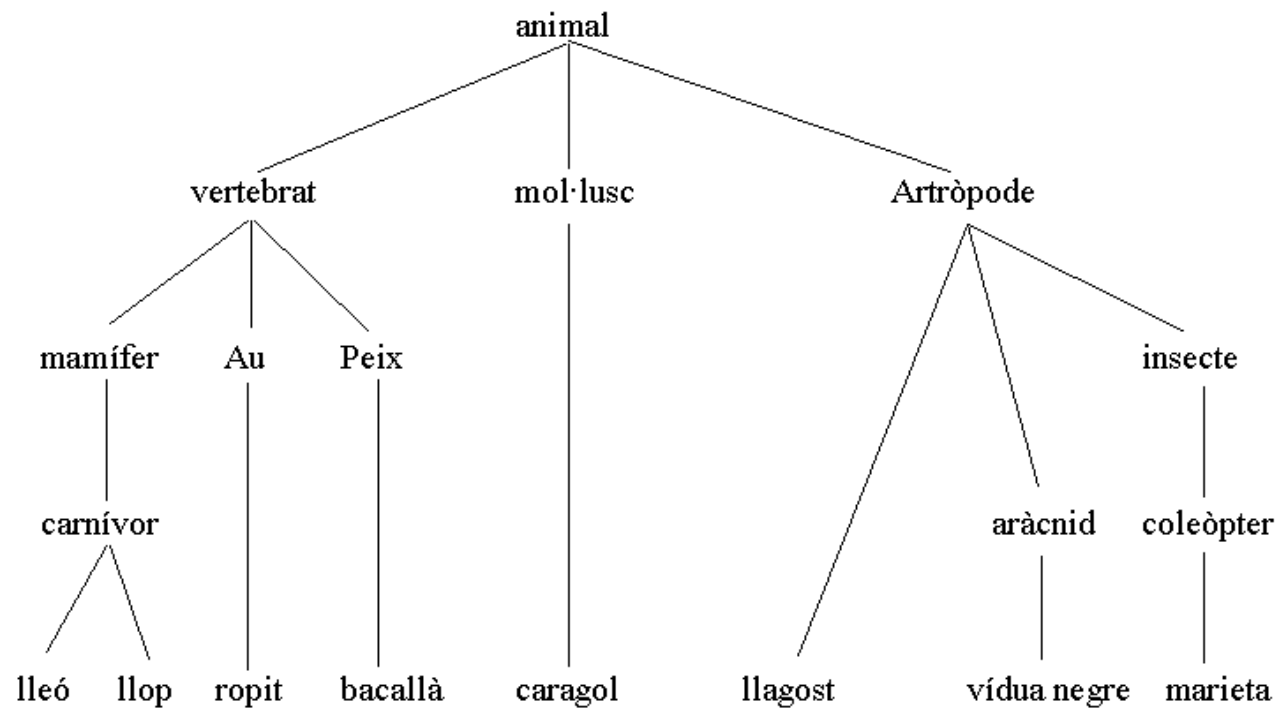
`nom_de_predicat(argument1, argument2...).`

ex:     `menja(ca, pa).`  
          descriu la relació “el ca menja pa”

Noltros establim la manera de llegir el predicat, també serem noltros els que haurem d'interpretar el resultat. No és el mateix el ca menja pa que el pa menja ca!

# Conceptes bàsics: els fets





és-un(marieta, coleopter).

és-un(coleòpter, insecte).

és-un(insecte, artropode).

és-un(artròpode, animal).

és-un(vidua-negre, aràcnid).

és-un(aràcnid, artròpode).

és-un(llagost, artròpode).

és-un(caragol, molusc).

és-un(molusc, animal).

és-un(bacalla, peix).

és-un(peix, vertebrat).

és-un(vertebrat, animal).

és-un(ropit, au).

és-un(au, vertebrat).

és-un(llop, carnívor).

és-un(lleó, carnívor).

és-un(carnívor, mamífer).

és-un(mamífer, vertebrat).

# Conceptes bàsics: estructures

- Una estructura és un objecte format per una col·lecció d'altres objectes anomenats components
- S'especifiquen amb un nom (functor) i separant les components per comes i entre parèntesi

`functor(component1, component2, ...).`

# Conceptes bàsics: estructures

- Ens poden servir per guardar la informació de forma ordenada:

```
llibre( titol(el-licenciat-vidriera),  
        autor(nom(miguel),cognom(cervantes)),  
        any(1613)).
```

# Conceptes bàsics: les preguntes

- PROLOG considera veritat tot el que té dins la base de coneixements i fals el que no hi és
- Una resposta afirmativa indica que ha demostrat el que demanàvem, una resposta negativa vol dir que amb les dades que té no ho pot demostrar. **No és una lògica bivaluada !!**
- PROLOG respon **true** o **false** depenent de si la resposta és veritat o falsa. També ens pot respondre **yes** o **no**, depèn de la versió de PROLOG amb la que treballem.

# Conceptes bàsics: les preguntes

?- és-un(coleòpter,insecte).

true

?- és-un(coleòpter,bitxo).

false

és-un(marieta, coleopter).	és-un(bacalla, peix).
és-un(coleòpter, insecte).	és-un(peix, vertebrat).
és-un(insecte, artropode).	és-un(vertebrat, animal).
és-un(artròpode, animal).	és-un(ropit, au).
és-un(vidua-negre, aràcnid).	és-un(au, vertebrat).
és-un(aràcnid, artròpode).	és-un(llop, carnívor).
és-un(llagost, artròpode).	és-un(lleó, carnívor).
és-un(caragol, molusc).	és-un(carnívor, mamífer).
és-un(molusc, animal).	és-un(mamífer, vertebrat).

# Conceptes bàsics: les variables

- S'utilitzen per representar objectes que inicialment són incògnites a les nostres preguntes
- Sempre comencen amb **majúscula** o subratllat “\_”
- Si no ens interessa la identitat d'un objecte, podem utilitzar la variable anònima “\_”
- **Dues variables anònimes dins la mateixa regla, son variables diferents**



# Conceptes bàsics: les variables

- Exemple:

```
?- és-un(coleòpter, _).
```

```
true
```

- demanam si coleòpter és alguna cosa, però no ens interessa el què

# Conceptes bàsics: les variables

- Si una variable representa un objecte, llavors es diu que està instanciada. Si no està definit el que representa, llavors es diu que no està instanciada (equivalent a tenir valor assignat o no)
- **Variables amb el mateix nom dins clàusules distintes són completament diferents**
- L'objectiu principal del PROLOG és instanciar totes les variables amb el que s'anomena procés d'UNIFICACIÓ

# El procés d'unificació

- La unificació és un mecanisme amb el que s'intenta que una hipòtesi plantejada tengui una forma "compatible" amb una de les clàusules de la base de coneixements
- Es cerca la base de coneixements (**d'amunt avall i d'esquerra a dreta**) cercant un predicat unificable amb la pregunta, si es troba llavors s'intenten unificar tots els arguments en ordre i valor

# El procés d'unificació

?- és-un(X, artròpode).

X=insecte; // el ; indica que es cerqui una

X=aràcnid; // altra solució

X=llagost;

false

?- és-un(X, artròpode).

X=insecte <return>

true

és-un(marieta, coleopter).	és-un(bacalla, peix).
és-un(coleòpter, insecte).	és-un(peix, vertebrat).
és-un(insecte, artropode).	és-un(vertebrat, animal).
és-un(artròpode, animal).	és-un(ropit, au).
és-un(vidua-negre, aràcnid).	és-un(au, vertebrat).
és-un(aràcnid, artròpode).	és-un(llop, carnívor).
és-un(llagost, artròpode).	és-un(lleó, carnívor).
és-un(caragol, molusc).	és-un(carnívor, mamífer).
és-un(molusc, animal).	és-un(mamífer, vertebrat).

# Regles d'unificació

- Dues variables no instanciades, unifiquen i es converteixen en la mateixa variable
- Dues variables instanciades, unifiquen únicament si tenen el mateix valor
- Una variable no instanciada i una instanciada unifiquen i la primera agafa el valor de la segona
- Els números i els àtoms unifiquen si són iguals
- Dues estructures unifiquen si tenen el mateix nom, el mateix número d'arguments i tots ells es poden unificar

# Regles d'unificació

?- X=lleó, X=Y.

X=lleó

Y=lleó

true

?- llop = lleó.

false

?- llop = llop.

true

és-un(marieta, coleopter).	és-un(bacalla, peix).
és-un(coleòpter, insecte).	és-un(peix, vertebrat).
és-un(insecte, artropode).	és-un(vertebrat, animal).
és-un(artròpode, animal).	és-un(ropit, au).
és-un(vidua-negre, aràcnid).	és-un(au, vertebrat).
és-un(aràcnid, artròpode).	és-un(llop, carnívor).
és-un(llagost, artròpode).	és-un(lleó, carnívor).
és-un(caragol, molusc).	és-un(carnívor, mamífer).
és-un(molusc, animal).	és-un(mamífer, vertebrat).

# Regles d'unificació

$$?- 4+2 = 6.$$

$$?-f(X,a)=f(a,X).$$

$$?-agrada(juana,X)=agrada(X,pere).$$

$$?-f(X,Y) = f(P,P)$$

# Conceptes bàsics: conjuncions

- La conjunció “i” es representa amb una coma “,” i ens permet combinar més d’un objectiu dins una mateixa pregunta:

?- és-un(lleó,X), és-un(au,Y).

X=carnívor

Y=vertebrat;

false

és-un(marieta, coleopter).	és-un(bacalla, peix).
és-un(coleòpter, insecte).	és-un(peix, vertebrat).
és-un(insecte, artropode).	és-un(vertebrat, animal).
és-un(artròpode, animal).	és-un(ropit, au).
és-un(vidua-negre, aràcnid).	és-un(au, vertebrat).
és-un(aràcnid, artròpode).	és-un(llop, carnívor).
és-un(llagost, artròpode).	és-un(lleó, carnívor).
és-un(caragol, molusc).	és-un(carnívor, mamífer).
és-un(molusc, animal).	és-un(mamífer, vertebrat).



# Conceptes bàsics: conjuncions

- Dins una conjunció, dues variables amb el mateix nom són la mateixa variable (estan dins la mateixa clàusula)

?- és-un(lleó,X), és-un(llop,X).

X=carnívor;

false

?- és-un(lleó,X), és-un(X,Y).

X=carnívor

Y=mamífer;

false

és-un(marieta, coleopter).	és-un(bacalla, peix).
és-un(coleòpter, insecte).	és-un(peix, vertebrat).
és-un(insecte, artropode).	és-un(vertebrat, animal).
és-un(artròpode, animal).	és-un(ropit, au).
és-un(vidua-negre, aràcnid).	és-un(au, vertebrat).
és-un(aràcnid, artròpode).	és-un(llop, carnívor).
és-un(llagost, artròpode).	és-un(lleó, carnívor).
és-un(caragol, molusc).	és-un(carnívor, mamífer).
és-un(molusc, animal).	és-un(mamífer, vertebrat).

# Conceptes bàsics: disjuncions

- La conjunció “o” es representa amb un punt i coma “;” i ens permet demanar si es verifica alguna d’entre varies preguntes:  
?- és-un(lleó,X); és-un(ropit,X).  
X=carnívor  
X=au;  
false
- És equivalent a demanar dues preguntes independents, X no és la mateixa variable  
?- és-un(lleó,X).  
X = carnívor  
  
?- és-un(ropit,X).  
X = au

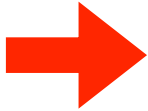
# Conceptes bàsics: negacions

- La negació ens permet fer preguntes sobre la falsetat d'un fet. S'utilitza el predicat "not"

?- not(és-un(lleó,carnívor)).

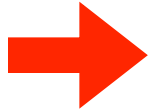
false

# Fins ara hem vist




- Constants: poden ser àtoms o números
  - Els àtoms són identificadors que han de començar en minúscula
- Fets: nom-predicat(llista d'arguments). /\* acaba en punt \*/
  - Els noms de predicats comencen per minúscules, arguments separats ","
- Estructures: functor(llista d'arguments).
  - Els arguments es poden definir recursivament amb altres estructures
- Preguntes: nom\_predicat(llista d'arguments).
  - Si la resposta és no, vol dir que no ho sap
- Variables: Sempre comencen per majúscula (instanciades o no)
- El procés d'unificació
- conjuncions (i): representades per una coma (,)
- disjuncions (o): representades per un punt i coma(;) o varies regles
- negacions (no)

# Fins ara hem vist

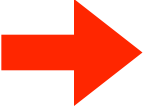


- Constants: poden ser àtoms o números
  - Els àtoms són identificadors que han de començar en minúscula
- Fets: nom-predicat(llista d'arguments). /\* acaba en punt \*/
  - Els noms de predicats comencen per minúscules, arguments separats ","
- Estructures: functor(llista d'arguments).
  - Els arguments es poden definir recursivament amb altres estructures
- Preguntes: nom\_predicat(llista d'arguments).
  - Si la resposta és no, vol dir que no ho sap
- Variables: Sempre comencen per majúscula (instanciades o no)
- El procés d'unificació
- conjuncions (i): representades per una coma (,)
- disjuncions (o): representades per un punt i coma(;) o varies regles
- negacions (no)

# Fins ara hem vist

- Constants: poden ser àtoms o números
  - Els àtoms són identificadors que han de començar en minúscula
- Fets: nom-predicat(llista d'arguments). /\* acaba en punt \*/
  - Els noms de predicats comencen per minúscules, arguments separats ","
-  • Estructures: functor(llista d'arguments).
  - Els arguments es poden definir recursivament amb altres estructures
- Preguntes: nom\_predicat(llista d'arguments).
  - Si la resposta és no, vol dir que no ho sap
- Variables: Sempre comencen per majúscula (instanciades o no)
- El procés d'unificació
- conjuncions (i): representades per una coma (,)
- disjuncions (o): representades per un punt i coma(;) o varies regles
- negacions (no)

# Fins ara hem vist

- Constants: poden ser àtoms o números
  - Els àtoms són identificadors que han de començar en minúscula
- Fets: nom-predicat(llista d'arguments). /\* acaba en punt \*/
  - Els noms de predicats comencen per minúscules, arguments separats ","
- Estructures: functor(llista d'arguments).
  - Els arguments es poden definir recursivament amb altres estructures
-  • Preguntes: nom\_predicat(llista d'arguments).
  - Si la resposta és no, vol dir que no ho sap
- Variables: Sempre comencen per majúscula (instanciades o no)
- El procés d'unificació
- conjuncions (i): representades per una coma (,)
- disjuncions (o): representades per un punt i coma(;) o varies regles
- negacions (no)

# Fins ara hem vist

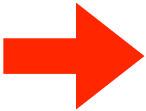
- Constants: poden ser àtoms o números
  - Els àtoms són identificadors que han de començar en minúscula
- Fets: nom-predicat(llista d'arguments). /\* acaba en punt \*/
  - Els noms de predicats comencen per minúscules, arguments separats ","
- Estructures: functor(llista d'arguments).
  - Els arguments es poden definir recursivament amb altres estructures
- Preguntes: nom\_predicat(llista d'arguments).
  - Si la resposta és no, vol dir que no ho sap
- Variables: Sempre comencen per majúscula (instanciades o no)
- El procés d'unificació
- conjuncions (i): representades per una coma (,)
- disjuncions (o): representades per un punt i coma(;) o varies regles
- negacions (no)





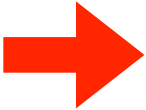
# Fins ara hem vist

- Constants: poden ser àtoms o números
  - Els àtoms són identificadors que han de començar en minúscula
- Fets: nom-predicat(llista d'arguments). /\* acaba en punt \*/
  - Els noms de predicats comencen per minúscules, arguments separats ","
- Estructures: functor(llista d'arguments).
  - Els arguments es poden definir recursivament amb altres estructures
- Preguntes: nom\_predicat(llista d'arguments).
  - Si la resposta és no, vol dir que no ho sap
- Variables: Sempre comencen per majúscula (instanciades o no)
- El procés d'unificació
- conjuncions (i): representades per una coma (,)
- disjuncions (o): representades per un punt i coma(;) o varies regles
- negacions (no)



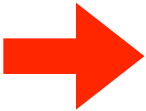
# Fins ara hem vist

- Constants: poden ser àtoms o números
  - Els àtoms són identificadors que han de començar en minúscula
- Fets: nom-predicat(llista d'arguments). /\* acaba en punt \*/
  - Els noms de predicats comencen per minúscules, arguments separats ","
- Estructures: functor(llista d'arguments).
  - Els arguments es poden definir recursivament amb altres estructures
- Preguntes: nom\_predicat(llista d'arguments).
  - Si la resposta és no, vol dir que no ho sap
- Variables: Sempre comencen per majúscula (instanciades o no)
- El procés d'unificació
- conjuncions (i): representades per una coma (,)
- disjuncions (o): representades per un punt i coma(;) o varies regles
- negacions (no)



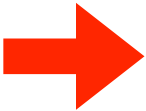
# Fins ara hem vist

- Constants: poden ser àtoms o números
  - Els àtoms són identificadors que han de començar en minúscula
- Fets: nom-predicat(llista d'arguments). /\* acaba en punt \*/
  - Els noms de predicats comencen per minúscules, arguments separats ","
- Estructures: functor(llista d'arguments).
  - Els arguments es poden definir recursivament amb altres estructures
- Preguntes: nom\_predicat(llista d'arguments).
  - Si la resposta és no, vol dir que no ho sap
- Variables: Sempre comencen per majúscula (instanciades o no)
- El procés d'unificació
- conjuncions (i): representades per una coma (,)
- disjuncions (o): representades per un punt i coma(;) o varies regles
- negacions (no)



# Fins ara hem vist

- Constants: poden ser àtoms o números
  - Els àtoms són identificadors que han de començar en minúscula
- Fets: nom-predicat(llista d'arguments). /\* acaba en punt \*/
  - Els noms de predicats comencen per minúscules, arguments separats ","
- Estructures: functor(llista d'arguments).
  - Els arguments es poden definir recursivament amb altres estructures
- Preguntes: nom\_predicat(llista d'arguments).
  - Si la resposta és no, vol dir que no ho sap
- Variables: Sempre comencen per majúscula (instanciades o no)
- El procés d'unificació
- conjuncions (i): representades per una coma (,)
- disjuncions (o): representades per un punt i coma(;) o varies regles
- negacions (no)



## L'interprète de PROLOG

- SWI-PROLOG
  - Versión Linux, Windows i MacOS

Instal·lació de swi-prolog (windows, macOS i linux)

<http://www.swi-prolog.org/download/stable>

# Comparació: palíndroms

- Un palíndrom és una paraula o frase que es llegeix igual tan a l'endret com al revés
- Exemple:  
    "no sap pas on"

# Paradigma imperatiu

```
funció esPalíndrom(frase f) retorna booleà
    n:enter;
inici
    n := f.longitud;
    per i des de 1 fins a n/2 fer
        si f[i] != f[n-i+1] retorna fals;
    fper
    retorna veritat;
fi
```

<b>i</b>	<b>p[i]</b>	<b>p[n-i+1]</b>	<b>n</b>	<b>n/2</b>	<b>retorna</b>
1	n	n	13	6	veritat
2	o	o			
3					
4	s	s			
5	a	a			
6	p	p			



## Paradigma lògic

`palíndrom(F) ← invertir(F, F)`



# Conceptes bàsics: regles d'inferència

Les regles s'utilitzen si es vol establir una dependència entre fets. Tenen la forma:

cap :- cos.

on el cap especifica el fet que es vol descriure i el cos són les condicions que s'han de verificar perquè el cap sigui cert

# Conceptes bàsics: regles d'inferència

Fets:

fill(pep, pere).

fill(lluis, pere).

en pep és fill de'n Pere

# Conceptes bàsics: regles d'inferència

Fets:

fill(pep, pere).                      en pep és fill de'n Pere  
fill(lluis, pere).

Regles d'inferència:

pare(X,Y):-fill(Y,X).      X és pare de Y si Y és fill de X  
germà(X,Y):-pare(Z,X),pare(Z,Y).

# Conceptes bàsics: regles d'inferència

Fets:

fill(pep, pere).

fill(lluis, pere).

Regles d'inferència:

pare(X,Y):-fill(Y,X).

germà(X,Y):-pare(Z,X),pare(Z,Y).

Preguntes:

?- germà(pep,lluis).

yes

# Operadors

@<, @>, @=<, @>= comparen àtoms simbòlics

Aritmètics		Lògics	
Símbol	Operació	Símbol	Operació
+	Suma	=	Igualtat
-	Resta	==	Igualtat
*	Multiplicació	\=	Desigualtat
/	Divisió	=\=	Desigualtat
//	Divisió entera	=<	Menor o igual
mod	Mòdul	>=	Major o igual
rem	Residu	<	Menor
^	Exponencial	>	Major
		is	Operador d'avaluació

Operació	Resultat
$A=3+4$	$A=3+4$
$A \text{ is } 3+4$	$A=7$
$7 \text{ is } 3+4$	true
$2+4=3+3$	false
$2+4 \text{ is } 3+3$	false
$2+4:=3+3$	true
$2+4\backslash=3+3$	true
$2+4=\backslash=3+3$	false

- Els operadors relacionals  $=$  i  $\backslash=$  comparen símbols.
- Els operadors  $:=$  i  $=\backslash=$  comparen valors (avaluen els dos costats abans de comparar).
- L'operador  $\text{is}$  avalua només la part de la dreta i la unifica amb la de l'esquerra.



# Operadors

què respon PROLOG si en l'exemple anterior demanem:

?- germà(X,Y).

# Operadors

què respon PROLOG si en l'exemple anterior demanem:

?- germà(X,Y).

X = Y, Y = pep ;

X = pep,

Y = lluis ;

X = lluis,

Y = pep ;

X = Y, Y = lluis.

# Operadors

què respon PROLOG si en l'exemple anterior demanem:

?- germà(X,Y).

Com es pot arreglar?

germà(X, Y) :- pare(Z, X), pare(Z, Y), X \=Y.

# Operadors

?- germà(X,Y).

X = pep,

Y = lluis ;

X = lluis,

Y = pep ;

**false.**

# Exemples d'unificació amb “=”

?-  $4+2 = 6$ .

false

?-  $4+2$  is 6.

false

?- 6 is  $4+2$ .

true

# Exemple

població(palma, 300000).

població(manacor, 50000).

població(santa-maria, 4000).

superfície(palma, 10).

superfície(manacor, 6).

superfície(santa-maria, 2).

# Exemple

població(palma, 300000).

població(manacor, 50000).

població(santa-maria, 4000).

superfície(palma, 10).

superfície(manacor, 6).

superfície(santa-maria, 2).

densitat(X,Y):- població(X, P), superfície(X,S), Y is P/S.

# Exemple

horòscop(àries,21,3,21,4).

horòscop(toro,21,4,21,5).

horòscop(geminis,21,5,21,6).

horòscop(càncer,21,6,21,7).

horòscop(leo,21,7,21,8).

horòscop(virgo,21,8,21,9).

horòscop(libra,21,9,21,10).

horòscop(escorpí,21,10,21,11).

horòscop(sagitari,21,11,21,12).

horòscop(capricorn,21,12,21,1).

horòscop(aquari,21,1,21,2).

horòscop(peixos,21,2,21,3).



# Exemple

signe(Dia, Mes, Signe):-

    horòscop(Signe,D1,M1,D2,M2),

    Mes=M1, Dia>=D1.

signe(Dia, Mes, Signe):-

    horòscop(Signe,D1,M1, D2, M2),

    Mes=M2, Dia<D2.

# Recursivitat

Calcular el factorial d'un número:

- el factorial de 0 és 1
- el factorial de N és el  $N * \text{factorial}(N-1)$

# Recursivitat

% El factorial de 0 és 1

**factorial(0,1).**

% El factorial de N és  $N * \text{factorial}(N-1)$

**factorial(N,X):-    N1 is N-1,  
                      factorial(N1,X1),  
                      X is N\*X1.**

# Recursivitat

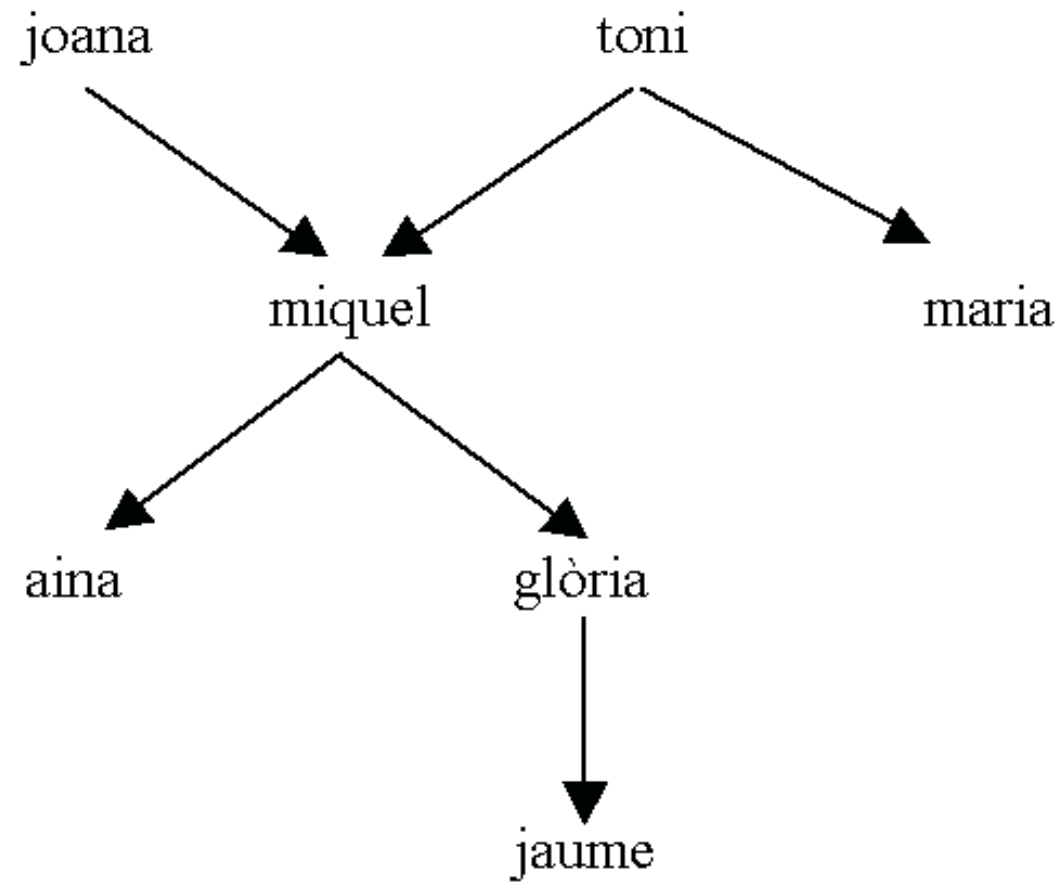
% El factorial de 0 és 1

**factorial(0,1).**

% El factorial de N és  $N \cdot \text{factorial}(N-1)$

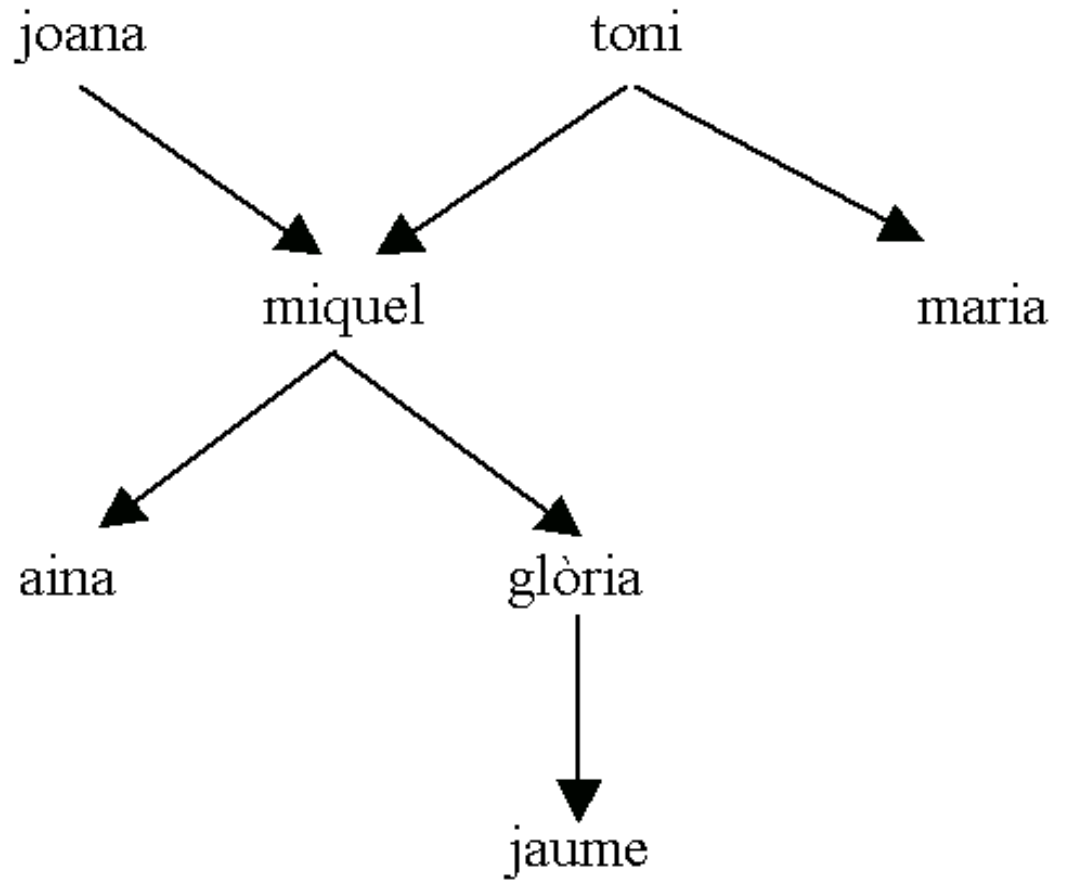
**factorial(N,X):-**    **N>0,**        **/\* no pot ser negatiu \*/**  
                      **N1 is N-1,**  
                      **factorial(N1,X1),**  
                      **X is N\*X1.**

# Recursivitat



# Recursivitat

ascendent(joana, miquel).  
ascendent(toni, miquel).  
ascendent(toni, maria).  
ascendent(miquel, aina).  
ascendent(miquel, glòria).  
ascendent(glòria, jaume).



# Antecessor

Volem saber si una persona és antecessor (a qualsevol nivell) d'una altra

# Antecessor

Volem saber si una persona és antecessor (a qualsevol nivell) d'una altra

**R1. antecessor(X,Z):- ascendent(X,Z).**



# Antecessor

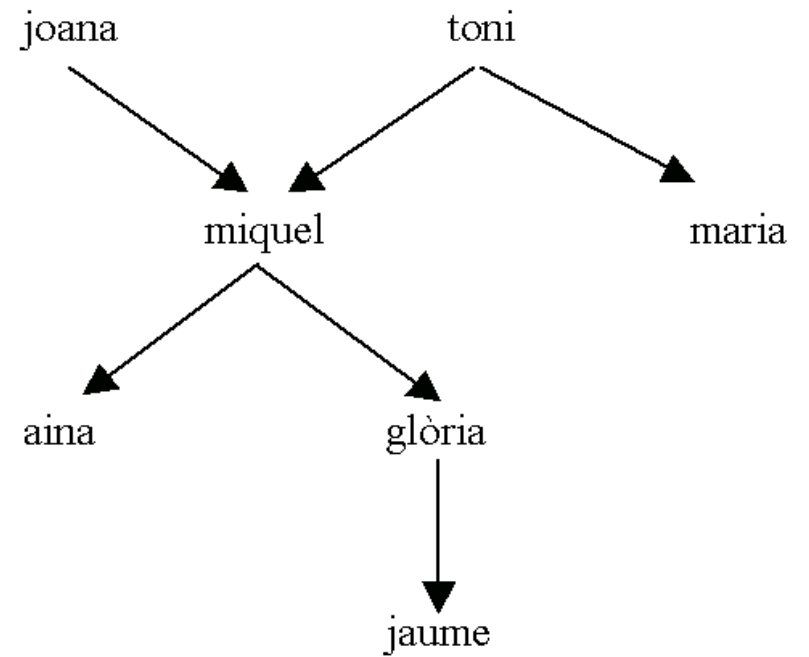
Volem saber si una persona és antecessor (a qualsevol nivell) d'una altra

**R1. antecessor(X,Z):-      ascendent(X,Z).**

**R2. antecessor(X,Z):-      ascendent(X,Y),  
                                 antecessor(Y,Z).**

?- antecessor(toni,gloria).

R1. X=toni, Z=gloria  
ascendent(toni,gloria).  
no



?- antecessor(toni,gloria).

R1. X=toni, Z=gloria

ascendent(toni,gloria).

no

R2. ascendent(toni,Y), antecessor(Y,gloria).

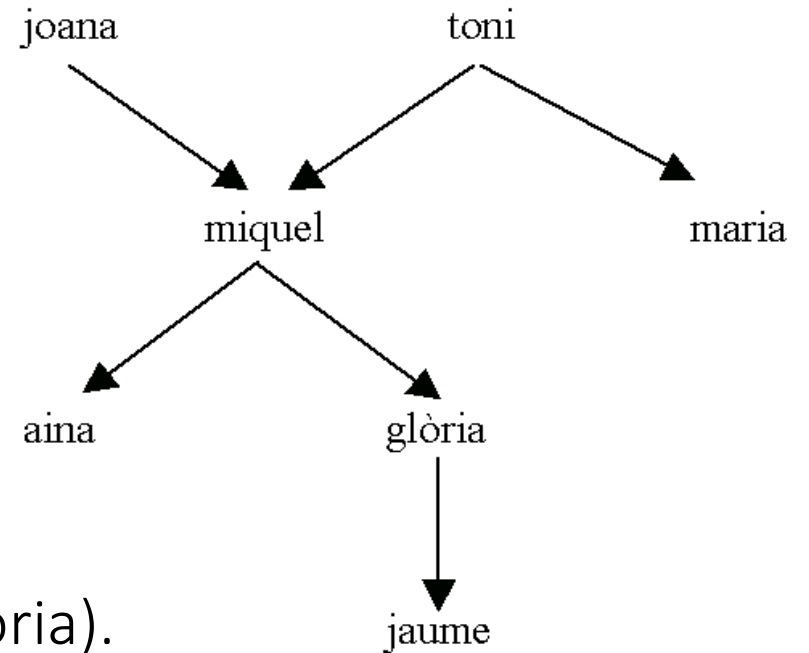
ascendent(toni,Y)

Y=miquel

antecessor(miquel,gloria).

R1. ascendent(miquel,gloria).

yes



## *Backtracking* o reavaluació

- És juntament amb la unificació, un dels mecanismes fonamentals de PROLOG
- Consisteix en, una vegada una hipòtesi no s'ha pogut demostrar, intentar demostrar-la seguint un altre camí

## *Backtracking* o reavaluació

- PROLOG intenta verificar o satisfer un objectiu cercant a la base de coneixements des del principi
- Si troba un fet o un cap d'una regla que es pugui unificar, fa una marca en aquest punt i instància totes les variables necessàries
- Si és una regla el que ha trobat, llavors també intentarà satisfer els nous objectius (subobjectius)
- Si no troba cap fet o cap de regla unificable, l'objectiu falla i el procés de *backtracking* comença intentant resatisfer l'objectiu inicial

## Backtracking o reavaluació

Provem a canviar l'ordre de les clàusules:

**R2. antecessor(X,Z):- ascendent(X,Y),  
antecessor(Y,Z).**

**R1. antecessor(X,Z):- ascendent(X,Z).**

?- antecessor(toni, gloria).

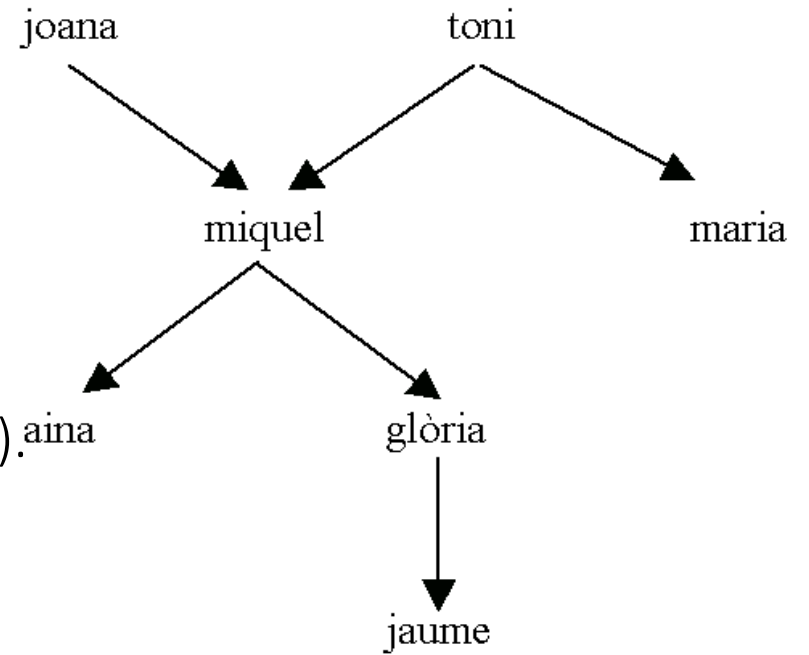
R2: X=toni, Z=gloria

ascendent(toni, Y), antecessor(Y, gloria).

ascendent(toni, Y)

Y=miquel

antecessor(miquel, gloria)



antecessor(miquel, gloria)

R2: X=miquel, Z=gloria

ascendent(miquel, Y), antecessor(Y, gloria).

ascendent(miquel, Y)

Y=aina

antecessor(aina, gloria)

R2: X=aina, Z=gloria

ascendent(aina, Y), antecessor(Y, gloria)

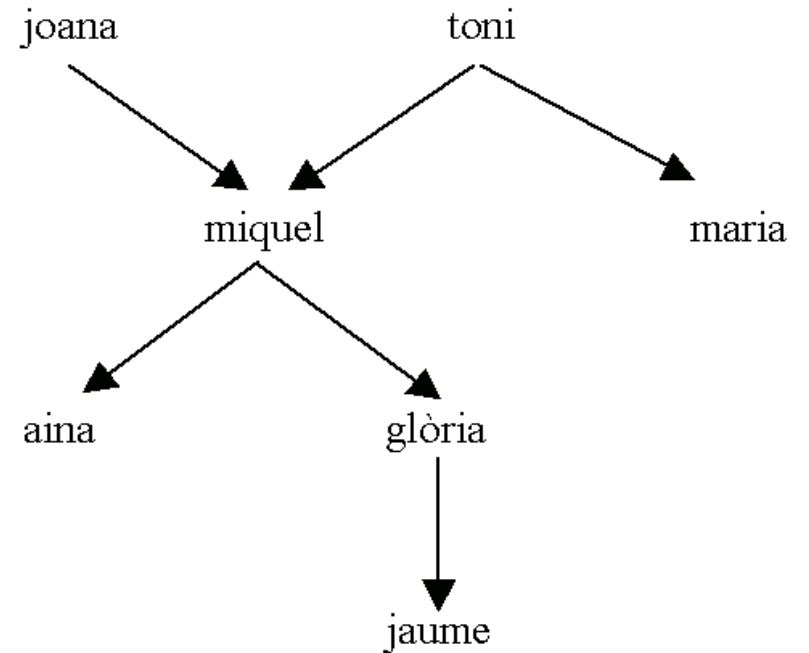
ascendent(aina, Y)

no

R1: X=aina, Z=gloria

ascendent(aina, gloria)

no





antecessor(miquel, gloria)

Y=gloria

antecessor(gloria, gloria)

R2: X=gloria, Z=gloria

ascendent(gloria, Y), antecessor(Y, gloria)

ascendent(gloria, Y)

Y=jaume

antecessor(jaume, gloria)

R2: ascendent(jaume, Y), antecessor(Y, gloria).

no

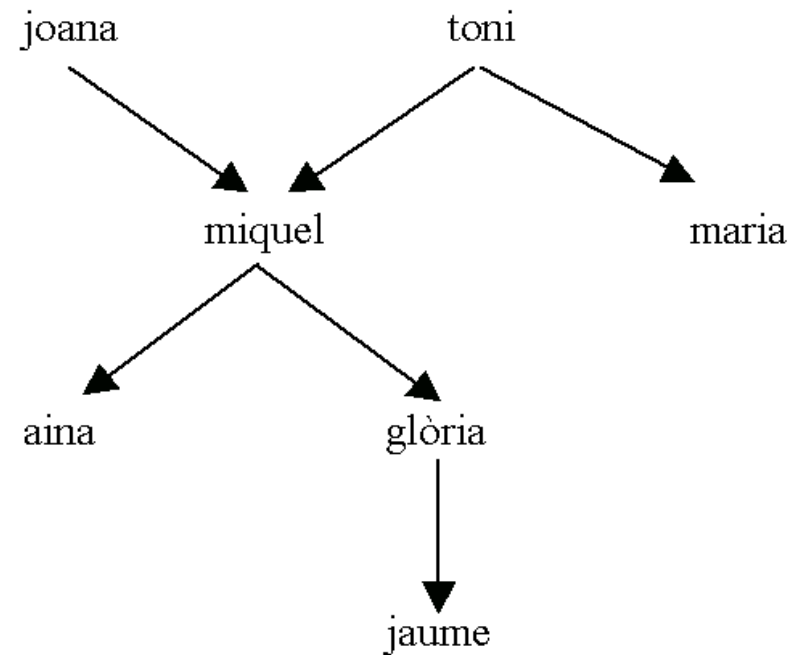
R1: ascendent(jaume, gloria)

no

R1: X=gloria, Z=gloria

no

no

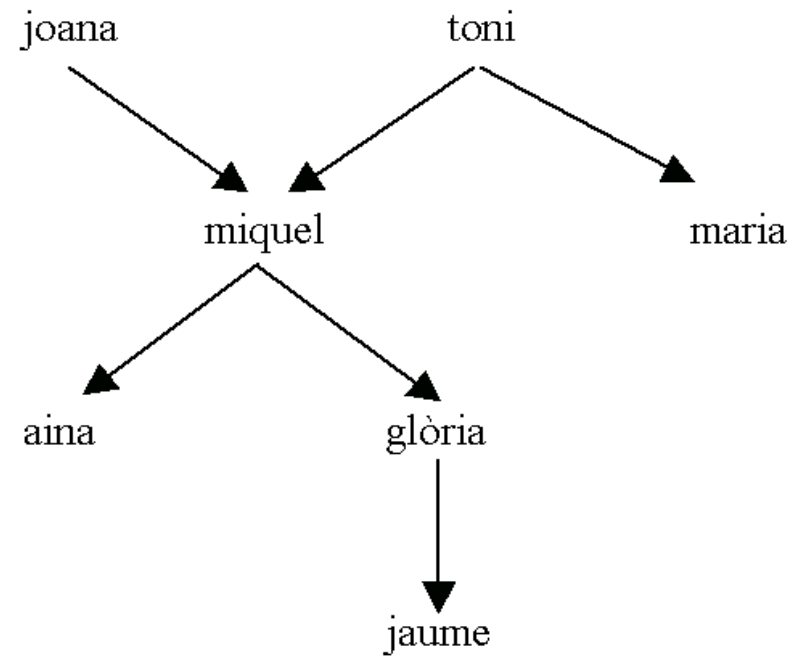


antecessor(miquel,gloria)

R1: X=miquel, Z=gloria

ascendent(miquel, gloria)

si



i si canviem l'ordre dins el cos

R1. antecessor(X,Z):-ascendent(X,Z).

R2. antecessor(X,Z):-antecessor(X,Y), ascendent(Y,Z).

antecessor(toni, gloria).

R1: X=toni, Z=gloria.

ascendent(toni, gloria).

no

R2: antecessor(toni, Y), ascendent(Y, gloria).

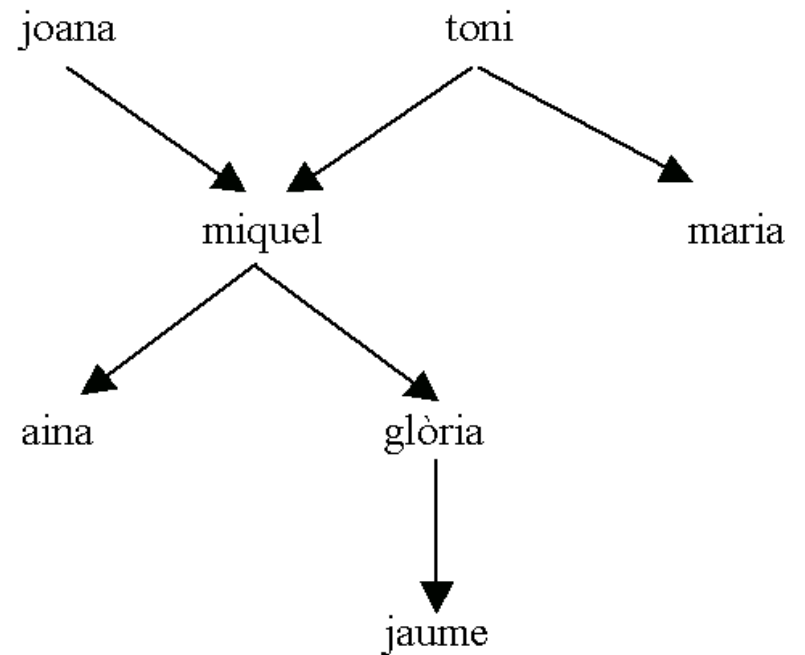
antecessor(toni,Y)

R1: ascendent(toni, Y)

Y=miquel

ascendent(miquel, gloria)

si



i ara canviam l'ordre de R1 i R2

R2. antecessor(X,Z):-antecessor(X,Y), ascendent(Y,Z).

R1. antecessor(X,Z):-ascendent(X,Z).

antecessor(toni, gloria).

R2: antecessor(toni, Y), ascendent(Y, gloria).

antecessor(toni,Y)

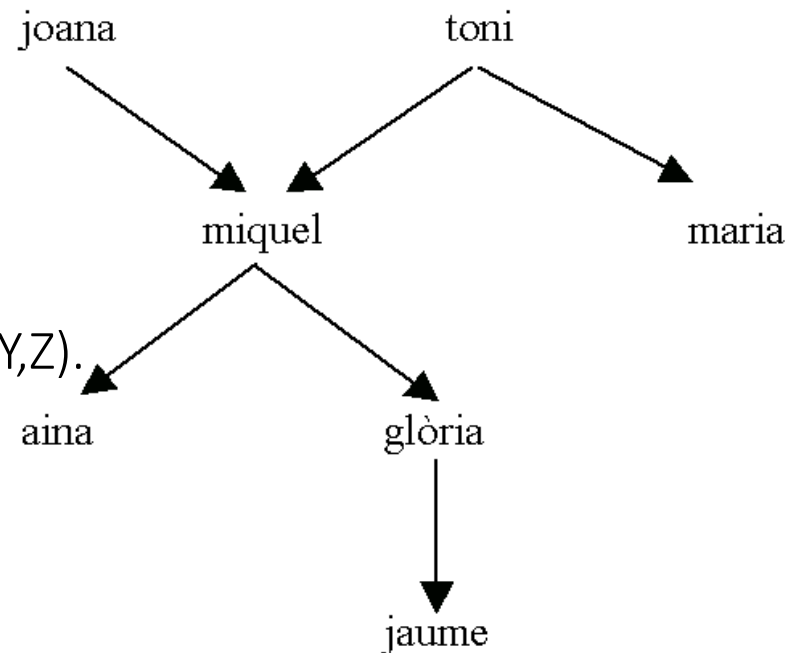
R2: antecessor(toni, Y'), ascendent(Y', Y).

antecessor(toni,Y')

R2: antecessor(toni, Y''), ascendent(Y'', Y').

antecessor(toni,Y'')

...



# Conclusió

És important:

- Posar els casos bàsics al principi!
- Evitar la recursivitat per l'esquerra!

# Llistes

- Una llista en PROLOG és una seqüència ordenada d'elements tancats entre claudàtors i separats per comes, de qualsevol longitud. Els elements poden ser termes o altres llistes
- Exemples:
  - [a,e,i,o,u]
  - [mamífer(llop), aràcnid(vídua-negre), au(ropit)]
  - [[]]

# Llistes

- La creació i la descomposició de llistes es fa utilitzant el procés d'unificació i l'operador “|” que separa una llista amb el seu cap i la seva cua

?-  $[X|Y] = [a,e,i,o,u]$ .

$X=a$

$Y=[e,i,o,u]$

?-  $L=[e,i,o,u], X=[a|L]$

$X=[a,e,i,o,u]$

# Unificació de llistes

$[a,b,c] = [c,b,a]$

→ distint ordre

$[X] = [a,b,c]$

→ distinta longitud

$[X|L] = [a,b,c]$

→  $X=a, L=[b,c]$

$[X,Y,Z|L] = [a,e,i,o,u]$

→  $X=a, Y=e, Z=i, L=[o,u]$

$[X|Y] = []$

→ no !

$[X|Y] = [[a,[b,c]],d]$

→  $X=[a,[b,c]], Y=[d]$

$[X|Y] = [a]$

→  $X=a, Y=[]$



# Recursivitat

Comptar el número d'elements d'una llista:

- el número d'elements d'una llista buida és 0
- el número d'elements d'una llista no buida és sumar 1 al número d'elements de la coa

# Recursivitat

Comptar el número d'elements d'una llista:

```
elements([],0).
```

```
elements([X|L],N):-elements(L,N1), N is N1+1.
```

## Exercici: Predicats amb llistes

- Pertany un element a una llista
- Afegir dues llistes
- Trobar el darrer element d'una llista
- Invertir una llista
- Esborrar un element d'una llista
- Sumar els elements d'una llista
- Trobar el màxim element d'una llista
- Permutar els elements d'una llista

# Pertany un element a una llista ?

```
?-pertany(a, [b,a,c]).
```

```
true
```

```
?-pertany(a, [b,d,c]).
```

```
false
```

# Pertany un element a una llista ?

```
pertany(X,[X | L]).
```

```
pertany(X,[Y | L]):-pertany(X,L).
```

# Pertany un element a una llista ?

```
pertany(X,[X|L]).  
pertany(X,[Y|L]):-pertany(X,L).
```

**fixau-vos** com es pot utilitzar la variable anònima:

```
pertany(X,[X|_]).  
pertany(X,[ _ |L]):-pertany(X,L).
```

# Pertany X a la llista L ?

?-pertany(2, [1,2,3]).

yes

R1. pertany(X,[X|L]).

R2. pertany(X,[Y|L]):-pertany(X,L).

R1:  $X=2$ ,  $X=1$ ,  $Y=[2,3]$  no !

R2:  $X=2$ ,  $Y=1$ ,  $L=[2,3]$  → pertany(2,[2,3]).

R1:  $X=2$ ,  $X=2$ ,  $Y=[3]$  si !

Que passa si demanam...

?-pertany(X, [1,2,3]).



?-pertany(X, [1,2,3]).

R1. pertany(X,[X|L]).

R2. pertany(X,[Y|L]):-pertany(X,L).

R1:  $X'=X$ ,  $X'=1$ ,  $L=[2,3] \rightarrow$  escriu  $X=1$  ;

R2:  $X'=X$ ,  $Y=1$ ,  $L=[2,3] \rightarrow$  pertany( $X'$ ,[2,3]).

R1:  $X''=X'$ ,  $X''=2$ ,  $L=[3] \rightarrow$  escriu  $X=2$  ;

R2:  $X''=X'$ ,  $Y=2$ ,  $L=[3] \rightarrow$  pertany( $X''$ ,[3]).

R1:  $X'''=X''$ ,  $X'''=3$ ,  $L=[] \rightarrow$  escriu  $X=3$  ;

R2:  $X'''=X''$ ,  $Y=3$ ,  $L=[] \rightarrow$  pertany( $X'''$ ,[]).

no

# Que passa si demanam...

?-pertany(X, [1,2,3]).

X=1;

X=2;

X=3;

no

# Afegir dues llistes

?- afegir([a,b],[c,d],L).

L=[a,b,c,d]

# Afegir dues llistes

```
afegir([],L,L).
```

```
afegir([X|L1],L2,[X|L3]):-afegir(L1,L2,L3).
```

```
?- afegir([a,b],[c,d],L).
```

```
L=[a,b,c,d] ;
```

```
no
```

# Afegir dues llistes

R1. `afegir([],L,L).`

R1. `afegir([X|L1],L2,[X|L3]):-afegir(L1,L2,L3).`

?- `afegir([a,b],[c,d],L).`

R1: no unifica `[]=[a,b]`

R2: `[a,b]=[X|L1], [c,d]=L2, L=[X|L3]`

`X=a, L1=[b], L2=[c,d], L=[a|L3]`

`afegir([b],[c,d],L3).`

R1: no unifica `[]=[b]`

R2: `[b]=[X'|L1'], [c,d]=L2', L3=[X|L3']`

`X'=b, L1'=[], L2'=[c,d], L3=[b|L3']`

`afegir([], [c,d], L3')`

R1: `[]=[], L=[c,d], L3'=L → L3'=[c,d]`

`L3=[b|L3']=[b|[c,d]]=[b,c,d]`

`L=[a|L3]=[a|[b,c,d]]=[a,b,c,d]`

Que passa si demanam...

?-afegir(L1, L2, [a,b,c]).

?- afegir(L1, L2, [a,b,c]).

R1. afegir([],L,L).

R2. afegir([X|L1],L2,[X|L3]):-afegir(L1,L2,L3).

R1: []=L1, L=L2, L=[a,b,c]  $\rightarrow$  **escriu L1=[], L2=[a,b,c];**

R2: [X|L1']=L1, L2'=L2, [X|L3']=[a,b,c]

X=a, L3'=[b,c]  $\rightarrow$  afegir(L1',L2',[b,c])

R1: []=L1',L=L2', L=[b,c]  $\rightarrow$  L1'=[], L2'=[b,c]

L1=[a|L1']=[a], L2=L2'=[b,c]  $\rightarrow$  **escriu L1=[a], L2=[b,c];**

R2:[X|L1'']=L1', L2''=L2', [X|L3'']=[b,c]  $\rightarrow$  X=b,L3''=[c]

afegir(L1'',L2'',[c])

R1: []=L1'',L=L2'', L=[c]  $\rightarrow$  L1''=[], L2''=[c]

L1'=[b|L1'']= [b], L2'=L2''=[c]

L1=[a|L1']=[a,b], L2=L2'= [c]  $\rightarrow$  **escriu L1=[a,b], L2=[c];**

R2: ...

# Que passa si demanam...

?-afegir(L1, L2, [a,b,c]).

L1=[]

L2=[a,b,c];

L1=[a]

L2=[b,c];

L1=[a,b]

L2=[c];

L1=[a,b,c]

L2=[];

no



# Trobar el darrer element

?-darrer([a,b,c,d],X).

X=d

# Trobar el darrer element

```
darrer([X],X).
```

```
darrer([X|L],Y):-darrer(L,Y).
```

# Operacions de depuració

?- trace.

?- notrace.

El debugger ens dóna la següent informació:

- call: és una nova cridada a un predicat.
- exit: és quan torna satisfactòriament.
- redo: intenta un altra camí.
- fail: no troba cap solució

# Invertir una llista

?-invertir([a,b,c,d],X).

X=[d,c,b,a]

# Invertir una llista

`invertir([X],[X]).`

`invertir([X|L1],L2):-invertir(L1,L3), afegir(L3,[X],L2).`

# Esborrar un element d'una llista

?-esborrar(a,[b,a,c],L).

L=[b,c]

# Esborrar un element d'una llista

```
esborrar(_,[],[]).
```

```
esborrar(X,[X|L],L).
```

```
esborrar(X,[Y|L1],[Y|L2]):-esborrar(X,L1,L2).
```

# Sumar els elements d'una llista

?-sumar([3,1,5,9,4],N).

N=22



# Sumar els elements d'una llista

`sumar([],0).`

`sumar([X|L],N):-sumar(L,N1),N is N1+X.`

# Màxim d'una llista

?-màxim([1,6,3,9,4],X).

X=9

# Màxim d'una llista

```
màxim([X],X).  
màxim([X,Y | L],Z):-X>Y,màxim([X | L],Z).  
màxim([_ | L],Z):-màxim(L,Z).
```

# Permutacions d'una llista

?-permutació([a,b,c],L).

L = [a, b, c] ;

L = [b, a, c] ;

L = [b, c, a] ;

L = [a, c, b] ;

L = [c, a, b] ;

L = [c, b, a] ;

false.

## Permutacions d'una llista

```
permutació([],[]).  
permutació([X|Y],Z):-permutació(Y,L),  
                        inserir(X,L,Z).
```

```
inserir(E,L,[E|L]).  
inserir(E,[X|Y],[X|Z]):-inserir(E,Y,Z).
```

# Permutacions d'una llista

?- inserir(a,[1,2,3],L).

L = [a, 1, 2, 3] ;

L = [1, a, 2, 3] ;

L = [1, 2, a, 3] ;

L = [1, 2, 3, a] ;

false.

# Permutacions d'una llista

?-permutació([a,b,c],L).

L = [a, b, c] ;

L = [b, a, c] ;

L = [b, c, a] ;

L = [a, c, b] ;

L = [c, a, b] ;

L = [c, b, a] ;

false.

# Predicats de control

En PROLOG existeixen predicats que no ens diuen res del nostre univers de discurs, que no representen cap coneixement:

- Expressen informació de control
- Permeten la comunicació amb els programadors
- Afecten a l'estructura interna de les proposicions
- Afegeixen o eliminen informació a la base de coneixements



# Lectura i escriptura de termes

- `write(X)`: mostra X per la pantalla
- `nl`: nova línia
- `tab(X)`: desplaça el cursor X espais a la dreta
- `read(X)`: llegeix el següent terme entrat pel teclat (acabat en punt)
- `get(C)`: llegeix un caràcter imprimible (sense punt)
- `get0(C)`: llegeix qualsevol caràcter (sense punt)

# Lectura i escriptura a fitxers

- `tell(X)`: defineix el fitxer de treball de sortida
- `telling(X)`: retorna el fitxer de treball de sortida
- `told`: tanca el fitxer de treball i estableix la sortida a la pantalla
- `see(X)`, `seeing(X)` i `seen`: és el mateix que abans però per a la lectura

# Consulta de bases de coneixement

- `consult(X)`: llegeix totes les clàusules del fitxer X.
- `reconsult(X)`: igual que `consult` però substituint els predicats antics (a `swi-prolog` és exactament el mateix que el `consult`).
- `listing(X)`: mostra totes les clàusules referents al predicat X.
- `listing` sense arguments, mostra tot el que hi ha a la base de coneixements

# Consulta de bases de coneixement

- El PROLOG permet que l'entrada dels consult(X) i reconsult(X) es simplifiquin en forma de llista:

?-[arxiu1, arxiu2, -arxiu3].

representa el consult dels arxius 1,2 i reconsult del 3.

?-[user].

permet l'entrada interactiva de fets i regles

# Inserció i esborrat de clàusules

- `asserta(X)`: afegeix noves clàusules al principi de la base de coneixements
- `assert(X)`, `assertz(X)`: afegeixen noves clàusules al final de la base de coneixements.
- `retract(X)`: Elimina la primera clàusula que unifiqui amb X
- `retractall(X)`: Elimina totes les clàusules que unifiquin amb X

# Predicats bàsics de test

- `atom(X)`: veritat si X és un àtom.
- `number(X)`: veritat si X és un número.
- `atomic(X)`: veritat si X és un àtom o un número.
- `integer(X)`: veritat si X és un número enter.
- `float(X)`: veritat si X és un número real.
- `var(X)`: veritat si X és una variable instanciada.
- `nonvar(X)`: veritat si X és una variable no instanciada
- `is_list(X)`: veritat si X és una llista

# Funcions aritmètiques

• <code>abs(X)</code> : valor absolut de X.	• <code>min(X,Y)</code> : mínim de X i Y.
• <code>integer(X)</code> : Converteix X en enter.	• <code>max(X,Y)</code> : màxim de X i Y.
• <code>float(X)</code> : Converteix X en un real.	• <code>random(X)</code> : valor aleatori entre 0 i 1.
• <code>round(X)</code> : redondeja X.	• <code>sqrt(X)</code> : arrel quadrada de X.
• <code>truncate(X)</code> : elimina la part real de X.	• <code>log(X)</code> : logaritme neperià (base e).
• <code>floor(X)</code> : valor enter menor de X.	• <code>exp(X)</code> : e elevat a X.
• <code>ceiling(X)</code> : valor enter major que X	• <code>pi</code> : 3.141592653589793
• <code>between(A,B,X)</code> : X és un valor entre A i B	• <code>e</code> : 2.718281828459045

# Funcions trigonomètriques

•sin(X)	•asin(X)
•cos(X)	•acos(X)
•tan(X)	•atan(X)



# Funcions amb llistes

<code>member(X,L).</code>	<code>pertany</code>
<code>append(L1,L2,L3).</code>	<code>afegeix dues llistes</code>
<code>delete(L1,X,L2).</code>	<code>borra tots els X de L1</code>
<code>length(L,N).</code>	<code>N és la longitud de L</code>
<code>last(X,L).</code>	<code>X és el darrer element de L</code>
<code>nth0(N,L,X).</code>	<code>torna el n-éssim element començant per 0</code>
<code>nth1(N,L,X).</code>	<code>torna el n-éssim element començant per 1</code>
<code>reverse(L1,L2).</code>	<code>inverteix L1</code>
<code>merge(L1,L2,L3).</code>	<code>Si L1 i L2 estan ordenades, L3 és la mescla</code>
<code>sort(L1,L2).</code>	<code>L2 és la llista L1 ordenada sense repetits</code>
<code>msort(L1,L2).</code>	<code>L2 és la llista L1 ordenada amb repetits</code>

# Funcions addicionals

atom\_chars(X,L).    X és un àtom i L la llista dels seus caràcters

atom\_string(X,S).    X és un àtom i S un string amb el valor atom

random(I,F,V)        Generar un número aleatori V entre I i F

random\_permutation(L1,L2).    L2 és una permutació aleatòria de L1

fail.                És un predicat que sempre és fals

## Exemple 1: Aritmogrames

	*		-		=7
+		-		-	
	*	4	/		=2
+		+		+	
	-		+		=7
=7		=4		=7	

## Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

# Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

solució(A,B,C,D,E,F,G,H):-

# Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

solució(A,B,C,D,E,F,G,H):-  
digit(A),digit(B),digit(C),

# Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

solució(A,B,C,D,E,F,G,H):-  
digit(A),digit(B),digit(C),  
7 is A\*B-C,

# Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

```
solució(A,B,C,D,E,F,G,H):-  
    digit(A),digit(B),digit(C),  
    7 is A*B-C,  
    digit(D),digit(E),
```



## Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

```
solució(A,B,C,D,E,F,G,H):-  
    digit(A),digit(B),digit(C),  
    7 is A*B-C,  
    digit(D),digit(E),  
    2 is D*4//E,
```

## Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

```
solució(A,B,C,D,E,F,G,H):-  
    digit(A),digit(B),digit(C),  
    7 is A*B-C,  
    digit(D),digit(E),  
    2 is D*4//E,  
    digit(F),digit(G),digit(H),
```

## Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

```
solució(A,B,C,D,E,F,G,H):-  
    digit(A),digit(B),digit(C),  
    7 is A*B-C,  
    digit(D),digit(E),  
    2 is D*4//E,  
    digit(F),digit(G),digit(H),  
    7 is F-G+H,
```

## Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

```
solució(A,B,C,D,E,F,G,H):-  
    digit(A),digit(B),digit(C),  
    7 is A*B-C,  
    digit(D),digit(E),  
    2 is D*4//E,  
    digit(F),digit(G),digit(H),  
    7 is F-G+H,  
    7 is A+D+F,
```

# Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

```
solució(A,B,C,D,E,F,G,H):-  
    digit(A),digit(B),digit(C),  
    7 is A*B-C,  
    digit(D),digit(E),  
    2 is D*4//E,  
    digit(F),digit(G),digit(H),  
    7 is F-G+H,  
    7 is A+D+F,  
    4 is B-4+G,
```

# Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

```

solució(A,B,C,D,E,F,G,H):-
    digit(A),digit(B),digit(C),
    7 is A*B-C,
    digit(D),digit(E),
    2 is D*4//E,
    digit(F),digit(G),digit(H),
    7 is F-G+H,
    7 is A+D+F,
    4 is B-4+G,
    7 is C-E+H.
    
```

# Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

```

solució(A,B,C,D,E,F,G,H):-
    digit(A),digit(B),digit(C),
    7 is A*B-C,
    digit(D),digit(E),
    2 is D*4//E,
    digit(F),digit(G),digit(H),
    7 is F-G+H,
    7 is A+D+F,
    4 is B-4+G,
    7 is C-E+H.
    
```

```

digit(X):-pertany(X,[1,2,3,4,5,6,7,8,9]).
    
```

# Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

```

solució(A,B,C,D,E,F,G,H):-
    digit(A),digit(B),digit(C),
    7 is A*B-C,
    digit(D),digit(E),
    2 is D*4//E,
    digit(F),digit(G),digit(H),
    7 is F-G+H,
    7 is A+D+F,
    4 is B-4+G,
    7 is C-E+H.

```

```

?-solució(A,B,C,D,E,F,G,H):-
A = E, E = 2,
B = 5,
C = G, G = 3,
D = 1,
F = 4,
H = 6 ;
A = D, D = 2,
...

```



# Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

solució:-

digit(A),digit(B),digit(C),  
 7 is A\*B-C,  
 digit(D),digit(E),  
 2 is D\*4//E,  
 digit(F),digit(G),digit(H),  
 7 is F-G+H,  
 7 is A+D+F,  
 4 is B-4+G,  
 7 is C-E+H.

# Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

solució:-

digit(A),digit(B),digit(C),  
 7 is A\*B-C,  
 digit(D),digit(E),  
 2 is D\*4//E,  
 digit(F),digit(G),digit(H),  
 7 is F-G+H,  
 7 is A+D+F,  
 4 is B-4+G,  
 7 is C-E+H.

?-solució.

true

## Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

solució:-

```
digit(A),digit(B),digit(C),  
7 is A*B-C,  
digit(D),digit(E),  
2 is D*4//E,  
digit(F),digit(G),digit(H),  
7 is F-G+H,  
7 is A+D+F,  
4 is B-4+G,  
7 is C-E+H,  
write([A,B,C,D,E,F,G,H]).
```

# Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

solució:-

```

digit(A),digit(B),digit(C),
7 is A*B-C,
digit(D),digit(E),
2 is D*4//E,
digit(F),digit(G),digit(H),
7 is F-G+H,
7 is A+D+F,
4 is B-4+G,
7 is C-E+H,
write([A,B,C,D,E,F,G,H]).

```

?- solució.

```
[2,5,3,1,2,4,3,6]
```

```
true;
```

```
[2,5,3,2,3,3,3,7]
```

```
true;
```

```
[2,6,5,2,4,3,2,6]
```

```
true
```

```
...
```

## Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

solució:-

```
digit(A),digit(B),digit(C),  
7 is A*B-C,  
digit(D),digit(E),  
2 is D*4//E,  
digit(F),digit(G),digit(H),  
7 is F-G+H,  
7 is A+D+F,  
4 is B-4+G,  
7 is C-E+H,  
write([A,B,C,D,E,F,G,H]),  
nl,fail.
```

# Exemple 1: Aritmogrames

A	*	B	-	C	=7
+		-		-	
D	*	4	/	E	=2
+		+		+	
F	-	G	+	H	=7
=7		=4		=7	

solució:-

```

digit(A),digit(B),digit(C),
7 is A*B-C,
digit(D),digit(E),
2 is D*4//E,
digit(F),digit(G),digit(H),
7 is F-G+H,
7 is A+D+F,
4 is B-4+G,
7 is C-E+H,
write([A,B,C,D,E,F,G,H]),
nl,fail.

```

?- solució.

[2,5,3,1,2,4,3,6]

[2,5,3,2,3,3,3,7]

[2,6,5,2,4,3,2,6]

...

## Exemple 2. Jugam a bolles

En Toni i en Joan juguen a bolles i en Toni li diu a n'en Joan:

“escolta, si me regales una de les teves bolles, en tendrem la mateixa quantitat”.

En Joan respon:

“es clar, i si tu me'n regales una de les teves jo en tendré el doble que tu”.

Quantes bolles té cadascú ?

## Exemple 2. Jugam a bolles

En Toni i en Joan juguen a bolles i en Toni li diu a n'en Joan:

“escolta, si me regales una de les teves bolles, en tendrem la mateixa quantitat”.

$X1 \text{ is } X+1, X1 \text{ is } Y-1$       % X bolles Toni, Y = bolles Joan

En Joan respon:

“es clar, i si tu me'n regales una de les teves jo en tendré el doble que tu”.

Quantes bolles té cadascú ?



## Exemple 2. Jugam a bolles

En Toni i en Joan juguen a bolles i en Toni li diu a n'en Joan:

“escolta, si me regales una de les teves bolles, en tendrem la mateixa quantitat”.

$X1 \text{ is } X+1, X1 \text{ is } Y-1$       % X bolles Toni, Y = bolles Joan

En Joan respon:

“es clar, i si tu me'n regales una de les teves jo en tendré el doble que tu”.

$X2 \text{ is } X-1, Y2 \text{ is } Y+1, Y2 \text{ is } X2*2.$

Quantes bolles té cadascú ?

## Exemple 2. Jugam a bolles

`b o l l e s (X,Y) :-`

`between(1,30,X) ,`

`between(1,30,Y) ,`

`X1 is X+1,`

`X1 is Y-1,       %i g u a l`

`X2 is X-1,`

`Y2 is Y+1,`

`Y2 is X2*2.       %doble`

## Exemple 2. Jugam a bolles

`bolles(X,Y) :-`

`between(1,30,X) ,`

`between(1,30,Y) ,`

`X1 is X+1,`

`X1 is Y-1,       %i g u a l`

`X2 is X-1,`

`Y2 is Y+1,`

`Y2 is X2*2.       %doble`

`?- bolles(X,Y).`

`X = 5,`

`Y = 7 ;`

`false.`

## Exemple 3. Va de rebaixes !

Estan a punt d'acabar les rebaixes i encara me queden 500 euros per gastar.

Vull comprar pantalons, camisetes i mocadors i a la tenda que he trobat millor de preu, els pantalons valen 25 euros cada un, les camisetes 5 euros la unitat i de mocadors en donen quatre per un euro.

Vull gastar tot el que em queda, que m'he puc comprar ?

## Exemple 3. Va de rebaixes !

`compra(P,C,M):-`

## Exemple 3. Va de rebaixes !

```
compra(P,C,M):-  
    MaxPantalons is 500//25,  
    MaxCamisetes is 500//5,  
    MaxMocadors is 500*4,
```

## Exemple 3. Va de rebaixes !

```
compra(P,C,M):-  
    MaxPantalons is 500//25,  
    MaxCamisetes is 500//5,  
    MaxMocadors is 500*4,  
    entre(1,MaxPantalons,P),  
    entre(1,MaxCamisetes,C),  
    entre(1,MaxMocadors,M),
```

## Exemple 3. Va de rebaixes !

```
compra(P,C,M):-  
    MaxPantalons is 500//25,  
    MaxCamisetes is 500//5,  
    MaxMocadors is 500*4,  
    entre(1,MaxPantalons,P),  
    entre(1,MaxCamisetes,C),  
    entre(1,MaxMocadors,M),  
    500 is ceiling(P*25+C*5+M//4).
```



## Exemple 3. Va de rebaixes !

```
compra(P,C,M):-  
    MaxPantalons is 500//25,  
    MaxCamisetes is 500//5,  
    MaxMocadors is 500*4,  
    entre(1,MaxPantalons,P),  
    entre(1,MaxCamisetes,C),  
    entre(1,MaxMocadors,M),  
    500 is ceiling(P*25+C*5+M//4).
```

```
?-compra(P,C,M).  
P = C, C = 1,  
M = 1877 ;  
...
```

## Exemple 3. Va de rebaixes !

```
compra(P,C,M):-  
    MaxPantalons is 500//25,  
    MaxCamisetes is 500//5,  
    MaxMocadors is 500*4,  
    entre(5,MaxPantalons,P),           % mínim 5 pantalons  
    entre(10,MaxCamisetes,C),          % mínim 10 camisetes  
    entre(1,MaxMocadors,M),M<500,     % menys de 500 mocadors  
    500 is ceiling(P*25+C*5+M//4).
```

```
?- compra(P,C,M).
```

```
P = 5,
```

```
C = 50,
```

```
M = 497 ;
```

```
P = 5,
```

```
C = 50,
```

```
M = 498;
```

```
...
```

## Exemple 3. Va de rebaixes !

```
compra(P,C,M):-  
    MaxPantalons is 500//25,  
    MaxCamisetes is 500//5,  
    MaxMocadors is 500*4,  
    entre(2,MaxPantalons,P),           % mínim dos pantalons  
    entre(4,MaxCamisetes,C),          % mínim 4 camisetes  
    entre(1,MaxMocadors,M),M<500,     % menys de 500 mocadors  
    T is ceiling(P*25+C*5+M//4), T=500,  
    R is T-P*25-C*5-M*0.25,  
    write("Sobren: "),write(R),write(" euros").
```

```
?- compra(P,C,M).  
Sobren: 0.75 euros  
P = 5,  
C = 50,  
M = 497 ;  
Sobren: 0.5 euros  
P = 5,  
C = 50,  
M = 498 ;  
...
```

## Exemple 4. Aiii ciclistes !

En una carrera de ciclistes, 4 aficionats comenten el resultat final dels favorits: a, b i c. Els comentaris són els següents:

- Guanyarà a o c
- Si a no guanya, llavors guanyarà b
- Si a arriba tercer, llavors c no guanyarà
- a o b arribaran el segon

Acaba la carrera i a, b i c arriben els tres primers, i a més, es compleixen totes les afirmacions anteriors.

En quin ordre han arribat els tres corredors ?

## Exemple 4. Aiii ciclistes !

Representarem la informació amb una llista ordenada de tres elements, on la posició dins la llista indica l'ordre d'arribada d'un ciclista:

- Guanyarà a o c  
comentari1( [ a , \_ , \_ ] ) .  
comentari1( [ c , \_ , \_ ] ) .
- Si a no guanya, llavors guanyarà b  
comentari2( [ a , \_ , \_ ] ) .  
comentari2( [ X , \_ , \_ ] ) :- X \= a , X = b.
- Si a arriba tercer, llavors c no guanyarà  
comentari3( [ X , \_ , a ] ) :- X \= c.  
comentari3( [ a , \_ , \_ ] ).  
comentari3( [ \_ , a , \_ ] ).
- a o b arribaran el segon  
comentari4( [ \_ , b , \_ ] ).  
comentari4( [ \_ , a , \_ ] ).

## Exemple 4. Aiii ciclistes !

tresPrimers (L):-

```
    permutació( [ a , b , c ] ,L) ,  
    comentari1(L) ,  
    comentari2(L) ,  
    comentari3(L) ,  
    comentari4(L ).
```

?- tresprimers(L).

L = [a, b, c] ;

false