

## SUMÁRIO – AULA 16

- Paradigma Lógico (2 aulas)



## COMO É O CÓDIGO

```
1 pato(pato_donald) .  
2 pássaro(professor_pardal) .  
3  
4 papagaio(loro) .  
5 coruja(maria) .  
6  
7 ave(X) :- papagaio(X) ; coruja(X) ; pato(X) ; pássaro(X) .
```

## APLICABILIDADE



## PARADIGMA LÓGICO

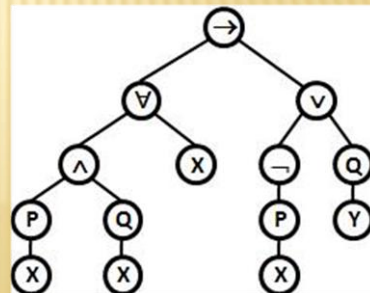


### □ Baseado no Cálculo de Predicados

- Lógica Matemática Dedutiva
- Lógica de Primeira Ordem

### □ Ex:

- Zé Carioca é um papagaio.
- Todo papagaio é uma ave.
- Logo, Zé Carioca é uma ave.



## ELEMENTOS DE UM PROGRAMA

- ❑ **Fatos:** cláusulas sem nenhuma negação
  - ❑ **verdades incondicionais**
- ❑ **Regras:** cláusulas com condicionais
  - ❑ **Se X e Y então Z**
- ❑ **Consultas:** verificação de uma verdade
  - ❑ execução do programa



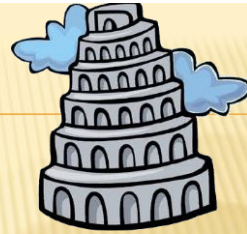
## ELEMENTOS DE UM PROGRAMA

- ❑ **Exemplo:**
  - ❑ **Fato:** Zé Carioca é um papagaio.
  - ❑ **Regra:** Todo papagaio é uma ave (ou "se X é um papagaio, X é uma ave")
  - ❑ **Consulta:** Zé Carioca é uma ave?
    - ❑ Solução/resposta/resultado:
      - ❑ Sim





## LINGUAGEM PROLOG



- ❑ PROgrammation en LOGique
- ❑ Criado por Alain Colmerauer e Philippe Roussel em 1972 para **processamento** de **linguagens naturais**.
- ❑ Apesar do longo tempo de desenvolvimento, ainda **não é** uma **linguagem portátil**
  - ❑ Diferentes **dialetos**, interpretadores e compiladores
    - ❑ cada implementação usa rotinas diferentes e incompatíveis entre si

## AMBIENTES

- ❑ SWI Prolog
- ❑ Turbo Prolog
- ❑ LPA
- ❑ Prolog
- ❑ GNU Prolog
- ❑ etc.





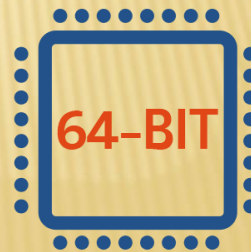
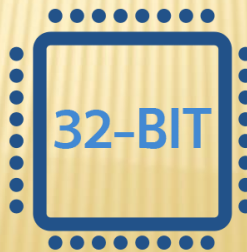
# SWI-PROLOG



❑ Baixar ambos com 32 ou 64 bits:

❑ SWI-Prolog

❑ SWI-Prolog-Editor



# SWI-PROLOG

❑ Implementação em código aberto da linguagem de programação Prolog

```

SWI-Prolog (Multi-threaded, version 6.6.1)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 6.6.1)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).
1 ?-
  
```

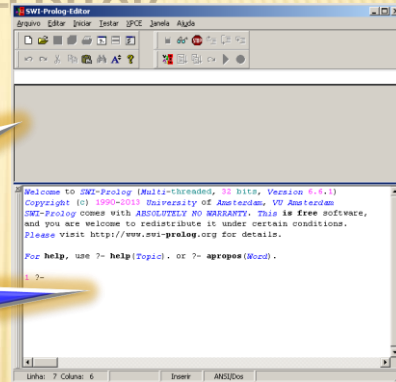
❑ <http://www.swi-prolog.org>

## SWI-PROLOG-EDITOR

### Editor gráfico

área de definições  
(onde salvo o código)

área de interações  
(onde ocorre a  
avaliação)



### Download

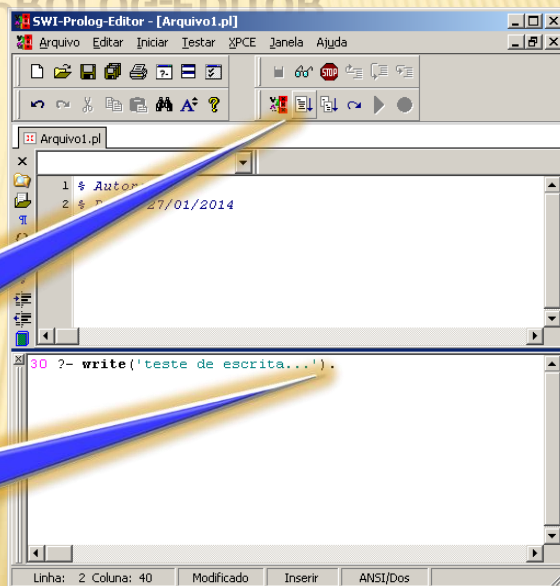
- [http://waikiki.zhaw.ch/~rege/psp\\_hs19/SWIPrologEditorSetup.exe](http://waikiki.zhaw.ch/~rege/psp_hs19/SWIPrologEditorSetup.exe)

## SWI-PROLOG-EDITOR

### Editor gráfico

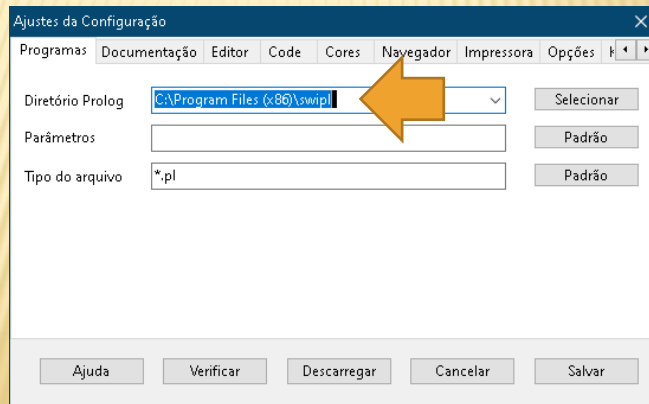
Qualquer interação com o programa exige **salvar** o arquivo e depois clicar em **Consultar**

Todo comando deve obrigatoriamente ser finalizado pelo caractere **ponto**.



## SWI-PROLOG-EDITOR

- ❑ Verificar caminho para o interpretador. Acessar:
  - ❑ Janela | Ajustes da Configuração | Programas



## SINTAXE SWI-PROLOG

- ❑ Mais na documentação em:
  - ❑ [http://www.swi-prolog.org/pldoc/doc\\_for?object=root](http://www.swi-prolog.org/pldoc/doc_for?object=root)

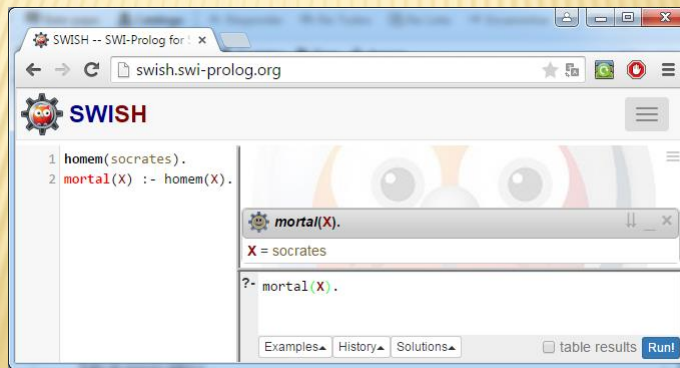


# SWISH



❑ Outra opção (online):

❑ <http://swish.swi-prolog.org>



## PROGRAMAS EM PROLOG

❑ "Zé Carioca é um papagaio"

❑ `papagaio('Ze Carioca').`

Fato

❑ "Todo papagaio é uma ave" (ou "se X é um papagaio, então X é uma ave")

❑ `ave(X) :- papagaio(X).`

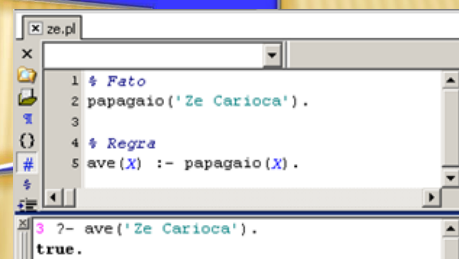
Regra

❑ Zé Carioca é uma ave?

❑ `?- ave('Ze Carioca').`

❑ `true`

Resposta



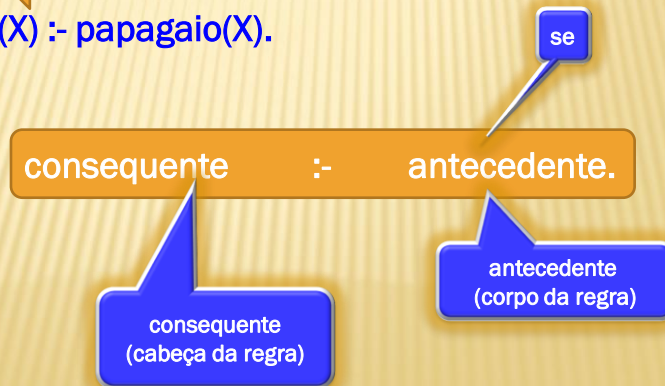


## PROGRAMAS EM PROLOG

- "Todo papagaio é uma ave" (ou "se X é um papagaio, então X é uma ave")



- `ave(X) :- papagaio(X).`



## COMO PROLOG FUNCIONA

- Usa mecanismo de **unificação**
  - atribuição de valores a variáveis
  - “casamento” entre termos
    - *matching*



## COMO PROLOG FUNCIONA

- ❑ ?- **ave**(X) = **ave**('Ze Carioca').
  - ❑ Quando o termo **X** é idêntico ao termo **Zé Carioca**?
  - ❑ Quando nós **substituímos X** com o valor **Zé Carioca**!
- ❑ ?- **ave**(X) = **ave**('Ze Carioca').
  - ❑ X = 'Ze Carioca'.

```
1 ?- ave(X) = ave('Ze carioca').
X = 'Ze carioca'.
```

## PARADIGMA LÓGICO



- ❑ Prolog é uma linguagem **declarativa**, na qual a computação é baseada na **representação** do **conhecimento** do **domínio** do **problema**.
- ❑ Nesta representação devemos considerar **todos** os **objetos** sobre os quais queremos falar, as **propriedades** de cada objeto e as **relações** entre objetos.

## PREDICADOS



- A unidade **básica** do Prolog é o **predicado**, que é um postulado **verdadeiro**, um **fato**.
- Predicados **expressam** algum **fato** sobre o mundo que o **programa** deve **conhecer**.



## PARADIGMA LÓGICO

- Prolog assume a hipótese do mundo fechado:
  - tudo o que **não está descrito** nos fatos
  - ou **não é deduzível** dos fatos

é **falso!**





## PROGRAMAS EM PROLOG



- ❑ Ciclo de programação em Prolog:
  - ❑ Definir a **base de fatos/regras/conhecimento**
  - ❑ Fornecer **base de fatos e regras** ao interpretador
    - ❑ carregar o programa
  - ❑ Solicitar a verificação de uma proposição/meta
    - ❑ **consulta/desafio**

## PROLOG: FATOS + REGRAS + ?

- ❑ Programar em Prolog envolve:
  - ❑ Declarar alguns **fatos** a respeito de **objetos** e seus **relacionamentos**. Ex:
    - ❑ `mulher(pam).` % **pam é mulher**
  - ❑ Definir algumas **regras** sobre os **objetos** e seus **relacionamentos**. Ex:
    - ❑ `genitor(pam,bob).` % **o genitor de pam é bob**
  - ❑ Fazer **perguntas** sobre os **objetos** e seus **relacionamentos**. Ex:
    - ❑ `genitor(X, bob).` % **quem são os pais de bob?**



## PROLOG: FATOS E REGRAS

- O predicado é a **relação** sobre os quais os objetos irão interagir. Ex:

- **mulher**(ann).

- **mulher**(pat).

- **amiga**(ann, pat).

- Relação de **amizade** entre dois objetos



## OPERADORES LÓGICOS

Símbolo	Conectivo	Operação Lógica
$\text{:-}$	IF	Implicação
,	AND	Conjunção
;	OR	Disjunção
not	NOT	Negação

## REGRAS COMPOSTAS



- ❑ conetivo “e” (,)
  - ❑ “Se X é pai de Y e Y é pai de Z, então X é avô de Z”
    - ❑ troque X pelo nome do seu avô.
    - ❑ troque Y pelo nome do seu pai
    - ❑ troque Z pelo seu nome
  - ❑ **avo(X,Z) :- pai(X,Y) , pai(Y,Z).**
- ❑ conetivo “ou” (;)
  - ❑ “Se X é um papagaio ou uma coruja, então X é uma ave”
  - ❑ **ave(X) :- papagaio(X) ; coruja(X).**

## PROLOG: FATOS X REGRAS

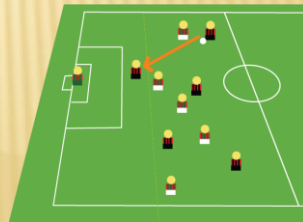
- ❑ Um fato é sempre uma informação verdadeira. Ex:

❑ **azul(céu).**



- ❑ Já uma regra precisa ser avaliada para se verificar sua veracidade. Ex:

**impedido(X) :- avançado(X), recebeBola(X).**



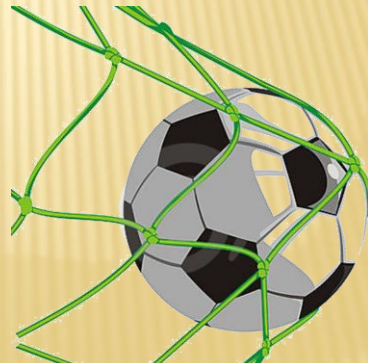
## O QUE FAZER x COMO FAZER

❑ O **que fazer** para ganhar no futebol?

❑ **fazer** mais gols que o outro time!

❑ E **como fazer** isso?

- ❑ jogadas ensaiadas
- ❑ táticas
- ❑ preparo físico
- ❑ disciplina
- ❑ habilidade
- ❑ etc...



## O QUE FAZER x COMO FAZER

❑ **Descrição** do problema ao invés de **algoritmos**

❑ Diremos **o que fazer** e não **como fazer**

❑ Especifica-se o **quê** se **sabe** sobre um problema e o **quê** deve ser **feito**.

❑ Bases de conhecimento + regras.

❑ É **mais** direcionada ao **conhecimento** e **menos** direcionada a **algoritmos**.

❑ não se especifica o “como deve ser feito”

## COMO PROLOG FUNCIONA

- ❑ A **descrição** do problema é **avaliada** por um **interpretador**.
- ❑ Ele utiliza um **motor de inferência** e realiza **deduções** em busca de **conclusões válidas** a partir das **consultas** realizadas pelos usuários.
- ❑ Uma computação em Prolog envolve a **dedução** de consequências a partir das **regras** e **fatos**.



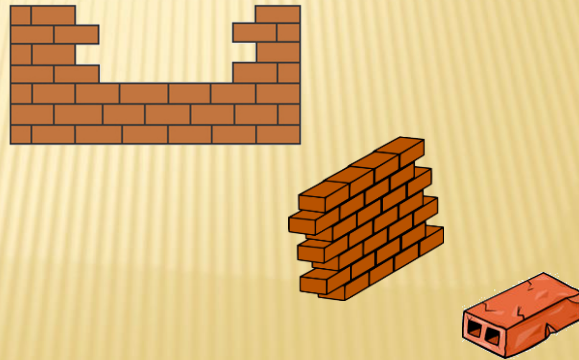
## COMO PROLOG FUNCIONA

- ❑ A **computação** destes programas é **equivalente** a **prova de um teorema** em lógica.
  - ❑ ?
- ❑ O **significado** de um programa é o conjunto de todas as consequências **deduzíveis** pela iterativa aplicação das **regras** sobre os **fatos iniciais** e os **novos fatos gerados**.
  - ❑ ? → R+ fatos → novos fatos → dedução/resposta



## COMO PROLOG FUNCIONA

- Quando o **usuário** realiza uma **consulta**, o **motor de inferência** tenta **resolver metas**, sendo que uma meta pode conter **submetas**.

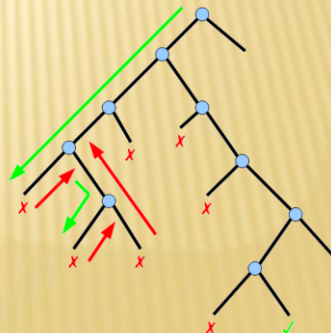


- ❑ Animal ?
  - ❑ Mamífero ?
  - ❑ ...
  - ❑ Ave ?
    - ❑ voa?
    - ❑ tem penas?

## COMO PROLOG FUNCIONA

- ❑ Quando **não existem** mais **metas** para serem **satisfeitas** em uma **linha de resolução**, o sistema utiliza encadeamento para trás (*backtracking*) em busca de **outras respostas** possíveis.



- ❑ Habilite o modo **Trace** acompanhe uma resolução...





## PARADIGMA LÓGICO

- ❑ Baseado em lógica **simbólica**
  - ❑ programação **simbólica** e não **numérica**
    - ❑ sim, podemos fazer cálculos, mas este não é o foco.
- ❑ Atua na solução de problemas que envolvam os **objetos** e suas **relações**



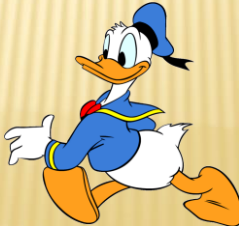




## ATIVIDADE

- ❑ Crie **fatos** e **novas regras** na base de conhecimento que estamos alimentando. Ex:
  - ❑ **papagaio('Ze Carioca')**.
  - ❑ **pato('pato donald')**.
  - ❑ **pássaro('professor pardal')**.
  - ❑ **ave(X) :- papagaio(X) ; coruja(X); pato(X).**
  - ❑ ...

## ATIVIDADE



❑ Crie consultas. Ex:

❑ “quem são as aves?”

❑ `?- ave(X).`

Para se obter mais de uma resposta digita-se ponto e virgula (;) depois da primeira resposta

❑ “*blue* é uma ave?”

❑ `?- ave('blue').`

❑ “quem são os patos?”

❑ `?- patos(P).`

## PARADIGMA LÓGICO

❑ Interpretação do Prolog em termos **matemáticos**:

❑ O sistema Prolog aceita os  **fatos**  e  **regras**  como um conjunto de  **axiomas**

❑ uma  **sentença**  ou  **proposição**  que não é provada ou demonstrada e é considerada como  **óbvia**  ou como um  **consenso inicial**  necessário para a construção ou aceitação de uma teoria.

❑ e a  **consulta**  do  **usuário**  como um  **teorema**  a ser provado.



## PARADIGMA LÓGICO

- ❑ A tarefa do sistema é **demonstrar** que o **teorema** pode ser **provado** com base nos **axiomas** representados pelo conjunto das **cláusulas** que constituem o programa.
- ❑ Ou seja, que a **consulta** possa ser **verificada** com base nos **fatos** e **regras** que compõe o programa.



## PARADIGMA LÓGICO

- ❑ `homem(socrates)`
- ❑ `mortal(X) :- homem(X).`
- ❑ `?- mortal(socrates).`
  - ❑ `"true"`
- ❑ `?- mortal(X)`
  - ❑ `"X=socrates."`

axiomas

Sócrates é um homem.

Todos os homens são mortais.

teoremas

Sócrates é mortal?

Quem é mortal?

Para todo X, se X é um homem  
então X é mortal.



## PARADIGMA LÓGICO



- ❑ Possui a princípio **todas** as vantagens do **paradigma funcional**.
- ❑ Permite concepção da aplicação em um **alto nível de abstração** (através de associações entre Entrada/Saída).

## SINTAXE SWI-PROLOG

- ❑ **comentários**
  - ❑ % este é um comentário de linha
  - ❑ /\* este é um comentário de bloco \*/
- ❑ **textos/ nomes compostos**
  - ❑ entre aspas simples / duplas
    - ❑ write('Hello world!').
    - ❑ write("Fulano de Tal").

## SINTAXE SWI-PROLOG

### ❑ Caracteres de escape (sempre entre '):

❑ `\n` - cria uma nova linha

❑ `\t` - tabulação

❑ `\\` - \

```
1 ?- write('Hello world! \n \n').
Hello world!

true.

3 ?- write('\t Hello world!').
    Hello world!

true.
```

|:  
Esperando dados...  
Finalizar com .

I/O

❑ % exibe a mensagem Teste

❑ `write('Teste de texto')` .

❑ % atribui à variável X o valor lido

❑ `read(X)`.

❑ % atribui e exibe a valor da variável X

❑ `read(X), write(X)`.

```
1 ?- write('Teste de texto').
Teste de texto
true.

3 ?- read(X).
|: 897.
X = 897.

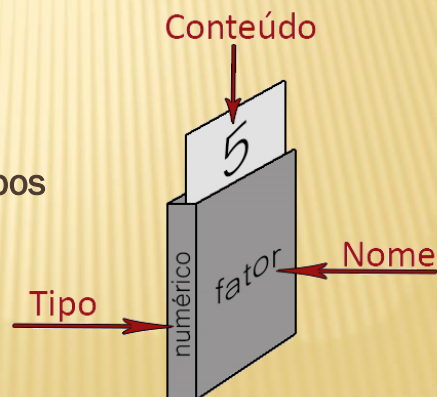
4 ?- read(X), write(X).
|: 98656.
98656
X = 98656.
```

## CONSTANTES E VARIÁVEIS

- ❑ São formados por letras, dígitos e *underscore*
  - ❑ devem começar com letra ou sublinhado.
  - ❑ **constantes:** inicial com letra minúscula
    - ❑ casa, maria, pedro, comer.
  - ❑ **variáveis:** iniciam com **LETRA MAIÚSCULA**
    - ❑ X, Valor, Casa\_propria.
  - ❑ **variáveis anônimas:** iniciam com **\_**
    - ❑ \_ignore, \_a, \_

## VARIÁVEIS?!

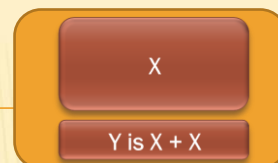
- ❑ Não são como variáveis de linguagens imperativas:
  - ❑ Não são declaradas
  - ❑ Não são vinculadas a tipos



## VARIÁVEIS?!

- ❑ Vinculação de valor (unificação/instanciação) ocorre no processo de **resolução**, para verificar se uma meta é verdadeira. Ex:
  - ❑ `papagaio('Ze Carioca')`.
  - ❑ `ave(X) :- papagaio(X)`.
    - ❑ onde `X = "Ze Carioca"`
    - ❑ `ave` e `papagaio` são predicados

## VARIÁVEIS?!



- ❑ O **escopo** de uma variável é valido **dentro de uma mesma regra** ou dentro de uma **pergunta**.
  - ❑ Se a variável `X` ocorre em duas regras/perguntas, então são duas **variáveis distintas**.
  - ❑ A ocorrência de `X` **dentro** de uma mesma regra/pergunta significa **a mesma variável**.

```
% um homem(X) que gosta de uma mulher(Y) que gosta dele é feliz
feliz(X) :- homem(X),mulher(Y), gosta(X,Y), gosta(Y,X).
```



## OPERADORES DE COMPARAÇÃO

### □ Unificação/Atribuição ( $X = Y$ )

□  $=$

□  $\backslash =$

Negação

$X*f(Y) = (5-[])*P$

### □ Igualdade de termos/ igualdade literal

□  $(X == X)$

□  $==$

□  $\backslash ==$

$5+6 == 5+6$

### □ Igualdade de expressões aritméticas

□  $(1*1 \text{ :}= 3-2)$

□  $\text{ :}=$

□  $\backslash \text{ :}=$

$\text{sqrt}(4) \text{ :}= 2$

## OPERADORES DE COMPARAÇÃO

### □ % são objetos iguais

□  $1 + 2 = 1 + 2.$

### □ % são objetos iguais

□  $1 + 2 = 2 + 1.$

### □ % igualdade de valores

□  $1 + 2 \text{ :}= 2 + 1.$

### □ % calculando/unificando

□  $1 + A = B + 2.$

```

1 ?- 1 + 2 = 1 + 2.
true.

3 ?- 1 + 2 = 2 + 1.
false.

4 ?- 1 + 2 :|= 2 + 1.
true.

5 ?- 1 + A = B + 2.
A = 2,
B = 1.
  
```

## OPERADORES ARITMÉTICOS

- ❑ +
- ❑ -
- ❑ \*
- ❑ / (divisão real)
- ❑ // (divisão inteira)

```
11 ?- X is 3+2.
X = 5.

12 ?- 5 is 3+2.
true.
```

- ❑ O operador **is** força a avaliação aritmética. Ex:
  - ❑ X is 3+2. %avalia e atribui o valor a variável
  - ❑ 5 is 3+2. %avalia e compara

## EVITAR ERROS COMUNS



- ❑ Variáveis iniciam por **MAIÚSCULA**
- ❑ Nomes de predicados iniciam por **minúsculas**
- ❑ Colocar **.** depois de uma **pergunta** ou depois de uma **cláusula**.
- ❑ O nome do predicado deve ser **colado** ao abre parênteses. Ex:
  - ❑ papagaio('Ze Carioca').

## PROLOG: FATOS E REGRAS

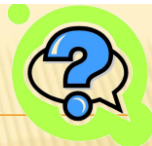
- Um predicado consiste de uma **cabeça** e um número de **argumentos**. Ex:

`gato(garfield)`



- Declara que *garfield* é um **gato**
- No caso, **gato** é a cabeça e *garfield* é o único argumento do predicado.

## PROLOG: CONSULTAS



- Exemplos:

- garfield* é um gato?

`?- gato(garfield).`  
yes.

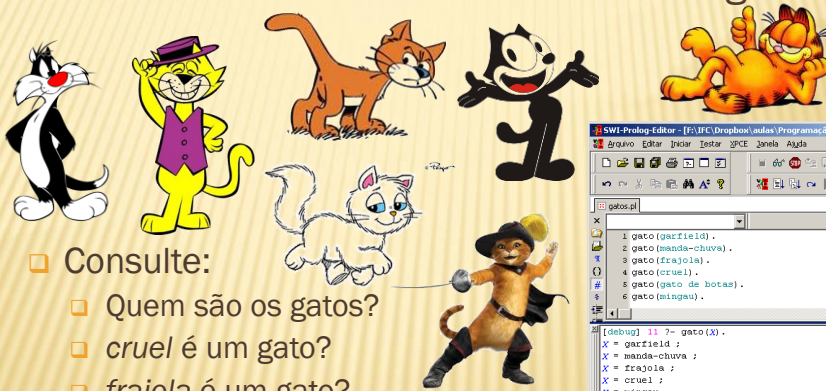
- que coisas (X) conhecidas são gatos?

`?- gato(X).`  
`X = garfield;`  
yes.

## PROLOG: FATOS E REGRAS



- Acrescente **novos fatos** sobre outros gatos:



- Consulte:
  - Quem são os gatos?
  - cruel* é um gato?
  - frajola* é um gato?
  - mingau* é um gato?
  - ...

```

1 gato(garfield).
2 gato(manda-chuva).
3 gato(frajola).
4 gato(cruel).
5 gato(gato_de_botas).
6 gato(mingau).

[debug] 11 ?- gato(X).
X = garfield ;
X = manda-chuva ;
X = frajola ;
X = cruel ;
X = gato_de_botas ;
X = mingau.

[debug] 12 ?- gato(cruel).
true.

[debug] 13 ?- gato(mingau).
false.
  
```

## PROLOG: FATOS E REGRAS



- Acrescente tais **predicados** sobre os gatos:

- preguiçoso(garfield).
- malandro(manda-chuva).
- malvado(cruel).
- malvado(frajola).
- corajoso(gato\_de\_botas).
- carinhoso(mingau).
- medroso(cruel).
- medroso(garfield).
- medroso(mingau).





## PROLOG: FATOS E REGRAS



- ❑ Sabendo que:
  - ❑ um gato é **legal** se ele for **carinhoso** e **corajoso**.
  - ❑ um gato é **chato** se for **preguiçoso** e **malandro**.
  - ❑ um gato é **arteiro** se for **malvado** e **malandro**.
  - ❑ um gato é **companheiro** se ele for **carinhoso** ou **corajoso** e não for **medroso**.
- ❑ Defina regras qualificando os gatos como:

legal

chato

arteiro

companheiro

## PROLOG: FATOS E REGRAS



- ❑ **legal(X)** :- gato(X), carinhoso(X), corajoso(X).
- ❑ **chato(X)** :- gato(X), preguiçoso(X), malandro(X).
- ❑ **arteiro(X)** :- gato(X), malvado(X), malandro(X).
- ❑ **companheiro(X)** :- gato(X),  
(carinhoso(X); corajoso(X)),  
not(medroso(X)).

## PROLOG: FATOS E REGRAS



❑ Quem são os gatos:

❑ ?- legal(X)

```
4 ?- legal(X).  
false.
```

❑ ?- chato(X)

```
5 ?- chato(X).  
false.
```

❑ ?- arteiro(X)

```
6 ?- arteiro(X).  
false.
```

❑ ?-companheiro(X)

```
7 ?- companheiro(X).  
X = gato_de_botas ;  
false.
```