Linguagens de Programação e a IA

Conteudista: Prof. Me. Manuel Fernandez Paradela Ledón
Revisão Técnica: Prof. Me. Douglas Almendro
Revisão Textual: Prof.ª M.ª Natalia Conti

Objetivos da Unidade:

- Conhecer as características básicas das linguagens de programação PROLOG e LISP;
- Resolver problemas básicos de representação e processamento de conhecimento utilizando as linguagens de programação do paradigma lógico e funcional.

📄 Material Teórico

Por que PROLOG, LISP e a Inteligência Artificial?

As linguagens de programação PROLOG e LISP são consideradas interessantes dentro da IA porque permitem representar, modificar e processar conhecimento em diferentes áreas e situações.

Você poderá verificar que as linguagens PROLOG e LISP oferecem possibilidades para o processamento simbólico (lembre o enfoque simbólico da IA, o sistema de símbolos físicos e a hipótese de Newell e Simon e as diferenças entre os enfoques simbólico e conexionista).

Nesta unidade estudaremos as principais características das linguagens de programação LISP e PROLOG, apresentaremos bastantes exemplos e deixaremos exercícios propostos para você praticar.

Veja na sugestão de materiais complementares as referências a duas versões gratuitas destas linguagens, que você poderá baixar para testar os exemplos deste material e seus próprios programas.

A Linguagem de Programação PROLOG

Como sabemos, uma linguagem de programação estabelece regras sintáticas, regras semânticas, símbolos, operadores a serem utilizados, etc. para escrever programas corretamente, ou seja, para programar as operações que serão executadas pelo computador.

Existem muitas linguagens de programação e também inúmeras classificações das linguagens, por exemplo:

- Paradigmas de programação: programação procedimental, programação orientada a objetos, programação lógica (Ex. PROLOG), programação funcional (Ex. LISP);
- Linguagens universais ou super-linguagens, linguagens "*assembly*", código ou linguagem de máquina, etc.;
- Linguagens de 1ª geração, ..., 5ª geração, etc.;
- Programação descritiva, programação prescritiva;
- Linguagem natural/linguagem artificial... outras...

A linguagem PROLOG surgiu no ano de 1972 (Robert Kowalski, como teórico e Alan Colmerauer na implementação) na França e na Inglaterra. PROLOG significa"*programming in logic*".

PROLOG é uma linguagem de programação baseada na lógica de predicados de primeira ordem, e por este motivo é considerada uma linguagem dentro do paradigma da programação lógica.

Em princípio, em PROLOG não programaremos como achar a resposta (como nas linguagens prescritivas), porque este procedimento é um mecanismo interno do PROLOG. Com PROLOG descreveremos um universo (programação descritiva), as características do problema, as relações, os fatos, definições. Estaremos preocupados na descrição de um problema e não em como resolvê-lo.

Podemos considerar que o estilo da programação em PROLOG é uma descrição de um universo, que a descrição desse universo está composta por fatos e regras eque essa descrição poderá ser "consultada".

Podemos interrogar o programa, fazer perguntas ou consultas sobre esse universo. Veja a Figura a seguir.

Figura 1

Um programa PROLOG pode ser considerado um banco de dados, porque ele contém uma base de **fatos** (uma coleção de fatos chamada de banco de dados PROLOG, database, mas observe a diferença do termo com a teoria de bancos de dados) e um conjunto de **regras** para analisar esses dados, inferir, raciocinar, modificar o conhecimento que estamos processando. Este banco de dados poderá ser consultado, utilizando **consultas**. Resumindo, um programa PROLOG está composto por **fatos**, **regras** e **consultas**.

Predicados na Linguagem PROLOG

A sintaxe dos predicados na linguagem PROLOG é:

```
nomepredicado (paramA, paramB, ...).
```

- Os identificadores (de nomes de predicados e constantes simbólicas) deverão começar com uma letra minúscula e poderão ter dígitos e sublinhados;

- Os parâmetros de um predicado deverão ser separados com vírgulas. Também começarão com letra minúscula se são constantes ou com letra maiúscula se são parâmetros;
- As variáveis deverão ser escritas com letra inicial maiúscula;
- A ordem dos argumentos ou parâmetros dependerá do critério do programador (na definição e nas chamadas);
- A quantidade de argumentos chama-se de *arity*.

Fatos na Linguagem PROLOG

A sintaxe dos predicados na linguagem PROLOG é:

```
nomepredicado (paramA, paramB, ...).
```

- Observe a obrigatoriedade do ponto final. Toda cláusula na linguagem PROLOG terminará com ponto;
- Os itens ou parâmetros de um fato poderão ser:
- constantes numéricas, por exemplo: 8 15000 9.8;
- constantes *strings*, por exemplo: “São Paulo” “Itália” “Curitiba” “Ana Lopes”;
- constantes simbólicas, por exemplo: azul brasil ana_lopes sao_paulo belo_horizonte;

- listas de itens, por exemplo: [verde, azul, amarelo] [sincero, honesto, inteligente, preocupado];

- variáveis, por exemplo: X Y Cidade Pais

Consultas na Linguagem PROLOG

A sintaxe de uma consulta que invoca um predicado na linguagem PROLOG será:

```
?- nomepredicado (paramA,paramB, ... ).
```

- Uma consulta começará com a combinação ?-

- Observe a obrigatoriedade do ponto final. Toda cláusula na linguagem PROLOG terminará com ponto;

- Os parâmetros utilizados na consulta poderiam ser constantes ou variáveis, dependendo do que desejamos procurar. Ser for uma variável: usar letra inicial maiúscula;

- Uma consulta poderá verificar vários objetivos, separados por vírgulas(conjunção, operação E) ou pontos e vírgulas (disjunção, operação OU).

Exemplo 1

Mostraremos na cor azul as respostas do PROLOG.

```
% fatos: (isto é um comentário)
cidade(sao_paulo,15000000).
cidade(brasilia,2000000).
cidade(rio_de_janeiro,7000000).
cidade(maripora,400000).
% consultas:
?-cidade(maripora,400000).
Yes
?-cidade(sao_paulo,15000000).
Yes
?-cidade(rio_de_janeiro,7000000).
Yes
?-cidade(maripora,3000).
No
?-cidade(curitiba,120000).
No
?- cidade(maripora,400000).
No
```

Exemplo 2

No exemplo a seguir utilizamos o predicado ***write*** (disponível em algumas versões de PROLOG, como *Strawberry Prolog*), que visualiza um conteúdo na tela. Também, o predicado ***nl***, que provoca uma quebra de linha.

```
% Consulta para listar todas as cidades.
% Veja que utilizamos as variáveis Cid e Pop:
?-cidade(Cid,Pop),write(Cid),write(“ - “),write(Pop),nl.
```

Exemplo 3

Mostraremos na cor azul as respostas do PROLOG.

```
?-cidade(Cid,Pop),write(Cid),write(“ - “),write(Pop),nl.
sao_paulo - 15000000
Yes.
brasilia - 2000000
Yes.
rio_de_janeiro - 7000000
Yes.
maripora - 400000
Yes.
No.
```

Exemplo 4

Observe que as vírgulas significam uma conjunção *E*. Então, se queremos consultar qualquer cidade *Cid*, com população *Pop*, e que esse valor *Pop* seja maior que 2.500.000 de habitantes:

```
% Consulta para listar as cidades com mais de
% 2500000 habitantes:
?- write("Cidades com mais de 2500000 de pessoas"),
   nl,nl, cidade(Cid,Pop),Pop>2500000,write(Cid),
   write(" - "),write(Pop),nl.
```

Exemplo 5

No exemplo a seguir utilizamos o predicado ***read*** (disponível em algumas versões de PROLOG, como *Strawberry Prolog*), que efetua a leitura de um termo, se utilizado o parâmetro t no final ou lê um texto, se utilizado os parâmetros no final.

```
% Consulta para listar a população de uma cidade:
?- read(Cid,"Qual é o nome da cidade",t),
   nl,nl, cidade(Cid,Pop),
   write("A população dessa cidade é "),
   write(Pop),nl.
```

Regras na Linguagem PROLOG

Como foi mencionado, na linguagem PROLOG, um programa poderá estar composto por: fatos, consultas e regras.

Uma regra será utilizada para:

Importante!

Caso a. Estabelecer um procedimento ou sequência de ações a executar.

Caso b. Para criar uma definição.

A estrutura geral de uma regra será a seguinte:

```
nomeRegra(parâmetros) : - objetivos a verificar.
```

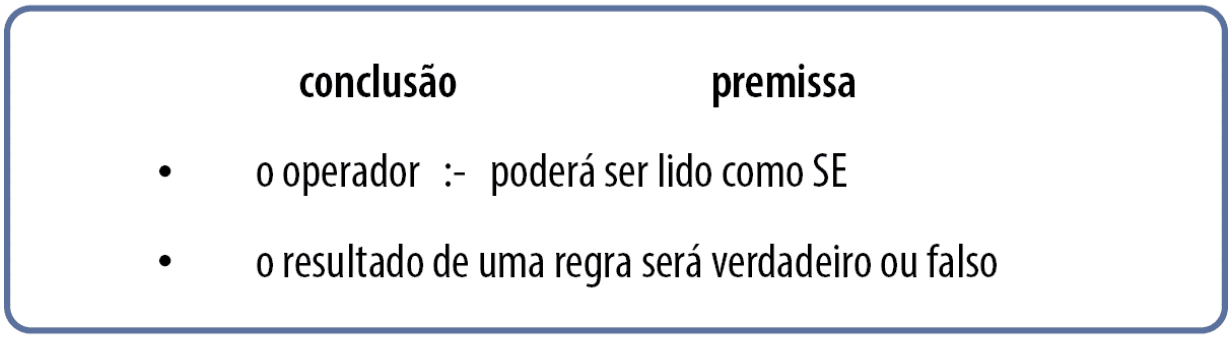

Figura 2

Comentários Sobre a Sintaxe das Regras em PROLOG

```
nomeRegra (paramA, paramB, ... ) :- objetivos.
```

- Uma regra utiliza o operador :- que significa SE;
- As regras são utilizadas preferentemente para estabelecer definições, ou seja, para definir algo;
- O nome de uma regra será escrito com letra inicial minúscula;
- Observe a obrigatoriedade do ponto final, como em toda cláusula na linguagem PROLOG;
- Os parâmetros utilizados na parte esquerda da regra estarão relacionados com as variáveis utilizadas nos objetivos;

- Os objetivos são invocações a predicados (fatos e outras regras), expressões de comparação, etc.;
- Uma regra poderá verificar vários objetivos, separados por vírgulas (operação E) ou pontos e vírgulas (operação OU).

Exemplo 6

```
pessoa(pedro).
pessoa(ana).
pessoa(rosa).
caracteristica(pedro,honesto).
caracteristica(pedro,responsavel).
caracteristica(ana,honesto).
caracteristica(rosa,responsavel).
conhece(pedro,java).
conhece(pedro,vbnet).
conhece(ana,php).
conhece(rosa,vb6).
candidatoVaga(P) :- caracteristica(P,honesto),
  caracteristica(P,responsavel), conhece(P,java),
  conhece(P,vbnet).
?-write("Candidatos à vaga:"), candidatoVaga(Pr), write(Pr), nl.
```

Exercícios de Programação em PROLOG

- Elabora vários fatos com dados de veículos, com os parâmetros mostrados a seguir: veiculo(marca, modelo, ano, preço e cor), por exemplo:veiculo(chevrolet, cruze, 2015, 89000, prata);

- Elabore um programa veiculos1.pro, que liste todos os veículos de uma marca fixa, com valor constante na consulta;

- Elabore um programa veiculos2.pro, que liste todos os veículos de uma marca solicitada pelo usuário, guardando o valor lido em uma variável, que será usada depois na consulta.

Representação de Listas em PROLOG

Uma lista em Prolog pode ser representada como um fato:

predicado ([elem1,elem2,elem3,elem4]).

Mas, uma lista será também:

[Cabeça | Cauda]

Sendo que:

Cabeça é o elemento na cabeça da lista
Cauda é uma lista (é a parte restante da lista, sem a cabeça).

Exemplos de listas na linguagem Prolog:

[honesto, serio, pontual, responsavel]
[honesto | [serio, pontual, responsavel]]

Comentários Sobre a Utilização de Listas

- Permitem o processamento de listas de itens, muitas vezes símbolos (processamento simbólico);
- Uma lista pode ser considerada como uma lista de itens;
- Os itens podem ser símbolos, valores numéricos e *strings*;
- Um objeto poderá ter uma ou várias listas de itens associadas.

% Exemplo Lista1.pro

```
%Mostraremos na cor azul as respostas do PROLOG.
% Dois fatos com listas.
% Veja que o que aparece entre colchetes é uma
% constante, de tipo lista.

cores([vermelho,azul,amarelo,cinza,rosa,marrom]).
cores([verde,laranja,preto,branco]).

% Consulta:

?- cores(X), write(X), nl.

[vermelho,azul,amarelo,cinza,rosa,marrom]
Yes.
[verde,laranja,preto,branco]
Yes.
```

```
% Consulta:
?- cores([Cabeca|Cauda]),write(Cabeca), write(“ “),
    write(Cauda), nl.

vermelho [azul,amarelo,cinza,rosa,marrom]
Yes.
verde [laranja,preto,branco]
Yes.
```

% Exemplo Lista2.pro

```
%Mostraremos na cor azul as respostas do PROLOG.
antibioticos([tetraciclina,kefl ex,sulfa]).
antibioticos([cefamox,cefalexina]).
antipireticos([aspirina,dipirona,paracetamol]).

?- write(“Antibióticos cadastrados:”), nl, antibioticos(Lista),
    write(Lista), nl.

Antibióticos cadastrados:
[tetraciclina,kefl ex,sulfa]
Yes.
[cefamox,cefalexina]
Yes.
No.
```

% Exemplo Lista3.pro

```
% Listas com elementos de tipo string
% Mostraremos na cor azul as respostas do PROLOG.
cidades(“Brasil” , 160,
    [“Campinas” , “São Paulo” , “Rio de Janeiro”,”BH”]).

cidades(“Cuba”, 12,
    [“Habana” , “Santiago de Cuba”, “Varadero” , “Viñales”]).

cidades(“Itália”, 70, [ “Roma” , “Florença”]).

?- cidades(Pais, _ ,ListaCidades), nl, write(Pais), write(“:”),
    nl, write(ListaCidades), nl, nl.

Brasil:
[“Campinas”,”São Paulo”,”Rio de Janeiro”,”BH”]
Yes.

Cuba:
[“Habana”,”Santiago de Cuba”,”Varadero”,”Viñales”]
Yes.

Itália:
[“Roma” , “Florença”]
Yes.
```

No.

% Exemplo Lista4.pro

```
% Mostraremos na cor azul as respostas do PROLOG.
% Fatos com três parâmetros, sendo que dois destes
% parâmetros são listas:

programador("Alex", [prolog,java,vb,jsp,asp],
     [honesto,serio,inteligente, pontual]).
programador("Alberto Lopes", [java,vb,php,asp],
     [serio,pontual,inteligente]).

% Exemplo de consulta:
?- write("Programadores cadastrados:"), nl,
    programador(Nome,Linguagens,Caract),
    write(Nome), write(":"), nl, write(Linguagens), nl,
    write(Caract), nl, nl.
```

Programadores cadastrados:
Alex:
[prolog,java,vb,jsp,asp]
[honesto,serio,inteligente,pontual]

Yes.
Alberto Lopes:

[java,vb,php,asp]
[serio,pontual,inteligente]

Yes.
No.

Identificando se um Elemento faz Parte de uma Lista em PROLOG

Com duas cláusulas: um fato e uma regra recursiva podemos determinar se um elemento faz parte (é um membro) de uma lista qualquer.

a) fato (fim da recursão)

membro(Elemento,[Elemento|_])

b) regra recursiva

membro(Elemento,[_|Cauda]):-membro(Elemento,Cauda)

Analisemos detalhadamente estas duas cláusulas:

O fato afirmativo

membro(Elemento,[Elemento|_])

que seria equivalente a afirmar que

membro(Elemento,[X|_]):-Elemento=X

Este fato pode ser lido como “Elemento será um membro da seguinte lista (não interessa qual seja a cauda da lista) SE Elemento for o item na cabeça da lista”. Algo que é evidente. Por exemplo, se temos uma lista [*w,e,u,q*] não temos a menor dúvida em afirmar que *w* é um elemento desta lista.

Agora no caso da segunda cláusula (regra recursiva):

membro(Elemento,[_|Cauda]):-membro(Elemento,Cauda)

Estamos afirmando nesta regra que “Elemento será membro de uma lista cuja cauda é Cauda se Elemento é um membro da lista Cauda”. Esta definição se baseia numa definição recursiva, que é válida pelo fato de que a cauda de uma lista é, também, uma lista.

% Exemplo Lista5.pr

```
% Mostraremos na cor azul as respostas do PROLOG.
antibioticos([tetraciclina,kefl ex,sulfa]).
antibioticos([cefamox,cefalexina]).

% Cláusulas para saber se um elemento é membro de uma lista:
membro(Elemento,[Elemento|_]).
membro(Elemento,[_|Cauda]):-membro(Elemento,Cauda).

?-write(“Antibióticos cadastrados:”),
   nl,antibioticos(Lista),write(Lista),nl.

?-antibioticos(L),membro(kefl ex,L),nl,nl,
   write(“Kefl ex é um antibiótico”),nl.
```

Antibióticos cadastrados:
[tetraciclina,kefl ex,sulfa]
Yes.
[cefamox,cefalexina]
Yes.

Kefl ex é um antibiótico
Yes.
No.

% Exemplo Lista6.pro

```
%Fatos:
%Os fatos têm esta estrutura:
%caracteristicas(nomedomodelo, listaitensdeserie,listaopcionais).

caracteristicas(corsa,
    [freioABC,travaelet,desembtras],
    [radio,vidroelet,cd,alarme]).

caracteristicas(vectra,
    [freioABC,travaelet,desembtras,cd,alarme,vidroelet],
    [ar,tetosolar]).

caracteristicas(audi,
    [freioABC,travaelet,desembtras,cd,alarme,vidroelet,ar],
    [tetosolar]).
```

```
% Cláusulas para saber se um elemento é membro de uma lista:
membro(Elem,[Elem|_]).
membro(Elem,[_|Resto]):-membro(Elem,Resto).

%Regra que define um veículo “classe A”

classeA(Carro):-
    caracteristicas(Carro,ListaBasica,Opcionais),
    membro(cd,ListaBasica),
    membro(alarme,ListaBasica).

?- read(Carro,”Digite a marca do carro”,t),
    caracteristicas(Carro,ListaBasica,Opcionais),
    write(“Básicos:”),nl,write(ListaBasica),nl,
    write(“Opcionais:”),nl,write(Opcionais),nl.

?- classeA(X), write(X),write(“ é um veículo classe A”),nl
```

% Exemplo Lista7.pro

```
% fatos com três parâmetros, dois deles são listas:

programador(“Alberto Messias”, [java,vb,php,asp],
    [serio,pontual,inteligente]).
```

```
programador("Luis Lopes", [pascal,vb,php,asp],
    [serio,responsavel,inteligente]).

programador("Ana Silva", [vb,vbnet,php,asp,cs],
    [serio,responsavel,inteligente,elegante]).

% Cláusulas para saber se um elemento é membro de uma lista:
membro(Elem,[Elem|_]).
membro(Elem,[_|Resto]):-membro(Elem,Resto).

?- read(Linguagem,"Digite a linguagem de programação desejada",t),
    read(Caract,"Digite a característica fundamental desejada",t),
    write("Programadores com estes dados:"), nl,
    programador(Programador,ListaLinguag,ListaCaract),
    membro(Linguagem,ListaLinguag),
    membro(Caract,ListaCaract),
    write(Programador),nl.
```

A Linguagem de Programação LISP

A linguagem de programação LISP foi criada por John McCarthy no final dos anos 50 – mencionam 1959 – como um formalismo para analisar situações recursivas com um modelo de computação.

De todas as linguagens que continuam sendo utilizadas, somente FORTRAN é mais antiga. Implementação importante: *AutoLisp* e ambiente Visual Lisp para o *software AutoCAD* da *Autodesk*.

Veja as anotações a seguir sobre LISP.

- No ano de 1994 *Common Lisp* converteu-se no primeiro *ANSI standard* em incorporar programação orientada a objetos, com um estilo bem especial;
- A descrição de desenvolvimento e história de LISP poderá ser encontrada no *link* disponibilizado a seguir:

Leitura

History of Lisp

Clique no botão para conferir o conteúdo.

ACESSE

- A importância de LISP dentro da IA surge pelas possibilidades da linguagem para processamento simbólico;
- A linguagem tem seu nome por uma característica fundamental: processamento de listas (*List Processor*);
- **LISP** se enquadra dentro do **paradigma da programação funcional**. Tudo em LISP é uma função. As listas, as operações, os "comandos tradicionais" possuem estrutura de função;
- O *site* da *Association of Lisp Users* se encontra no *link* disponibilizado a seguir;

Leitura

The Association of Lisp Users

Clique no botão para conferir o conteúdo.

ACESSE

- *Lisp Primer*: este é um material gratuito bem detalhado sobre a linguagem LISP, suas funções e possibilidades, que será encontrado no endereço disponibilizado a seguir, dos autores Colin Allen e Maneesh Dhagat;

Leitura

Lisp Primer

Clique no botão para conferir o conteúdo.

ACESSE

- Uma versão gratuita de LISP (CLISP dos autores Bruno Haible e Michael Stoll, uma implementação GNU CLISP e *ANSI Common Lisp*) para testar os programas deste material realize o *download* no *link* a seguir. Edite seus programas com qualquer editor de textos, compile e execute na linha de comandos, por exemplo: c:\clisp-2.49> clisp.exe ola.clisp

Site

Welcome to CLISP

Clique no botão para conferir o conteúdo.

ACESSE

Exemplo

```
; exemplolet.lisp
```

```
(let ((X 2) (Y 4) (Z 5))
    (print (+ X Y Z))
)


(let (a b c)
    (setq a 3)
    (setq b 4)
    (setq c 5)
    (print (+ a b c))
)
```

A primeira função *print* visualizará a soma de 2, 4 e 5 (atribuídos a *X, Y, Z* coma função let). A segunda função *print* visualizará a soma de *a, b* e *c* (com valores 3, 4 e 5 atribuídos com a função *setq*). Então, a saída deste programa será:

11

12

Listas e Funções na Linguagem LISP

A linguagem LISP trata uma função com uma lista, sendo que o primeiro elemento é o nome da função e os restantes elementos são os parâmetros.

(name-of-function fi rst-argument second-argument ...)

Por exemplo, na seguinte linha avaliamos a função + com os parâmetros 2, 10e 40 (calculamos a soma de 2, 10 e 40).

```
>(+ 2 10 40)
52
```

Atribuir uma lista a uma variável poderá ser feito com as funções *setq* e *let*, por exemplo:

```
> (setq lista '(honesto serio responsavel))
(HONESTO SERIO RESPONSAVEL)
> lista
(HONESTO SERIO RESPONSAVEL)
```

Observe a utilização da apóstrofe ' no exemplo anterior, para especificar que se trata de uma lista de símbolos e não de variáveis.

Ou seja, LISP considera este + como uma função com três parâmetros.

Veja este exemplo de programa completo, que usa uma função soma para calcular a soma de dois valores enviados como parâmetros.

Exemplo

```
soma.lisp
; utiliza uma função para calcular a soma de dois valores inteiros:
```

```
(defun soma (a b)     ;observe a sintaxe para definir os parâmetros
    (+ a b)
)

(write-line "Olá, mundo.")
(print "A soma de dois valores vale:")     ;cabeçalho
(print (soma 4 3) )     ;visualiza a soma
```

O mesmo exemplo, agora com valores reais:

Exemplo

```
; somareais.lisp
; utiliza uma função para calcular a soma de dois valores reais:

(defun soma (a b)
    (+ a b)
)

(write-line "Olá, mundo.")

(print "A soma de dois valores vale:") ;cabeçalho
(print (soma 4.1 3.4) )     ;visualiza a soma
```

Exemplo

Um exemplo com a função fatorial. A função if será explicada depois.

```
;;; ola.lisp
;;; uma função para calcular e retornar o fatorial de um valor inteiro:


(defun fatorial (n)
   (if (= n 1)
    1
    (* n (fatorial (- n 1)) )
   )
)


(write-line "Hello, world.") ;print the famous greeting
(write-line "Olá, mundo.") ;mostra a famosa frase de boas-vindas
(write-line "Hola, mundo.") ;muestra la famosa frase de bienvenida

(print "O fatorial de 3 vale:")     ;cabeçalho
(print (fatorial 3)) ;visualiza o fatorial de 3
```

Entrada e Saída Básica na Linguagem LISP

Já vimos em exemplos anteriores situações em que visualizamos informação na tela:

```
(write-line “Hola, mundo.”)
;a função anterior visualizará o texto, sem aspas e um retorno de linha
(print “O fatorial de 3 vale:”)
; a função anterior visualizará uma mensagem entre aspas
(print (factorial 3))
; a função anterior visualizará o valor do fatorial de 3, sem aspas
```

Nos exemplos anteriores, *write-line* e *print* são funções que permitem visualizar informação na tela. As chamadas são efetuadas com estrutura de listas LISP. Existe outra função para leitura de dados, a função *read*, que retorna o valor lido (que poderá ser um átomo, uma *string* ou uma lista).

```
(write-line “Digite seu nome:”)
(setq Nome (read))
;o nome da variável é Nome (pode ser com letra minúscula)
```

Algumas Funções Primitivas da Linguagem LISP

Algumas das funções primitivas (funções que fazem parte da linguagem, pré-definidas) são:

+, -, *, /, exp, expt, log, sqrt, sin, cos, tan, max, min.

Exemplo

```
(let ((a 3) (b 4) (c 5))
    (write-line “Maior de quatro valores:”)
    (print (max a b 9 c))
)
; visualizará 9
```

Exemplo

```
;; lembrando que Pi (180º) vale 3.141592 e Pi/2 (90º) vale 1.570796
(write-line “Seno, cosseno e tangente de 90º:”)
(print (sin (/ 3.141592 2)))
(print (cos 1.570796))
(print (tan 1.570796))
```

Exemplo

```
(write-line “Digite um valor numérico (para casas decimais use ponto)”)
(setq valor(read))
(print “A raíz quadradada do valor digitado é “)
(print (sqrt valor))
```

Funções em LISP para Processamento de Listas

LISP deve seu nome ao processamento de listas. O processamento simbólico é o motivo fundamental pelo qual LISP é, junto com a linguagem PROLOG, considerada como uma linguagem adequada para a IA.

Existem os chamados seletores, para acessar os elementos de uma lista.

```
>(fi rst ‘(a s d f))
a

>(fi rst ‘((a s) d f))
(a s)

>(rest ‘(a s d f))
(s d f)

>(rest ‘((a s) d f))
(d f)
```

```
>(rest '(a))
NIL

>(last '(a s d f))
f

>(length '(11 22 23))
3
```

Outras funções, chamadas construtores são: *cons*, *append* e *list*.

A função *list* permite construir uma lista. A função *cons* permite adicionar elementos na frente da lista.

```
> (cons 'verde '(azul cinza branco))
(VERDE AZUL CINZA BRANCO)
```

Mas devemos observar que a função cons não modifica a lista se ela estiver armazenada numa variável, como mostrado no seguinte exemplo.

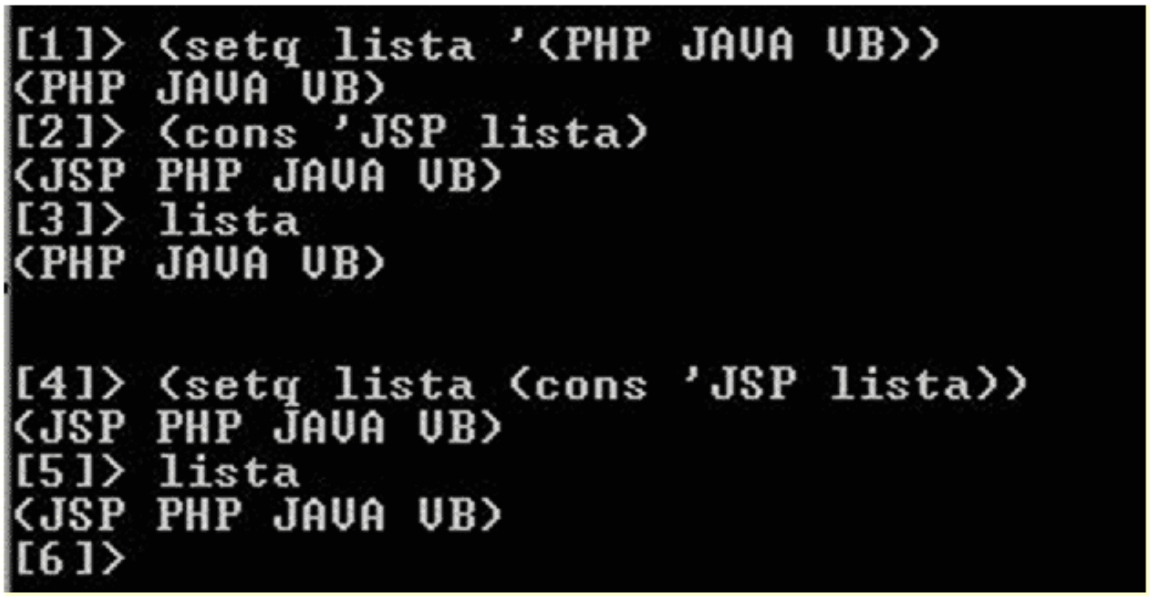

Figura 3

Observe que na linha [3] o conteúdo da lista sendo mostrado não considera a linguagem JSP antes acrescentada. Seria necessário utilizar uma atribuição com *setq/cons*, como mostrado na linha [4].

Tomadas de Decisão na Linguagem LISP

As funções para tomadas de decisões (*if* e *cond*) possuem, também, estrutura de listas. A função *if* possui a estrutura geral seguinte:

(if <expressãoTeste> <parteThen> <parteElse>)

No exemplo do fatorial mostrado em um tópico anterior utilizamos esta estrutura *if*:

```
( if (= n 1)
    1
   (* n (fatorial (- n 1))
)
```

Observe:

- a expressão de teste para a tomada de decisão (= n 1) ;
- o resultado retornado pela função em caso afirmativo, o valor 1;
- em caso de ser falsa a condição, o valor retornado será a expressão numérica da chamada recursiva (* n (fatorial (- n 1)).

A estrutura *cond* (função *cond*) para tomadas de decisões, permite verificar diferentes condições, o que seria equivalente a vários *if* aninhados. Em forma simplificada, a função *cond* seria:

```
(cond (<testA> <resultA>)
    (<testB> <resultB>)
    ...
    (<testN> <resultN>)
)
```

No seguinte exemplo, a função recebe um parâmetro *x* e retornará *x*-1 se x for menor que 10, retornará *x*+2 se x for maior que 32 e em qualquer outro caso retornará *x*+3.

```
(defun funcao (x)
  (cond ((< x 10) (- x 1))
    ((> x 32) (+ x 2))
    (t (+ x 3))
  )
)
```

Por exemplo, uma chamada

```
(print (funcao 40))
```

visualizará o valor 42 na tela.

Procurando um Elemento Dentro de uma Lista

Existe uma função predefinida para saber se um elemento pertence ou se encontra-se dentro de uma lista: a função *member*.

member elementoProcurado lista

Por exemplo, uma chamada como a seguinte:

>(member 'AZUL '(AMARELO AZUL VERDE CINZA))

retornará a lista a partir da posição onde foi encontrado o item 'AZUL que está sendo procurado na lista:

(AZUL VERDE CINZA)

e uma busca sem sucesso retornará NIL, como no exemplo:

>(member 'BRANCO '(AMARELO AZUL VERDE CINZA))

A função member possui outros parâmetros para caracterizar a busca, mas preferimos comentar somente uma situação simplificada.

Finalmente, mostramos um programa LISP completo que utiliza alguns dos conceitos estudados. A variável *progA* armazena as características pessoais e as linguagens que conhece um determinado programador. Observe a utilização de duas listas dentro de uma lista externa.

Exemplo

```
;remedios1.lisp
(setq Antibioticos '(Tetraciclina Sulfa Cefalexina))
(setq antipireticos '(aspirina dipirona paracetamol))

(write-line "")
(write "A lista de antibióticos é: ")
(print (list Antibioticos))

(write-line "")
(write "A lista de remédios para abaixar a febre (antipiréticos) é:")
(print antipireticos)
```

```
(write-line “”)
(write-line “Digite um remédio para pesquisar:”)
(setq Remed (read))
(write-line “”)
(print(
  if (member Remed antipireticos)
    (list Remed “ é um antipirético”)
    (list Remed “ não é um antipirético”)
  )
)
```

Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta Unidade:

Sites

Lisp Primer

Lisp Primer é um material gratuito bem detalhado sobre a linguagem LISP, suas funções e possibilidades, que será encontrado no *link* a seguir, dos autores Colin Allen e Maneesh Dhagat.

Clique no botão para conferir o conteúdo.

ACESSE

Welcome to CLISP

Uma versão gratuita de LISP (*CLISP* dos autores Bruno Haible e Michael Stoll, implementação GNU CLISP e *ANSI Common Lisp*) para testar os exemplos deste material no *link* a seguir. Edite seus programas com qualquer editor de textos, compile e execute na linha de comandos.

Clique no botão para conferir o conteúdo.

ACESSE

Strawberry Prolog

Uma implementação gratuita de PROLOG para testar os programas deste material, e também uma versão paga podem ser encontradas no *link* a seguir. Desenvolvida por Dimiter Dimitrov Dobrev e sua equipe, na Universidade de Sofia, Bulgária.

Clique no botão para conferir o conteúdo.

ACESSE

Leitura

Introdução à Programação Introdução à Programação Prolog

Um material complementar sobre a linguagem de programação PROLOG pode ser encontrado no *link* a seguir:

Clique no botão para conferir o conteúdo.

ACESSE

Referências

ALLEN, C.; MANEESH, D. ***Lisp Primer***. Disponível em: <https://goo.gl/m1vv24>.

BRATKO, I. ***Prolog***: *Programming For Artificial Intelligence*. 3. ed. Harlow: Addison-Wesley, 2001.

FIESER, J.; DOWDEN, B. ***Artificial Intelligence***. The University of Tennessee. Disponível em: <https://goo.gl/wtsY6H>. Acesso em: 30/10/2017.

GANASCIA, J. **Inteligência Artificial**. São Paulo: Ática, 1997.

LUGER, G. F. **Inteligência Artificial**. 6ª. Ed. São Paulo: Pearson Education do Brasil, 2013.

MARGOLIES, D. ***The Ansi Common Lisp Reference Book***. 2. ed. [s.l.]: Apress, 2005.

RICH, E.; KNIGHT, K. **Inteligência Artificial**. 2a ed. São Paulo: Makron Books, 1994.

ROSA, J. L. G. **Fundamentos da Inteligência Artificial**. Rio de Janeiro: Editora LTC, 2011.

RUSSELL, S. J.; NORVIG, P. **Inteligência Artificial:** Referência Completa para Cursos de Computação. 2. ed. Rio de Janeiro: Elsevier, 2004.

WINSTON, P. H. ***Artificial Intelligence***. 3. ed. Massachusetts: Addison-Wesley Publishing Cia., 1993.