

# Paradigmas de Programação

# Introdução

Linguística

Sintaxe, semântica e pragmática

Por quê estudar linguagens ?

Paradigmas de Programação

Procedural

Orientado a objetos

Funcional

Lógico

# Lingüística

Lingüística é o ESTUDO de linguagens.

Neste caso, linguagens de PROGRAMAÇÃO

De forma análoga, linguagens naturais e de programação possuem sintaxe (forma) e semântica (significado ).

Sendo a primeira mais expressiva e a segunda mais LIMITADA.

Como falado anteriormente, o estudo pode diminuir algumas limitações

# Propriedades das linguagens

Ser UNIVERSAL

devem suportar recursão ou iteração

Ser o mais NATURAL possível  
para a área que foi proposta

Ser IMPLEMENTÁVEL

Ser EFICIENTE

# Sintaxe, semântica e Pragmática

A tarefa de descrever qualquer linguagem (natural ou artificial), precisa considerar três elementos principais:

- **Sintaxe:** quais as sentenças e quais as sequências são validas em uma linguagem.
  - Normalmente é separado em léxico (identifica os elementos básicos) e sintático (identifica as sequências, como estes elementos podem ser agrupados).
- **Semântica:** qual o significado de uma da expressão ?
- **Pragmática:** como determinada linguagem é usada na prática.

# Sintaxe, semântica e Pragmática

A tarefa de descrever qualquer linguagem (na prática) precisa considerar três elementos principais:

- **Sintaxe:** quais as sentenças e quais as sequências válidas em uma linguagem.
  - Normalmente é separado em léxico (identifica os elementos básicos) e sintático (identifica as sequências, como estes elementos podem ser agrupados).
- **Semântica:** qual o significado de uma dada expressão ?
- **Pragmática:** como determinada linguagem é usada na prática.

*Conceitualmente  
léxico faz parte  
da sintaxe.*

# Sintaxe, semântica e Pragmática

A **sintaxe** influencia como os programas são escritos pelos programadores, lidos por outros programadores e pelo compilador.

**Semântica** determina como os programas são compostos pelo programador, entendido por outros programadores e interpretado pelo computador.

**Pragmatismo** influencia como os programadores projetam e implementam um dado problema na prática, como ele usa os elementos da linguagem.

**Por que estudar linguagens de programação ?**



# Por que estudar linguagens de programação?

“O aspecto mais importante, mas também o mais elusivo, de qualquer ferramenta é a sua influência nos hábitos daqueles que se treinam no seu uso. Se a ferramenta é uma linguagem de programação essa influência é, gostemos ou não, uma influência em nosso hábito de pensar”

Edsger W.  
Dijkstra

“The magic of computer programming doesn't come from using any particular tool, computer, or language. The real magic of programming comes from applying your own imagination and using programming as a means to achieve whatever you want to create.”

Wallace Wang  
Beginning Programming for Dummies, 4th

# Por que estudar linguagens de programação?

- Os principais benefícios no estudo de LPs são os seguintes:
  - Melhoria na capacidade de expressar idéias
  - Melhoria na capacidade de escolher linguagens apropriadas
  - Mais habilidade para aprender novas linguagens
  - Melhor entendimento da dificuldade de implementação das construções das LPs
  - Melhor uso das linguagens conhecidas
  - Avanço da área de computação de um modo geral

# Melhoria na capacidade de expressar idéias

Existe uma relação entre os pensamentos e a linguagem utilizada para comunicar esses pensamentos.

É difícil para as pessoas conceitualizarem estruturas que elas não podem descrever verbalmente ou na forma escrita

- Particularmente em termos das abstrações que podem manipular

Melhoria na capacidade de expressar idéias

Programadores também sofrem restrições durante o desenvolvimento de software

A linguagem que eles utilizam limitam os tipos de estruturas de controle, estruturas de dados e abstrações que podem ser usadas.

O conhecimento de uma variedade mais ampla de características de LPs pode reduzir tais limitações

# Melhoria na capacidade de escolher LPs apropriadas

Muitos programadores possuem pouca educação formal nas LPs que trabalham

Muitas vezes as linguagens que aprenderam não são mais usadas e muitas características de novas LPs não existiam na época

- Ex.: De Pascal, COBOL, C para Delphi, C++, Java, LPs p/ Web

# Melhoria na capacidade de escolher LPs apropriadas

- Desta forma muitas pessoas **escolhem sempre a mesma linguagem** para a resolução de problemas, o que não é uma boa estratégia
  - O conhecimento de uma variedade mais ampla de LPs e construções de outras LPs pode ajudar na seleção da linguagem **mais apropriada para o problema em mãos.**
- Mesmo que algumas das novas construções possam ser simuladas, nem sempre tais soluções são boas...
  - Ex.: Classes em C

# Mais habilidade para aprender novas linguagens

O processo de aprender uma nova LP normalmente é difícil e demorado

Aprender os principais conceitos de LPs ajuda a aprender novas linguagens.

Ex.: Programadores que conhecem os conceitos OO terão mais facilidade para aprender Java ou C++

# Mais habilidade para aprender novas linguagens

O mesmo ocorre com linguagens naturais

Quanto melhor você conhece a gramática de sua língua nativa, mais fácil será para você aprender uma segunda língua.

Adicionalmente, aprender uma segunda língua ensina a vc mais coisas sobre a primeira.



# Melhor entendimento da dificuldade de implementação das LPs

Ao projetar e usar LPs é interessante saber como as construções são implementadas

Isso pode ajudar a selecionar as construções mais eficientes para cada caso.

Não é o foco dessa disciplina discutir profundamente a implementação das construções de LPs

Esse é o foco da cadeira de Compiladores

# Melhor uso das linguagens conhecidas

Muitas das LPs atuais são grandes e complexas

É incomum um programador conhecer e usar todas as características de uma LP

O estudo de CLPs pode ajudar a aprender conceitos previamente desconhecidos e não usados da LP usada pelo programador.

# Paradigmas de programação

- De acordo com execução
  - Seqüencial e
  - Concorrente
- De acordo com a programação
  - Imperativo e
  - Declarativo

# SEQÜÊNCIAL E CONCORRENTE

## Seqüencial

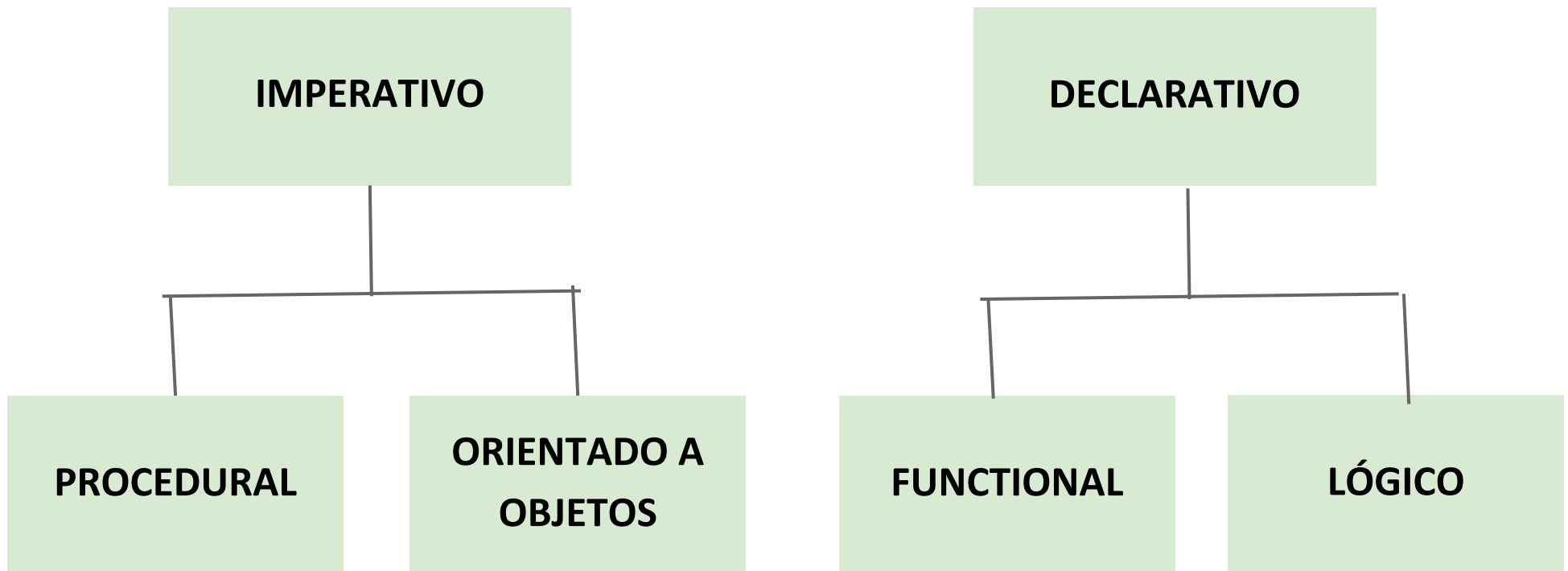
- Uma computação é realizada após o término da anterior
- Controle de fluxo de execução interno ao programa:
  - seqüência
  - seleção
  - iteração
  - invocações

## Concorrente

- Múltiplas computações podem ser executadas simultaneamente
- Computações paralelas
  - múltiplos processadores compartilham memória
- Computações distribuídas
  - múltiplos computadores conectados por uma rede de comunicação

# Paradigmas de programação

Modelo, padrão ou estilo de programação suportado por linguagens que agrupam certas características comuns



# IMPERATIVO E DECLARATIVO

## Modelo imperativo

- Linguagens expressam seqüências de comandos que realizam transformações sobre dados
- Base: máquina de von Neumann
  - orientadas a procedimentos
  - orientadas a objetos

## Modelo declarativo

- Linguagens que não possuem os conceitos de
  - seqüências de comandos
  - atribuição
- linguagens funcionais: ênfase em valores computados por funções
- linguagens lógicas: ênfase em axiomas lógicos

# PARADIGMAS DE PROGRAMAÇÃO

- Percentual de uso de acordo com o Tiobe index.

Category	Ratings Mar 2010	Delta Mar 2009
Object-Oriented Languages	53.6%	-2.7%
Procedural Languages	41.8%	+2.5%
Functional Languages	2.9%	-0.4%
Logical Languages	1.7%	+0.6%

# PARADIGMA IMPERATIVO

- Baseado fortemente na maquina de Turing e na arquitetura de Von Neuman (1947):

- memória;
- processador;
- dispositivos de entrada e saída.



- Primeiro paradigma a surgir e até hoje é o dominante.





# PARADIGMA IMPERATIVO

- Um programa neste paradigma é uma seqüência finita de instruções (ou comandos) de três tipos: atribuição, controle de fluxo, ou entrada/saída de dados.
- Comandos de atribuição alteram o “estado” do programa.

# PARADIGMA IMPERATIVO

```
int fatorial( int n ) {  
    int fat = 1;  
    while ( n > 1) {  
        fat = fat * n;  
        n = n - 1;  
    }  
    return fat;  
}
```

# SEMÂNTICA DA EXECUÇÃO

- A semântica da execução de um programa imperativo consiste numa seqüência de estado de memória, onde cada estado é obtido do anterior por uma operação de atribuição.
- A seqüência de estados é determinística, no sentido de que não há escolha possível de caminhos alternativos.
- O estado de um programa em determinado momento é o conteúdo da sua memória neste momento e pela instrução corrente.

# PARADIGMA IMPERATIVO

Vantagens?

- Eficiência
- Paradigma dominante e bem estabelecido
- Método “receita de bolo”

Desvantagens?

- difícil legibilidade
- descrições demasiadamente operacionais
- focalizam o como e não o quê

**O que vocês acham?**

# PARADIGMA IMPERATIVO - PROCEDURAL

Paradigma procedural emprega o conceito de funções e procedimentos:

**Procedimentos** são coleções de sentenças que definem computações parametrizáveis

- Podem modificar os parâmetros
- Podem modificar variáveis globais

**Funções** são similares a procedimentos, mas são semanticamente modeladas como funções matemáticas

- Não deveriam ter efeitos colaterais
- Na prática, são intercambiáveis...

# PROCEDIMENTOS PARA OBJETOS: TAD

## Tipos Abstratos de Dados (TAD)

- Conceito matemático que diz respeito a determinada entidade e às funções aplicadas sobre ela
- Um TAD é definido pela sua funcionalidade: “o que” se pode fazer com ele, e não “como” ele está de fato implementado
- Exemplo uma fila é um tipo abstrato de dados: pode-se inserir e retirar elementos de uma fila, verificar se a fila está cheia ou vazia, ou ainda contar quantos elementos estão nesta fila
- A utilização da fila (por uma aplicação) através destas funções independe de como a fila está de fato implementada



John Guttag, grande contribuição na formalização dos TAD.

J.V. Guttag, *The Specification and Application to Programming of Abstract Data Types*, Ph.D. Thesis, Dept. of Computer Science, University of Toronto (1975).

# PROGRAMAÇÃO MODULAR

- A programação modular implementa a noção de tipo abstrato de dados
- Implementação: encapsulamento de dados e funções na mesma unidade sintática – o módulo
- A principal estrutura é um módulo, constituído de uma interface e de uma implementação
- A interface contém todos os elementos visíveis (importáveis) por outros módulos
- A implementação contém os elementos que devem ficar invisíveis e as implementações das funções e procedimentos do módulo
- Exemplos de linguagens: Ada (DoD-USA), Modula-2 (Nicklaus Wirth).

# PROGRAMAÇÃO MODULAR

## Outros exemplos:

- Pilha
  - operações: pop, push, top
- Conta bancária
  - operações: depositar, retirar, tirar extrato, verificar saldo
- Agenda
  - operações: inserir, remover, alterar
- Figura
  - operações: desenhar, mover, rotacionar, colorir



# Orientação a objetos

- Evolução da programação modular, incorporando duas características fundamentais: herança e polimorfismo
- TAD → Módulo → **Classes de Objetos**
  - Classes são estruturas que agrupam características (atributos) e funcionalidades (métodos) comuns a um grupo de objetos
  - Uma variável de uma classe é denominada atributo, e uma função é denominada método
- Não existe um estado global do sistema: todas as variáveis e funções são locais aos objetos, que se comunicam entre si através de troca de mensagens (chamada de procedimentos/funções)

# Orientação a objetos



A word cloud featuring various concepts in Object-Oriented Programming (OOP). The words are arranged in a circular pattern, with 'classe' and 'objeto' being the most prominent. Other words include 'relação', 'composição', 'subclasse', 'herança', and 'polimorfismo'. The colors used are purple, blue, and red.

classe  
relação  
objeto  
composição  
subclasse  
herança  
polimorfismo

# Orientação a objetos

Um sistema é um conjunto de objetos e suas relações.



# Orientação a objetos

- Não é um paradigma no sentido estrito: é uma subclassificação do imperativo
- A diferença é mais de metodologia quanto à concepção e modelagem do sistema
- A grosso modo, uma aplicação é estruturada em módulos (classes) que agrupam um estado (atributos) e operações (métodos) sobre este
- Classes podem ser estendidas e/ou usadas como tipos (cujos elementos são objetos)

# Orientação a objetos: Conceitos chaves

- A classe é uma estrutura de onde se deriva vários objetos.
- Os objetos são as entidade criadas a partir de uma classe.
- Um objeto é uma “instância” de uma classe.
- Classes existem na modelagem enquanto os objetos em execução.
- A composição é um poderoso mecanismo de extensão.

# PROGRAMAÇÃO FUNCIONAL

Expressão

Recursão ~ Avaliação

Função

# PROGRAMAÇÃO FUNCIONAL

- Linguagens de programação implementam mapeamentos
- Programação imperativa:
  - Mapeamento através de comandos que lêem valores de entrada, os manipulam e escrevem valores de saída
  - Variáveis têm papel fundamental
- Em programação funcional:
  - Mapeamento ocorre através de funções
  - Baseado em **funções matemáticas**

# FUNÇÕES MATEMÁTICAS

- Correspondência biunívoca de membros do conjunto domínio para membros do conjunto imagem
- Ordem de avaliação de suas expressões é controlada por expressões condicionais e por recursão por expressões condicionais e por recursão.
  - Não pela seqüência ou pela repetição iterativa
- Não têm efeitos colaterais
  - Sempre definem o mesmo valor dado o mesmo conjunto de argumentos, diferentemente de um procedimento em linguagens imperativas.



# PROGRAMAÇÃO FUNCIONAL

- Definição de função

- Nome + lista de parâmetros + expressão de correspondência
- $\text{cubo}(x) = x * x * x$
- Um elemento do conjunto imagem é obtido para cada par:  
Nome da função + um elemento particular do conjunto domínio
  - $\text{cubo}(2.0) = 8.0$

- Definição de uma função separada da tarefa de nomeá-la

- Notação lambda (Church, 1941)
- $\lambda(x) x * x * x$

# PROGRAMAÇÃO FUNCIONAL

- Não existem variáveis
  - Expressões
- Não existem comandos
  - Funções
- Não existem efeitos colaterais
  - Declarações
- Não há armazenamento
  - Funções de alta ordem
  - Lazy evaluation
  - Recursão
- Fluxo de controle: expressões condicionais + recursão

# FUNÇÕES

Uma função é um mapeamento de valores de um tipo em outro tipo

`not :: Bool -> Bool`

`isDigit :: Char -> Bool`

`add :: (Int,Int) -> Int`

`add x y = x + y`

# APLICAÇÃO DE FUNÇÕES EM HASKELL

## Matemática

$f(x)$

$f(x,y)$

$f(g(x))$

$f(x, g(y))$

$f(x) g(y)$

$f(a,b) + c d$

## Haskell

`f x`

`f x y`

`f (g x)`

`f x (g y)`

`f x * g y`

`f a b + c * d`

# SOMA DE 1 A 100 EM C++

- Operação baseada em atribuição de variáveis

```
int total = 0;
```

```
for (int i = 1; i <= 100; i++)
```

```
    total = total + i;
```

- A mesma variável (total) muda de valor 100 vezes durante a operação (efeito colateral)
- Como programar sem efeitos colaterais?

# CARACTERÍSTICAS DO HASKELL

- Lazy avaluation, avaliação preguiçosa ou chamada por nome.
- **Funções como valores de primeira classe**, ou funções de alta ordem.
- Funções e tipos paramétricos;

# HASKELL – LAZY EVALUATION

- **Estratégia *call-by-value*** --- Avalia primeiro o argumento antes de aplicar a função (Pascal, C, Java, etc).

$$\begin{aligned} & (\backslash x \rightarrow x+1) (2+3) \\ &= (x+1)+5 \\ &= 5+1 \\ &= 6 \end{aligned}$$

- **Estratégia *call-by-name*** (ou *lazy evaluation*) --- Aplica imediatamente a função ao argumento, adiando para mais tarde a avaliação desse argumento (Haskell e Miranda)

$$\begin{aligned} & (\backslash x \rightarrow x+1) (2+3) \\ &= (2+3)+1 \\ &= 5+1 \\ &= 6 \end{aligned}$$

# FUNÇÕES VALORES DE PRIMEIRA CLASSE

- Significa que as funções têm um estatuto tão importante como o dos inteiros, reais, e outros tipos predefinidos. Concretamente, numa linguagem funcional as funções podem:
  - Ser passadas como argumento para outras funções;
  - Podem ser retornadas por outras funções;
  - Podem ser usadas como elementos constituintes de estruturas de dados;

```
Prelude>map (\x->2*x) [1,2,3]  
[2,4,6]
```



## PARADIGMA LÓGICO

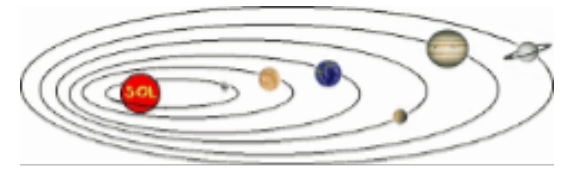
Lógico

Regras  
Unificação

Fatos  
Recurso

Pergunta

# FATOS, REGRAS E PERGUNTAS



- Um fato é uma afirmação **sempre verdadeira**.

A Terra é um planeta.  
O Sol é uma estrela.



- Uma regra é uma afirmação que para ser verdadeira **depende de outras regras** ou fatos.

A Terra é um planeta **se** ela não  
for uma estrela.



- Podemos fazer **perguntas** sobre os **fatos ou regras**.

A Terra é um planeta ?  
Quem é um planeta ?  
A Terra é uma estrela ?



# PARADIGMA LÓGICO

- Programa-se de forma declarativa, ou seja, especificando **o que** deve ser computado ao **invés de como** deve ser computado
- Sem instruções explícitas e seqüenciamento
- Aplicações em IA, robotica, linguagem natural

# PARADIGMA LÓGICO

Programar em lógica envolve:

1. declarar alguns  **fatos**  a respeito de objetos e seus relacionamentos,
2. definir algumas  **regras**  sobre os objetos e seus relacionamentos e
3. fazer  **perguntas**  sobre os objetos e seus relacionamentos.

# LÓGICA E PROLOG

- **Prolog** = **Pro**gramming in **Log**ic.
- Principal linguagem do paradigma lógico , proposto na década de 70.
  - Robert Kowalski (Edinburgh)
  - Maarten van Emden (Edinburgh)
  - Alan Colmerauer (Marseilles)

# FATOS EM PROLOG

## Fatos

O Sol é uma estrela.  
A Terra é um planeta.

estrela(sol).  
planeta(terra).

## Notação geral de fatos.

predicado(arg1, arg3, ..., argn ).

Não pode ter  
espaço entre  
predicado e  
parênteses.

Predicados e  
argumentos são  
escritos por letras

Um fato ou regra deve  
terminar por ponto

# FATOS EM PROLOG

- Exemplos de fatos, com um argumento

estrela(sol).

planeta(terra).

planeta(marte).

satelite(lua).

- Exemplos de fatos, com mais de um argumento

**Verbo Sujeito e Objeto**

pai(joao, ana).



João é pai de  
Ana.

filho(jose, maria, antonio).



José é filho de Maria e  
Antonio

# PERGUNTAS EM PROLOG

Consultas, perguntas, queries ou goals.

A Terra é um planeta ?

A Terra é uma estrela ?

O sol é um planeta ? ...

?- planeta(terra).

*true.*

?- estrela(terra).

*false.*

A “hipótese do mundo fechado”.

% base de dados

planeta(terra).

% goals

?- planeta(marte).

*false.*



# PERGUNTAS EM PROLOG

Consultas, perguntas, queries ou goals.

A Terra é um planeta ?  
A Terra é uma estrela ?  
O sol é um planeta ? ...

?- planeta(terra).

*true.*

?- estrela(terra).

*false.*

Um predicado  
sempre retorna  
um valor lógico,  
verdadeiro ou  
falso.

A “hipótese do mundo fechado”.

% base de dados  
planeta(terra).

% goals

?- planeta(marte).

*false.*

# PERGUNTAS EM PROLOG

Consultas, perguntas, queries ou goals.

A Terra é um planeta ?  
A Terra é uma estrela ?  
O sol é um planeta ? ...

?- planeta(terra).

*true.*

?- estrela(terra).

*false.*

Um predicado  
sempre retorna  
um valor lógico,  
verdadeiro ou  
falso.

A “hipótese do mundo fechado”.

% base de dados  
planeta(terra).

% goals

?- planeta(marte).

*false.*

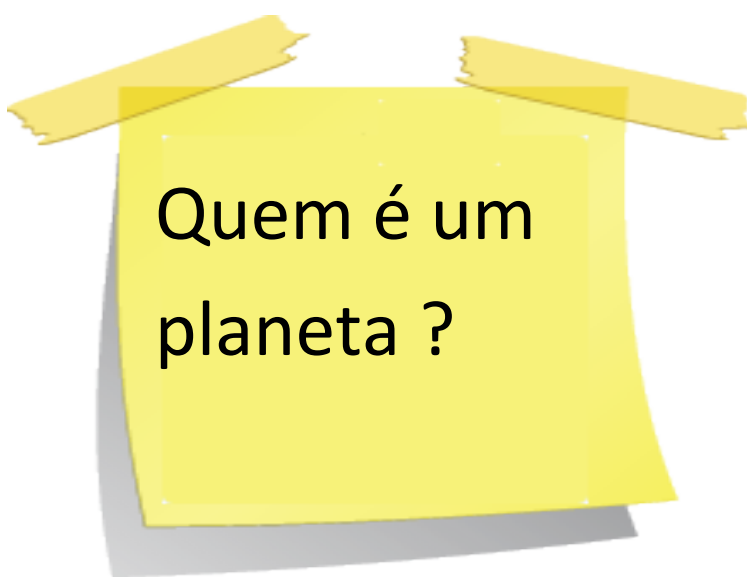
se um fato não é  
conhecido ,  
assume-se que  
ele é falso.

# PERGUNTAS EM PROLOG

Consultas, perguntas, queries ou goals.

A Terra é um planeta ?  
A Terra é uma estrela ?  
O sol é um planeta ? ...

?- planeta(terra).  
*true.*  
?- estrela(terra).  
*false.*



Quem é um  
planeta ?

# VARIÁVEIS EM PROLOG

Variáveis em Prolog é como uma incógnita que pode ser encontrada por inferência (deduções lógicas) aos fatos e regras:

?- planeta(X).

*X = terra.*

Dedução do  
interpretador

Variável  
X

*Toda variável  
começa com  
letra maiúscula*

# REGRAS OU CLAUSULAS

A Terra é um planeta **se** ela não for uma estrela.

```
planeta(terra) :-  
not(estrela(terra)).
```

irmao(X,Y) :- filho(X, H,M) , filho(Y,H,M) , Y \== X.

Pé da  
sentença (.)

Cabeça,  
definição do  
predicado

Pescoço,  
equivale ao "if"  
ou "se".

Corpo,  
equivale a  
uma função.

Uma virgula (,) equivale ao  
operador "e" e um ponto e  
virgula (;) ao operador "ou".

The diagram illustrates the structure of a Prolog clause. The clause 'irmao(X,Y) :- filho(X, H,M) , filho(Y,H,M) , Y \== X.' is shown. Annotations with arrows point to different parts: 'Cabeça, definição do predicado' points to 'irmao(X,Y)'; 'Pescoço, equivale ao "if" ou "se".' points to the colon-hyphen separator ':-'; 'Corpo, equivale a uma função.' points to the first predicate 'filho(X, H,M)'; 'Uma virgula (,) equivale ao operador "e" e um ponto e virgula (;) ao operador "ou".' points to the comma separator between 'filho(X, H,M)' and 'filho(Y,H,M)'; 'Pé da sentença (.)' points to the period at the end of the clause.