

Youtube

▼ Playlist das aulas

- <https://www.youtube.com/playlist?list=PLZ-Bk6jzsb-OScKa7vhpcQXoU2uxYGaFx>

▼ Tipos de dados

▼ Variáveis - **Teste**, **_Teste**

- Iniciadas com maiúsculas ou underscore (_), seguidos de qualquer caractere alfanumérico
- Somente underscore define uma variável anônima
- Ex: **X**, **Y1**, **_Nome**

▼ Átomos - **teste**, **'Teste'**

- São constantes, devem ser iniciadas com minúsculas seguidas de qualquer caractere alfanumérico, ou qualquer sequência entre ' ' (aspas simples)
- Ex: **joao**, **'João'**, **'16'**

▼ Inteiros - **18**, **"t"**

- Qualquer sequência numérica que não contenha ponto (.)
- Caracteres ASCII entre " " (aspas duplas) são tratados como listas de inteiros
- Ex: **1**, **6**, **-1**, **"a"**

▼ Floats - **2.5**

- Números com um ponto (.) e pelo menos uma casa decimal
- Ex: **5.3** (correto), **7**. (incorreto)

▼ Listas - **[t, e, s, t]**

- Sequência ordenada de elementos entre [] e separados por vírgulas
- Ex: **[a, b, c]**, **[a | b, c]**

▼ Sintaxe

- **Write('Termo').** ou **Write(variável).** - Exibir mensagem
- **Read(Variável).** - Ler informação

▼ Caracteres especiais

- **nl, \n, \l** - Nova linha
- **\r** - Retorna ao início da linha
- **\t** - Tabulação
- **\%** - Imprimir símbolo %

▼ Comentários

- **%** - Linha inteira
- **/* */** - Todo o texto entre os símbolos

▼ Fatos

- Um programa Prolog é uma coleção de fatos e regras
- Fatos são sempre verdadeiros, mas regras precisam ser avaliadas
- Como criar um fato em uma base Prolog:
 - **homem(x).** - significa que "x é um homem"
 - **genitor(x, y).** - significa que "x é genitor de y" ou "y é gerado de x"
- É responsabilidade do programador definir os predicados corretamente
- Predicado é uma declaração que deve ser verdadeira ou falsa dependendo do valor de suas variáveis

▼ Consultas

- A cláusula **proximo(Brasil, Japao).** é uma consulta Prolog, pois "**Brasil**" e "**Japão**" são variáveis
- Para responder consultas o Prolog utiliza:
 - **Matching** - checa se determinado padrão está presente, para saber quais fatos e regras podem ser utilizados
 - **Unificação** - substitui o valor de variáveis para determinar se a consulta é satisfeita pelos fatos ou regras da base (programa)

- **Resolução** - verifica se uma consulta é consequência lógica dos fatos e regras da base (programa)
- **Recursão** - utiliza regras que chamam a si mesmas para realizar demonstrações
- **Backtracking** - para checar todas as possibilidades de resposta

▼ Exemplo

Consultas

Exemplo das etapas na consulta...

```

homem(tom). % fato
mulher(pam). % fato
genitor(pam, bob). % fato
genitor(tom, bob). % fato

?- genitor(tom, X). % tom é genitor de quem (X)?
X = tom ;      <- Unifica [X/bob]
No             <- genitor(tom, bob). (matching)
               <- busca outras soluções (backtrack)
               <- Nenhum outro fato satisfaz a consulta
          
```

```

?- genitor(X, bob). %Quem (X) é/são o(s) genitor(es) de bob?
X = tom ;      <- Unifica [X/tom]
X = pam ;      <- genitor(tom, bob). (matching)
No             <- busca outras soluções (backtrack).
               <- Unifica [X/pam]
               <- genitor(pam, bob). (matching)
               <- busca outras soluções (backtrack).
               <- Nenhum outro fato satisfaz a consulta.
          
```

▼ Regras

- Facilitam a execução de consultas e tornam um programa muito mais expressivo
- Uma cláusula Prolog é equivalente à uma fórmula em lógica de 1ª ordem, então, em Prolog, existem os conectivos
 - **:- (se)**, equivalente à implicação
 - **,** (**e**), equivalente à conjunção
 - **;** (**ou**), equivalente à disjunção
- Exemplo:
 - A fórmula: $A(x) \rightarrow B(x) \vee (C(x) \wedge D(x))$
 - Seria escrita em Prolog como: **a(X) :- b(X); (c(x), d(x))**
- Prolog não utiliza quantificadores explicitamente, porém, trata todas as regras como se elas estivessem universalmente quantificadas e usa ~ EU (eliminação do universal)

- Consultas são realizadas sobre regras do mesmo modo como ocorrem sobre fatos
- Uma regra se divide em conclusão (ou cabeça) e condição, da seguinte forma:
 - **CONCLUSÃO(ARG) :- CONDIÇÃO1(ARG) CONECTIVO CONDIÇÃO2(ARG)...**
- Utilizando **matching**, Prolog encontra quais regras podem ser utilizadas para satisfazer uma consulta
- Cada vez que um **matching** ocorre a satisfação da regra passa a ser a meta atual
- Exemplo:
 - Regra: **prole(X, Y) :- genitor(Y, X)**
 - Consulta: **?- prole(pam, bob)**

▼ Manipulação da base de conhecimento

- A princípio, os predicados carregados pela instrução **consult** na base de conhecimento são estáticos
- Porém, existem predicados pré-definidos que permitem fazer a manipulação da base de conhecimento
- Ou seja, predicados que permitem acrescentar e/ou retirar fatos e regras da base de conhecimento durante a execução de um programa
- Para criar predicados dinâmicos é necessário utilizar a diretiva:
 - **:- dynamic nomedopredicado/aridade.**
- Contudo, as alterações são voláteis (não alteram o arquivo original)
- **listing(fato/aridade).** - lista todos os fatos/regras presentes
- **assert(fato(variável)).** - acrescenta o fato/regra como último item do predicado
- **asserta(fato(variável)).** - acrescenta o fato/regra como primeiro item do predicado
- **retract(fato(variável)).** - remove da base de conhecimento a primeira cláusula (fato ou regra) que corresponde ao termo passado como

parâmetro

- **retractall(fato(variável)).** - remove da base de conhecimento todos os fatos ou regras cuja cláusula (fato ou regra) corresponde ao termo que é passado como parâmetro
- **abolish(fato/aridade)** - remove da base de conhecimento todos os fatos e regras pelo nome da regra ou fato/aridade que é passada como parâmetro (são removidos predicados estáticos também)
- **abolish(fato, aridade)** - semelhante a abolish/1, mas passando o nome do fato/regra e a sua aridade separadamente (são removidos predicados estáticos também)

▼ Aritmética / Operadores

▼ Tabela - Operadores aritméticos e relacionais

Átomos e Números

❑ Os números usados em Prolog incluem **inteiro** e **real**:

Operadores Aritméticos	
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Divisão inteira	//
Resto divisão inteira	mod
Potência	**
Atribuição	is

Operadores Relacionais	
$X > Y$	X é maior que Y
$X < Y$	X é menor que Y
$X \geq Y$	X é maior ou igual a Y
$X \leq Y$	X é menor ou igual a Y
$X =:= Y$	X é igual a Y
$X = Y$	X unifica com Y
$X \neq Y$	X é diferente de Y

- Podemos utilizar duas notações para representar expressões em Prolog
 - Infixa: $2 * a + b * c$
 - Prefixa: $+(*(2, a), *(b, c))$
- Principais operadores de cálculo

Operador	Significado
+, -, *, /	Realizar soma, subtração, multiplicação e divisão, respectivamente
is	atribui uma expressão numérica à uma variável
mod	Obter o resto da divisão
^	Calcular potenciação
cos, sin, tan	Função cosseno, seno e tangente, respectivamente
exp	exponenciação
ln, log	logaritmo natural e logaritmo
sqrt	raiz

- Predicados de conversão
 - **integer(X).** - converte X para inteiro
 - **float(X).** - converte X para ponto flutuante
- Predicados de comparação

Operador	Significado
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
:=	Igual
\=	diferente
\+	Negação – retorna sucesso se o predicado for falso e vice-versa.

- Os operadores =, == e := realizam diferentes tipos de comparação
 - = - checa se os "objetos" são iguais, ou atribui valores para as variáveis (unificação de termo)
 - == - avalia a "igualdade simbólica" entre os termos
 - := - avalia se os valores são iguais (comparação)

▼ Recursão / Regras recursivas

- Uma regra é recursiva se sua condição depende dela mesma, tal como:
 - **a(X) :- b(X), a(X).**
- Um conjunto de regras com o mesmo nome é denominado procedimento ou predicado

▼ Comandos de Corte

- O **retrocesso (backtracking)** verifica todas as alternativas de solução
- É possível controlar o **retrocesso** através de um predicado especial chamado **corte**, denotado por **!** (exclamação)
- Visto como uma cláusula, seu valor é sempre verdadeiro
- Sua função é provocar um efeito colateral que interfere no processamento padrão de uma consulta
- **"Corte verde"** - se **!** retirado, não afeta a lógica do funcionamento
- **"Corte vermelho"** - se **!** retirado, afeta a lógica do funcionamento
- Usar um **Corte** na última cláusula (caso seja verdadeiro) faz com que o código não "retroceda/busque" os próximos procedimentos de mesmo nome
- Usar um **Corte** em alguma cláusula do meio faz com que o código "trave" os valores que já foram definidos antes (sem permitir que "retroceda/busque" outros)

```
m(1).
m(2) :- !.
m(3).
m1(X, Y) :- m(X), m(Y).
m2(X, Y) :- m(X), !, m(Y).
/*
Respostas
m(X). - 1, 2
m1(X, Y). - (1, 1), (1, 2), (2, 1), (2, 2)
m2(X, Y). - (1, 1), (1, 2)
m(3). - True
*/
```

```
?- m(X).
X = 1 ;
X = 2.

?- m1(X, Y).
X = Y, Y = 1 ;
X = 1,
Y = 2 ;
X = 2,
Y = 1 ;
X = Y, Y = 2.

?- m2(X, Y).
X = Y, Y = 1 ;
X = 1,
Y = 2.
```

▼ Comandos de Fail e Repeat

▼ Fail

- A cláusula **fail** faz com que o programa retorne uma falha
- Sendo assim, ele não para a execução e busca outras respostas

- Forçando então um **backtracking**

```

1  aluno(marcelo).
2  aluno(andre).
3  aluno(roberto).
4
5  escreverSemFail :- aluno(X), write(X).
6  escreverComFail :- aluno(X), write(X), nl, fail.
7

```

```

?- escreverSemFail.
marcelo
true .

?- escreverComFail.
marcelo
andre
roberto
false.

```

▼ Repeat

- A cláusula **repeat** força uma repetição até a cláusula inteira ser verdadeira ou encontrar um **corte**
- Sendo assim, ele pode manter a execução de maneira indeterminada (Looping infinito)
- É utilizado com a Entrada e Processamento de dados, com uma condição no final que para a execução ao ser verdadeira

```

adivinhe_numero :-
    N is random(5) + 1, % gera de 0 a 4, ao somar com 1 se torna de 1 a 5
    repeat,
        lerDados(G),
        processarDados(G, N).

lerDados(G) :- write('Digite um número (1 até 5): '),
               read(G).

processarDados(G, N) :- G == N, write('Parabéns, você acertou!'), nl.
processarDados(G, N) :- G \= N, write('Você errou!'), nl, fail.

```

▼ Listas

- É uma sequência ordenada de elementos de qualquer tipo de dados de Prolog
- Os elementos contidos em uma lista devem ser separados por vírgulas, e precisam estar entre colchetes
- Existem notações alternativas, porém, esta é a mais simples
 - **[pam, liz, pat, ann, tom, bob, jim]**
 - **[1, 2, 3, 4, 5]**
 - **[a, [b, c], d, e]** - onde **[b, c]** é o segundo elemento da lista
- Listas podem ser de dois tipos:
 - **Vazias** - quando não contém nenhum elemento, representadas por **[]**
 - **Não-vazias** - quando contém ao menos um elemento
- Toda lista tem uma lista vazia dentro dela (o último elemento)
- Listas **não-vazias** possuem duas partes, são elas
 - **Cabeça (head)** - corresponde ao primeiro elemento da lista (representado apenas pelo próprio elemento)
 - **Cauda (tail)** - corresponde aos elementos restantes da lista (representado por uma lista com os elementos restantes)
- Operador **Pipe (|)** - Uma lista não vazia pode ser representada de uma forma a apresentar explicitamente sua cabeça e sua cauda, usando a sintaxe **[Cabeça | Cauda]**
- Essa sintaxe é útil em consultas quando queremos decompor uma lista em cabeça e cauda

```
% lista [b]
?- L = [b|[]]

% lista [a, b]
?- L = [a|[b|[]]]
```

```
?- [Head|Tail] = [mia, vincent, jules, yolanda].
Head = mia,
Tail = [vincent, jules, yolanda]

?- [X,Y | W] = [[], dead(zed), [2, [b, chopper]], [], Z].
X = [],
Y = dead(zed),
W = [[2, [b, chopper]], [], Z]
```

- Unificação de listas (Lista1 = Lista2)

Lista 1	Lista 2	Unificação
[mesa]	[X Y]	X/mesa Y/[]
[a,b,c,d]	[X,Y Z]	X/a Y/b Z/[c,d]
[[ana,Y] Z]	[[X,foi],[ao,cinema]]	X/ana Y/foi Z/[[ao,cinema]]
[ano,bissexto]	[X,Y Z]	X/ano Y/bissexto Z/[]
[ano,bissexto]	[X,Y,Z]	não unifica
[data(7,Z,W),hoje]	[X Y]	X/data(7,Z,W) Y/hoje
[data(7,W,1993),hoje]	[data(7,X,Y),Z]	X/W Y/1993 Z/hoje