# Inteligência Artificial

## Programação em Lógica II

Licenciatura em Engenharia Informática

2024/2025

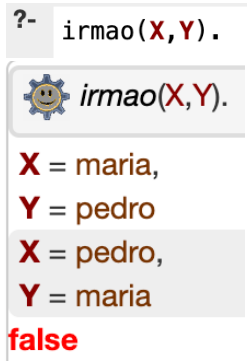
Regras

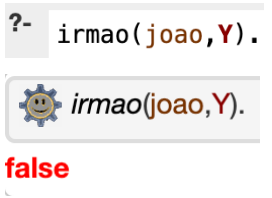
# _introdução

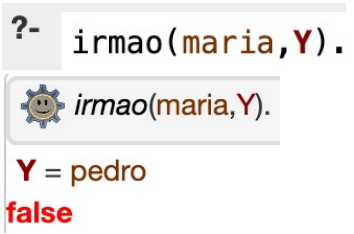
- Regras (*rules*) são fundamentais para definir relacionamentos lógicos entre factos e para realizar inferências a partir desses factos
- Elas permitem que o programa tome decisões e conclua novos conhecimentos com base nas informações existentes
- Uma regra tem a seguinte estrutura
  - cabeçalho :- corpo
  - O cabeçalho, composto por functor e zero ou mais argumentos, corresponde ao que queremos provar ou inferir (conclusão)
  - O corpo define as conclusões que têm que ser verdadeiras para que o cabeçalho (conclusão) também o seja
- Tal como nos factos, podemos ter regras com o mesmo nome e mesmo nº de argumentos, ou até com o mesmo nome e nº de argumentos diferentes

```
% factos
pai(joao, maria).
pai(joao, pedro).
pai(joao, tozé).
mae(ana, maria).
mae(ana, pedro).

% regras
irmao(X, Y) :- pai(P, X), pai(P, Y),
    mae(M, X), mae(M, Y), X \= Y.
```



Regras

# _exemplo

```
passa(X):- nota(X,Y), Y > 9.5.

filho(X,Y):-homem(X),
            (descendente(X,Y,_);descendente(X,_,Y)).

potência(_,0,1):-!.
potência(X,N,P):- N1 is N-1,
                  potência(X,N1,P1),
                  P is X*P1.
```

Regras

# _introdução

- As regras são fundamentais em Prolog porque
  - Permitem deduzir nova informação, a partir dos factos existentes
  - Permitem generalizar o conhecimento existente num domínio, evitando ter que se definir todos os factos
- Exemplo
  - Os ângulos agudos medem menos de 90 graus
  - Os ângulos retos medem 90 graus
  - Os ângulos obtusos medem mais de 90 graus
  - Os ângulos rasos medem 180 graus

```
angulo(0,agudo).
angulo(1,agudo).
angulo(2,agudo).
%...
angulo(90,reto).
%...
angulo(91, obtuso).
angulo(92, obtuso).
angulo(93, obtuso).
angulo(94, obtuso).
%...
angulo(180, raso).
```



```
angulo(90,reto).
angulo(180,raso).
angulo(X,agudo):-X>=0,X<90.
angulo(X,obtuso):-X>90,X<180.
```

# _prova de teoremas

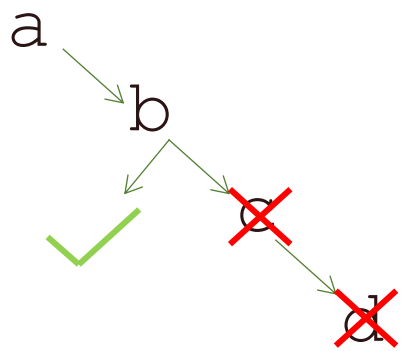
- A linguagem PROLOG usa o "Princípio da Resolução" para a "Prova de Teoremas", através de um processo de inferência
- A Prova de Teoremas segue a Pesquisa Primeiro em Profundidade, na tentativa de provar o teorema, a este mecanismo acrescenta-se o retorno atrás quando ocorre falha (*backtracking*)
- As variáveis (literais começados por maiúscula) podem residir num de dois estados: não instanciadas ou instanciadas, quando é encontrada uma solução podem ser exibidas as instanciações possíveis

# _como funciona

```
a :- b.
b :- c.
c :- d.
b.
```

Qual o resultado de **?- a.**

Set of goals to prove

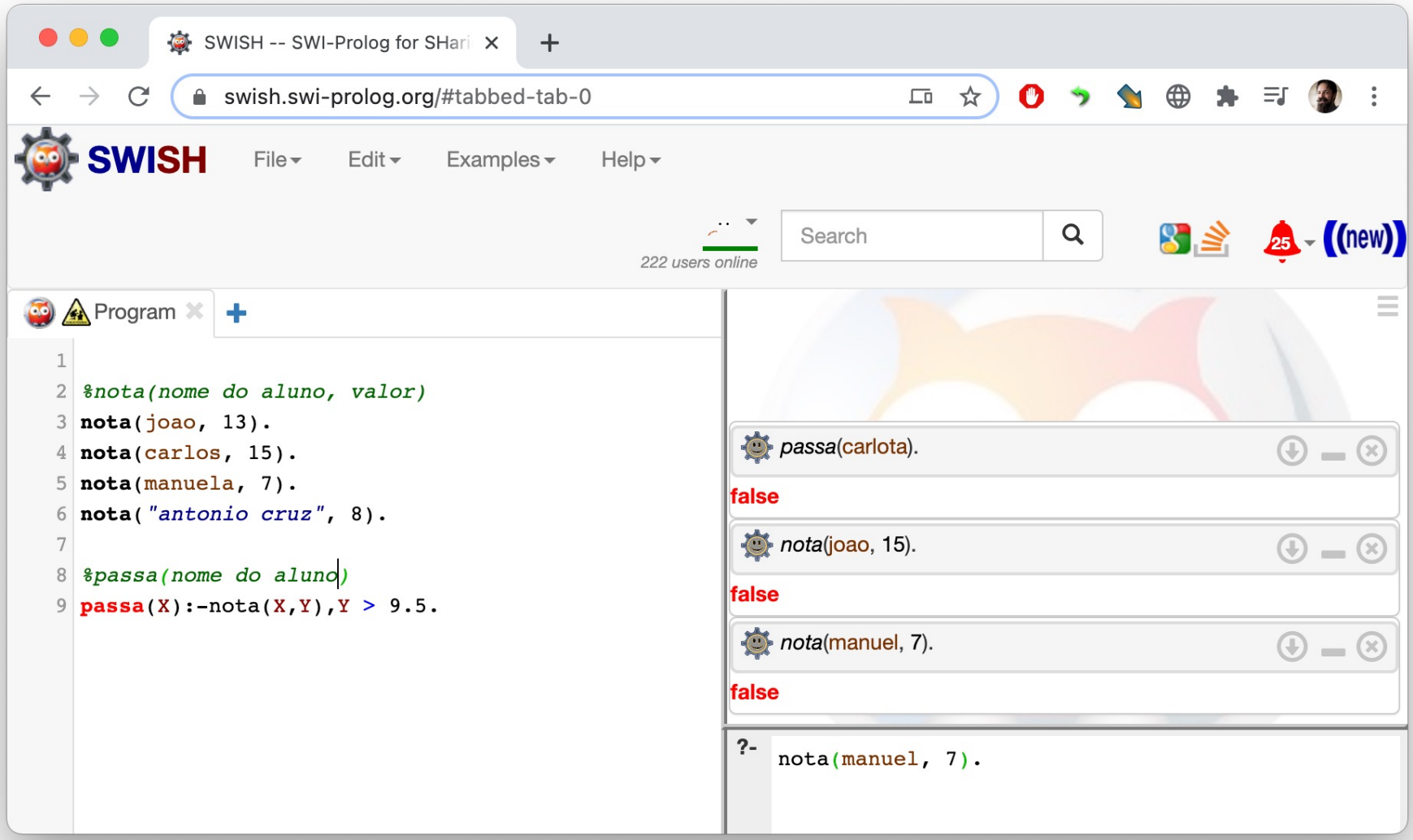


# _backtracking

- Mecanismo do prolog para encontrar múltiplas soluções
- Quando um caminho de prova falha, tenta outras alternativas até as esgotar todas

# _princípio do mundo fechado

- Quando algo não está explicitamente definido como um axioma é assumido como sendo falso (Princípio do Mundo Fechado)
- Há várias extensões ao PROLOG que assumem uma lógica tri-valor (verdadeiro, falso ou desconhecido)
- Em PROLOG é ainda possível criar/remover dinamicamente axiomas (*não será abordado*)


SWISH -- SWI-Prolog for SHari
swish.swi-prolog.org/#tabbed-tab-0
SWISH
File
Edit
Examples
Help
222 users online
Search
Program
%nota(nome do aluno, valor)
nota(joao, 13).
nota(carlos, 15).
nota(manuela, 7).
nota("antonio cruz", 8).
%passa(nome do aluno)
passa(X):-nota(X,Y),Y > 9.5.
passa(carlota).
false
nota(joao, 15).
false
nota(manuel, 7).
false
?-
nota(manuel, 7).

# _regras e factos

```
fica(porto,portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).

passa(douro,porto).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).
```

*Defina as seguintes regras:*

% um rio é português se passa numa cidade que fica em Portugal
*rio_portugues(R)*

% duas cidades são banhadas pelo mesmo rio se esse rio passa em ambas as cidades
*banhadas_mesmo_rio(C1, C2)*

# _questões

```
fica(porto,portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).

passa(douro,porto).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).

rio_português(R):-passa(R,C),fica(C,portugal).
banhadas_mesmo_rio(C1,C2):-passa(R,C1),passa(R,C2),C1\==C2.
```

```
?-rio_português(Rio).
```

# _questões

```
fica(porto,portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).

passa(douro,porto).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).

rio_português(R):-passa(R,C),fica(C,portugal).
banhadas_mesmo_rio(C1,C2):-passa(R,C1),passa(R,C2),C1\==C2.
```

sucesso (2)

sucesso (1)

```
?-rio_português(Rio).
Rio=douro
yes
```

Na chamada à regra, do lado esquerdo, Rio e R passam a ser a mesma variável

passa(R,C) tem sucesso com R=douro e C=porto

A chamada seguinte já é feita com C já instanciada com porto, na prática essa chamada é feita como sendo fica(porto,portugal)

Quando se atinge o ponto a regra tem sucesso

# _questões

```
fica(porto,portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).

passa(douro,porto).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).

rio_português(R):-passa(R,C),fica(C,portugal).
banhadas_mesmo_rio(C1,C2):-passa(R,C1),passa(R,C2),C1\==C2.
```

```
?-rio_português(tejo).
```

# _questões

```
fica(porto,portugal).                falha
fica(lisboa,portugal).               sucesso (2)
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).

passa(douro,porto).                  falha
passa(douro,zamora).                 falha
passa(tejo,lisboa).                  sucesso (1)
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).

rio_português(R):-passa(R,C),fica(C,portugal).
banhadas_mesmo_rio(C1,C2):-passa(R,C1),passa(R,C2),C1\==C2.
```

```
?-rio_português(tejo).
yes
```

Na chamada à regra, do lado esquerdo, R fica instanciada com o valor tejo

Dentro da regra (lado direito) a 1ª chamada já é feita como sendo passa(tejo,C) e tem sucesso com R=tejo e C=lisboa

A chamada seguinte já é feita com C já instanciada com lisboa, na prática essa chamada é feita como sendo fica (lisboa,portugal)

Quando se atinge o ponto a regra tem sucesso

# _ordem dos factos/questões

```
?-rio_português(tejo).
```

```
fica(porto,portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).

passa(douro,porto).
passa(douro,zamora).
passa(tejo,toledo).
passa(tejo,lisboa).
passa(minho,caminha).
passa(minho,orense).

rio_português(R):-passa(R,C),fica(C,portugal).
banhadas_mesmo_rio(C1,C2):-passa(R,C1), passa(R,C2),C1\==C2.
```

# _ordem dos factos/questões

```
fica(porto,portugal).          falha       falha
fica(lisboa,portugal).         falha       sucesso (3)
fica(coimbra,portugal).        falha
fica(caminha,portugal).        falha
fica(madrid,espanha).          falha
fica(barcelona,espanha).       falha
fica(zamora,espanha).          falha
fica(orense,espanha).          falha
fica(toledo,espanha).          falha

passa(douro,porto).            falha
passa(douro,zamora).           falha
passa(tejo,toledo).            sucesso (1)
passa(tejo,lisboa).            sucesso (2)
passa(minho,caminha).
passa(minho,orense).

rio_português(R):-passa(R,C),fica(C,portugal).
banhadas_mesmo_rio(C1,C2):-passa(R,C1), passa(R,C2),C1\==C2.
```

```
?-rio_português(tejo).
yes
```

Na chamada à regra, do lado esquerdo, R fica instanciada com o valor tejo;

Dentro da regra (lado direito) a 1ª chamada já é feita como sendo passa(tejo,C) e tem sucesso com R=tejo e C=toledo;

A chamada seguinte já é feita com C já instanciada com toledo, na prática essa chamada é feita como sendo fica(toledo,portugal) e falha

Volta-se atrás (*backtracking*) e é tentada uma nova solução para passa(tejo,C), ficando C=lisboa

A chamada seguinte já é feita com C já instanciada com lisboa, na prática essa chamada é feita como sendo fica(lisboa,portugal)

Quando se atinge o ponto a regra tem sucesso

Prolog

# _ordem dos factos/questões

```
fica(porto,portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).

passa(douro,porto).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).

rio_português(R):-passa(R,C), fica(C,portugal).
banhadas_mesmo_rio(C1,C2):-passa(R,C1), passa(R,C2),
                           C1\==C2.
```

```
?-banhadas_mesmo_rio(C1,C2).
```

# _ordem dos factos/questões

```
fica(porto,portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).

passa(douro,porto).     suc.(1) suc.(2)
passa(douro,zamora).    suc.(3)
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).

rio_português(R):-passa(R,C), fica(C,portugal).
banhadas_mesmo_rio(C1,C2):-passa(R,C1), passa(R,C2),
                           C1\==C2.
```

```
?-banhadas_mesmo_rio(C1,C2).
C1=porto C2=zamora
yes
```

Na chamada à regra, do lado esquerdo, C1 e C2 continuam como sendo não instanciadas;

Dentro da regra (lado direito) a 1ª chamada já é feita como sendo passa(R,C1) e tem sucesso com R=douro e C1=porto;

A chamada seguinte já é feita com R já instanciada com douro, na prática essa chamada é feita como sendo passa(douro,C2) e tem sucesso com C2=porto

O teste seguinte falha (porto não é diferente de porto) e faz-se o *backtracking*

Agora passa(douro,C2) tem sucesso com C2=zamora

O teste seguinte tem sucesso (porto é diferente de zamora)

Quando se atinge o ponto a regra tem sucesso

# _questões

```
fica(porto,portugal).
fica(lisboa,portugal).
fica(coimbra,portugal).
fica(caminha,portugal).
fica(madrid,espanha).
fica(barcelona,espanha).
fica(zamora,espanha).
fica(orense,espanha).
fica(toledo,espanha).

passa(douro,porto).
passa(douro,zamora).
passa(tejo,lisboa).
passa(tejo,toledo).
passa(minho,caminha).
passa(minho,orense).

rio_português(R):-passa(R,C),fica(C,portugal).
banhadas_mesmo_rio(C1,C2):-passa(R,C1),passa(R,C2),C1\==C2.
```

Experimente fazer as seguintes questões e efectue as "traçagens"

```
?-banhadas_mesmo_rio(orense,C).

?-banhadas_mesmo_rio(C,lisboa).

?-banhadas_mesmo_rio(zamora,porto).

?-banhadas_mesmo_rio(lisboa,porto).

?-banhadas_mesmo_rio(coimbra,C).
```

# _o predicado cut

- Quando não é necessário encontrar todas as soluções possíveis pode usar-se o predcado cut: !
- Permite indicar ao prolog objetivos já satisfeitos e que não precisam ser reconsiderados no processo de backtracking

```
a1(X, Y) :- b(X), c(Y).

b(1).
b(2).
b(3).

c(1).
c(2).
c(3).
```

# _o predicado cut

- Quando não é necessário encontrar todas as soluções possíveis pode usar-se o predcado cut: !
- Permite indicar ao prolog objetivos já satisfeitos e que não precisam ser reconsiderados no processo de backtracking

```
a1(X, Y) :- b(X), c(Y).

b(1).
b(2).
b(3).

c(1).
c(2).
c(3).
```

```
a1(X,Y).
X = Y, Y = 1 ;
X = 1, Y = 2 ;
X = 1, Y = 3 ;
X = 2, Y = 1 ;
X = Y, Y = 2 ;
X = 2, Y = 3 ;
X = 3, Y = 1 ;
X = 3, Y = 2 ;
X = Y, Y = 3.
```

# _o predicado cut

- Quando não é necessário encontrar todas as soluções possíveis pode ser usar o predcado cut: !
- Permite indicar ao prolog objetivos já satisfeitos e que não precisam ser reconsiderados no processo de backtracking

```
a2(X, Y) :- b(X), !, c(Y).

b(1).
b(2).
b(3).

c(1).
c(2).
c(3).
```

# _o predicado cut

- Quando não é necessário encontrar todas as soluções possíveis pode ser usar o predcado cut: !
- Permite indicar ao prolog objetivos já satisfeitos e que não precisam ser reconsiderados no processo de backtracking

```
a2(X, Y) :- b(X), !, c(Y).

b(1).
b(2).
b(3).

c(1).
c(2).
c(3).
```

```
a2(X,Y).
X = Y, Y = 1 ;
X = 1, Y = 2 ;
X = 1, Y = 3.
```

# _o predicado cut

- O predicado cut permite muitas vezes simplificar as regras e/ou a sua execução, evitando a pesquisa quando já sabemos que não há alternativas possíveis

```
angulo(90,reto).
angulo(180,raso).
angulo(X,agudo):-X>=0,X<90.
angulo(X,obtuso):-X>90,X<180.
```

```
trace,angulo(90,X).
   Call: angulo(90,_5544)
   Exit: angulo(90,reto)
X = reto
   Redo: angulo(90,_446)
      Call: 90>=0
      Exit: 90>=0
      Call: 90<90
      Fail: 90<90
   Redo: angulo(90,_440)
      Call: 90>90
      Fail: 90>90
   Fail: angulo(90,_440)
false
?- trace,angulo(90,X).
```

```
angulo(90,reto):-!.
angulo(180,raso):-!.
angulo(X,agudo):-X>=0,X<90,!.
angulo(_,obtuso).
```

```
trace,angulo(90,X).
   Call: angulo(90,_5568)
   Exit: angulo(90,reto)
X = reto
?- trace,angulo(90,X).
```

# _variáveis

- As variáveis em PROLOG têm um desempenho diferente das variáveis de outras linguagens
- Em PROLOG uma variável pode estar apenas em dois estados: não instanciada ou instanciada
- Uma vez instanciada uma variável só pode mudar de valor pelo processo de *backtracking*, ou seja, voltando a ficar como não instanciada para tomar outro valor
  - Anteriormente, C tomou o valor toledo e depois por backtracking passou a tomar o valor lisboa
  - Anteriormente, C2 tomou o valor porto e depois por backtracking passou a tomar o valor zamora

# _atribuição

- = para a atribuição simbólica — X=a
- is para a atribuição numérica — X is 5

- A atribuição simbólica é bidireccional

- Para X=Y
  - Se X não está instanciado e Y está então temos X←Y
  - Se X está instanciado e Y não está então temos X→Y
  - Se nenhum está instanciado então passam a ser a mesma variável
  - Se ambos estão instanciados com o mesmo valor então há sucesso
  - Se ambos estão instanciados com valores diferentes então ocorre uma falha

# _atribuição

- A atribuição numérica é unidireccional
- Do lado direito do *is*, se estiverem envolvidas variáveis, elas devem estar instanciadas
- Do lado esquerdo a variável não deve estar instanciada, senão ocorre uma falha
- Em PROLOG N is N+1 nunca tem sucesso

# _variáveis

- Em PROLOG o incremento de uma variável nunca pode ser feito como N is N+1
  - Se N não estiver instanciado ocorre uma falha quando se tenta avaliar N+1
  - Se N estiver instanciado não poderemos obrigar a mudar o seu valor
  - Pode ser usado N1 is N+1
- Se for pedida a impressão de uma variável não instanciada aparecerá um nº antecedido de _ (por exemplo _22456) que tem a ver com a referência da variável e não com o seu valor
- Quando num facto ou regra não interesse o valor de uma variável, esta pode ser substituída por _

# _exemplo

```
?- write('X='),write(X),nl,X=a,write('X='),write(X),nl.
X =_22650
X = a
X = a

?- write('X='),write(X),nl,X=a,write('X='),write(X),nl,X=b.
X =_39436
X = a
no
```

# _atribuição

```
?- X=Y,X=a.
X = Y = a
```

```
?- Y=a,X=Y.
Y = X = a
```

```
?- X=a,X=Y.
X = Y = a
```

```
?- X=Y.
X = Y = _
```

```
?- X=a,Y=a,X=Y.
X = Y = a
```

```
?- X=a,Y=b,X=Y.
no
```

# Inteligência Artificial

## Programação em Lógica II

Licenciatura em Engenharia Informática

2024/2025