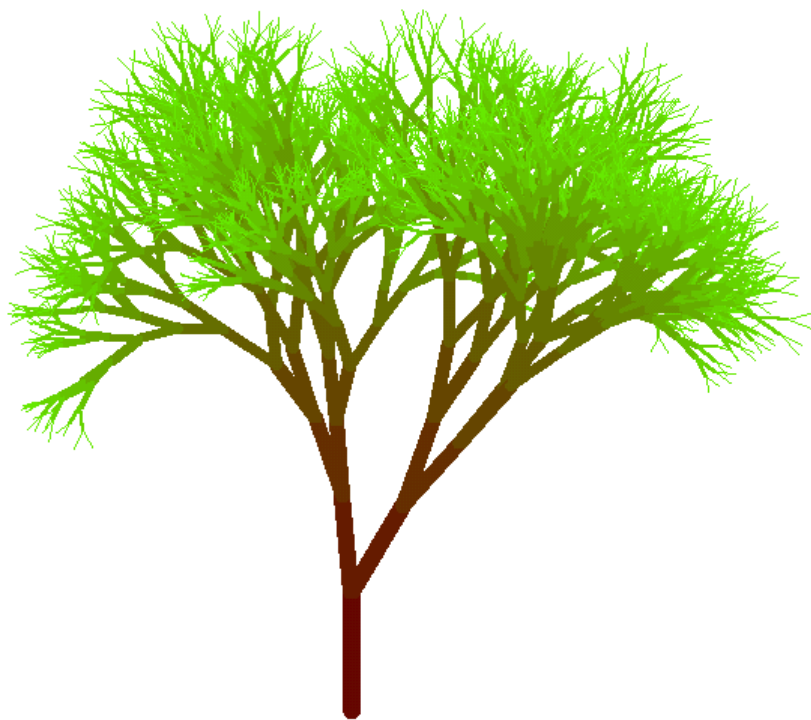


Lógica e Linguagem de Programação

Uma Introdução ao Pensamento
Procedimental



Prof. Ricardo Bezerra de Menezes Guedes

Sumário

SUMÁRIO.....	2
1. INTRODUÇÃO.....	4
O QUE É INFORMÁTICA?.....	4
O QUE É INFORMAÇÃO?.....	4
O QUE É COMUNICAÇÃO?.....	5
O QUE É UMA LINGUAGEM?.....	5
.....	5
LINGUAGEM DE PROGRAMAÇÃO DE COMPUTADOR.....	6
2. TIPOS DE PROGRAMAS DE COMPUTADOR.....	7
AMBIENTE INTEGRADO DE DESENVOLVIMENTO - IDE.....	8
EXEMPLOS.....	8
3. A LINGUAGEM LOGO.....	10
SENTENÇAS NA LINGUAGEM LOGO.....	12
<i>Exercícios:</i>	14
AULA PRÁTICA I - CONVERSANDO COM A TARTARUGA NA LINGUAGEM LOGO.....	15
A PALAVRA REPITA.....	21
4. PROGRAMANDO COMPUTADORES.....	22
ALTERANDO/CORRIGINDO PROCEDIMENTOS.....	25
EXERCÍCIOS.....	27
5. CONCEITO DE SUBPROCEDIMENTO.....	30
AULA PRÁTICA 3 - CONCEITOS DE SUBPROCEDIMENTO.....	33
6. CONCEITO DE VARIÁVEL.....	34
EXEMPLOS DE PROCEDIMENTOS GRÁFICOS COM VARIÁVEIS.....	35
EXEMPLOS DE PROCEDIMENTOS NUMÉRICOS COM VARIÁVEIS.....	37
7. CONCEITOS DE ESTADO E DE OPERADORES DE MUDANÇA DE ESTADO.....	41
<i>Exemplos</i>	44
8. EXPRESSÕES ARITMÉTICAS.....	47
OPERADORES ARITMÉTICOS DA LINGUAGEM LOGO.....	47
FUNÇÕES MATEMÁTICAS PREDEFINIDAS DA LINGUAGEM LOGO.....	47
EXERCÍCIOS.....	48
9. EXPRESSÕES LÓGICAS.....	49
OPERADORES RELACIONAIS DA LINGUAGEM LOGO.....	49
OPERADORES LÓGICOS DA LINGUAGEM LOGO.	49
10. ESTRUTURAS DE CONTROLE.....	50
ESTRUTURAS DE CONTROLE DE DECISÃO.....	50
ESTRUTURA DE CONTROLE DE DECISÃO SIMPLES.....	51

<i>ESTRUTURA DE CONTROLE DE DECISÃO COMPOSTA</i>	51
<i>ESTRUTURAS DE CONTROLE DE REPETIÇÃO</i>	52
<i>O Comando para</i>	53
<i>O Comando enquanto</i>	54
11. RECURSIVIDADE	56
<i>SEQÜÊNCIAS DEFINIDAS RECURSIVAMENTE</i>	56
<i>OPERAÇÕES DEFINIDAS RECURSIVAMENTE</i>	57
<i>FIGURAS DEFINIDAS RECURSIVAMENTE</i>	58
<i>OUTROS EXEMPLOS DE FIGURAS GERADAS COMO A CURVA DE KOCH</i>	60
<i>EXEMPLO 5 – ÁRVORES</i>	65
12. ESTRUTURAS DE DADOS	67
<i>LISTAS E PALAVRAS</i>	67
<i>O CARACTER ESPECIAL "\"</i>	68
<i>criação de listas</i>	68
<i>Usando a colchetes como delimitadores</i>	68
<i>Usando a palavra lista</i>	69
<i>Usando a palavra sentença</i>	69
<i>operações sobre listas e palavras</i>	70
<i>operadores de inserção</i>	70
<i>operadores de consulta</i>	71
<i>operadores de exclusão</i>	72
<i>EXEMPLOS DE PROCEDIMENTOS</i>	72
<i>OUTROS OPERADORES</i>	73
<i>EXERCÍCIOS</i>	74

1. INTRODUÇÃO

Quando estudamos a ciência da computação é importante que entendamos em primeiro lugar, alguns conceitos básicos tais como informática, informação, comunicação e linguagem.

O QUE É INFORMÁTICA?

Do dicionário Aurélio obtemos a seguinte definição para informática:

“Ciência do tratamento racional e automático da informação, considerada esta como suporte dos conhecimentos e comunicações”.

Da definição acima destacamos dois conceitos importantes para o estudo da lógica e linguagem de programação: Informação e comunicação.

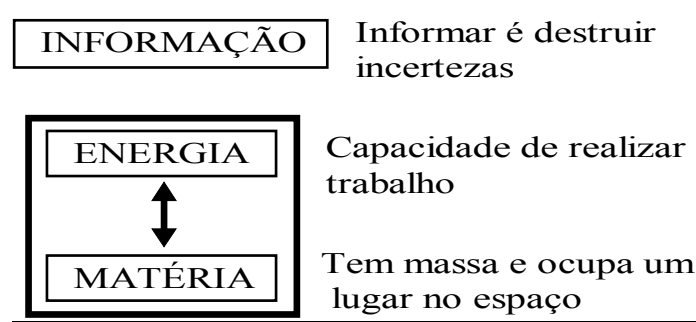
O QUE É INFORMAÇÃO?

Vamos explicar o conceito de informação por analogia a dois conceitos muito conhecidos pela ciência: Matéria e Energia.

O que é matéria? Matéria é tudo aquilo que tem massa e ocupa um lugar no espaço. Decoramos essa definição para fazer as provas na escola. A quantidade de matéria de um corpo pode ser medida através da sua massa. Medimos o espaço ocupado pela massa no espaço tridimensional. Dominamos, assim, o conceito de matéria, pois definimos e medimos matéria.

O que energia? Energia é definida nos livros de Física como a capacidade de realizar trabalho. Sabemos que existem vários tipos de energia, ou seja, a energia se apresenta em diversas formas diferentes. Sabemos, também, que podemos transformar um tipo de energia em outro. Medimos a quantidade de energia. Dominamos, assim, o conceito de energia, pois definimos e medimos energia.

Algumas pesquisas científicas recentes colocam a informação como um dos três elementos básicos da natureza, juntamente com matéria e energia.



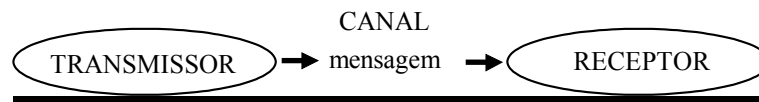
O conceito de matéria já é bastante conhecido e manipulamos algebricamente grandezas tais como peso e massa. Da mesma maneira, grandezas tais como trabalho e energia são estudadas em cursos de nível médio. Estamos vivendo a era da informação

e devemos saber que é possível medir a quantidade de informação contida em uma mensagem assim como podemos medir a quantidade de massa de um corpo. A disciplina que trata desse tema é a teoria da informação. A teoria da informação fornece as bases teóricas para as áreas de informática e telecomunicações.

O QUE É COMUNICAÇÃO?

Comunicação é o processo de transporte de *informação* de um ponto a outro. Toda comunicação pressupõe a existência de pelo menos quatro elementos mostrados a seguir:

1. TRANSMISSOR: emite a mensagem.
2. RECEPTOR: recebe a mensagem.
3. CANAL: meio através do qual a mensagem é transmitida.
4. MENSAGEM: *informação* codificada numa linguagem. Este código deve ser conhecido tanto pelo transmissor como pelo receptor.



O QUE É UMA LINGUAGEM?

Sistema de sinais convencionados que serve de meio de comunicação. Exemplos: linguagem visual, linguagem falada, linguagem escrita, etc.

As linguagens são compostas por:

ALFABETO: conjunto de símbolos (normalmente letras).

PALAVRAS: seqüência de símbolos.

VOCABULÁRIO: conjunto de palavras da linguagem.

SENTENÇAS: seqüência de palavras da linguagem.

Observações:

1. Nem toda seqüência de símbolos (palavra) pertence ao vocabulário da linguagem.
1. Nem toda seqüência de palavras (sentença) pertence ao conjunto de sentenças da linguagem. Somente aquelas ordenadas de acordo com as *regras sintáticas* estabelecidas.

LINGUAGEM DE PROGRAMAÇÃO DE COMPUTADOR

LINGUAGEM DE PROGRAMAÇÃO DE COMPUTADOR é uma *linguagem* que utilizamos para nos *comunicarmos* com o computador.

As linguagens de programação, como todas as linguagens, têm um alfabeto, palavras, um vocabulário e regras sintáticas para formação de sentenças. Existem várias linguagens de programação de computadores. A primeira linguagem de programação que estudaremos é a linguagem *Logo* ou “linguagem da tartaruga”. Esta linguagem de programação foi desenvolvida para tornar mais fácil o processo de aprender a programar computadores.

A tabela abaixo mostra, em ordem cronológica, algumas das principais linguagens de programação e sua aplicação principal.

Nome	Desenvolvida	Ano	Aplicação
FORTRAN	John Backus e equipe na IBM	1954	Científica Engenharia
LISP	John McCarthy no MIT	1956	Científica Inteligência Artificial
COBOL	Grace Murray Hopper e comitê.	1959	Comercial.
ALGOL	Comitê internacional	1960	Científica
APL	Kenneth Iverson da IBM	1961	Científica Indústria Aeronáutica
PL/1	Equipe da IBM	1964	Científica, Comercial e Educativa
BASIC	John Kemeny e Thomas Kurtz do Dartmouth College	1965	Científica Educativa
Logo	Seymour Papert e equipe do MIT	1968	Educativa
Pascal	Niklaus Wirth do Instituto Federal de Tecnologia da Suíça.	1971	Científica Educativa.
C	Dennis Ritchie do Bell Laboratories	1973	Científica Desenvolvimento
Ada	Jean Ichbiah e equipe em Honeywell	1979	Militar
Python	Guido van Rossum	1991	aumentar a produtividade do programador
Java	James Gosling da Sun	1995	Aplicativos baseados na Internet

Tabela 1: Principais linguagens de programação de alto nível

2. Tipos de Programas de Computador

Os programas de computador que mais utilizamos são os *Sistemas Operacionais* e os *Programas Aplicativos*.

SISTEMA OPERACIONAL é um programa que gerencia os recursos do computador, tais como, teclado, impressora, discos, memória, etc. Como exemplos de sistemas operacionais podemos citar o WINDOWS 98, o LINUX e o UNIX.
--

PROGRAMA APLICATIVO é aquele que utilizamos para realizar alguma tarefa específica tal como digitar um texto (usando, por exemplo, o Word da Microsoft) ou elaborar uma planilha de cálculo (usando o EXCEL).

Por analogia podemos dizer que o sistema operacional faz o papel do apresentador de um programa de auditório cujas atrações são os programas aplicativos.

Os PROGRAMAS DE COMPUTADOR devem ser escritos numa linguagem que o computador “entenda”. Na verdade, a única linguagem que o computador entende é sua linguagem particular, conhecida como linguagem de máquina.

LINGUAGEM DE MÁQUINA.

Código binário de 0's e 1's que representam os níveis alto e baixo de tensão do circuito eletrônico digital. É a única linguagem que o computador “entende”.
--

Existe uma linguagem de máquina para cada tipo de microprocessador usado pelo computador. Para tornar mais fácil o processo de programação foi criada uma linguagem mnemônica chamada *assembly*.

LINGUAGEM ASSEMBLY

Linguagem em que as <i>instruções</i> são dadas usando-se letras para as <i>operações</i> e números em hexadecimal para os <i>dados</i> ou <i>endereços</i> .

Para que o computador possa “entender” a linguagem assembly é necessário que se utilize um *programa montador* ou *assembler*, que traduzirá o procedimento escrito na linguagem *assembly* para a *linguagem de máquina*.

Para tornar ainda mais fácil a programação de computadores foram criadas várias linguagens de programação que se aproximam das línguas naturais utilizadas pelo homem. Algumas dessas linguagens foram listadas na tabela da primeira lição e são conhecidas como linguagens de alto nível.

LINGUAGENS DE ALTO NÍVEL.

Linguagem de programação de computador que tenta se aproximar da língua natural humana.

É necessário que se utilize um *programa tradutor* para transformar o procedimento escrito em linguagem de alto nível para linguagem de máquina. Existem dois tipos de programas tradutores:

1. **COMPILADORES:** são programas que traduzem automaticamente a *totalidade* de um procedimento escrito numa linguagem de alto nível, chamado PROGRAMA FONTE, em um procedimento completo escrito em linguagem de máquina, chamado PROGRAMA OBJETO.
Exemplos de linguagens compiladas: PASCAL, C++, CLIPPER e DELPHI (Object Pascal).
2. **INTERPRETADORES** são programas que traduzem para linguagem de máquina, *instrução por instrução* de um procedimento escrito numa linguagem de alto nível. Cada instrução é executada imediatamente após a tradução.
Exemplos de linguagens interpretadas: LISP, BASIC e LOGO.

AMBIENTE INTEGRADO DE DESENVOLVIMENTO - IDE

Segundo a Wikipédia, **IDE**, do inglês *Integrated Development Environment* ou **Ambiente Integrado de Desenvolvimento**, é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de software com o objetivo de agilizar este processo.

As características e ferramentas mais comuns encontradas nos IDEs são:

- **Editor** - edita o código-fonte do programa escrito na(s) linguagem(ns) suportada(s) pela IDE;
- **Tradutor** (*compilador ou interpretador*) - traduz o código-fonte do programa, editado em uma linguagem específica e o transforma em linguagem de máquina;
- **Depurador** (*debugger*) - auxilia no processo de encontrar e corrigir erros (bugs) no código-fonte do programa, na tentativa de aprimorar a qualidade de software;

Exemplos

- **Delphi** - Trabalha originalmente com a linguagem Object Pascal/Pascal, agregando na suite Delphi Studio 2005, a linguagem C# e a extensão da *Object Pascal* para .NET;

- **Eclipse** - Gera código Java (através de plugins, o Eclipse suporta muitas outras linguagens como Python e C/C++);
- **Netbeans** - Gera código Java.
- **BlueJ** - Gera código Java.
- **Visual Studio .NET** - Gera código para Framework .NET, suportando linguagens Visual Basic .NET, C#, C++, e J#.
- **DEV-C++** - Geram código para C e C++

Chamaremos, nessa apostila, IDE de ambiente de programação.

3. A Linguagem Logo

A linguagem de programação Logo foi desenvolvida, por Seymour Papert e equipe no M.I.T., no final dos anos 1960, para facilitar o processo de ensino da programação de programadores.

O ambiente de programação que utilizaremos é o XLogo. Ele foi escrito em Java, o que torna possível executá-lo em muito sistema operacionais diferentes, dentre eles, o Linux e o Windows. Ele possui interpretador para oito idiomas: alemão, árabe, francês, inglês, espanhol, português, galês e esperanto. O ambiente XLogo é distribuído sob licença GPL, ou seja, é um programa ("software") livre e gratuito.

IMPORTANTE: Para que um aplicativo escrito em Java possa ser executado, deve ser instalado no computador um ambiente de execução Java conhecido como JRE (Java Runtime Environment). Ele é gratuito pode ser copiado e instalado do endereço: http://www.java.com/pt_BR/ e clique no botão "Download gratuito do Java".

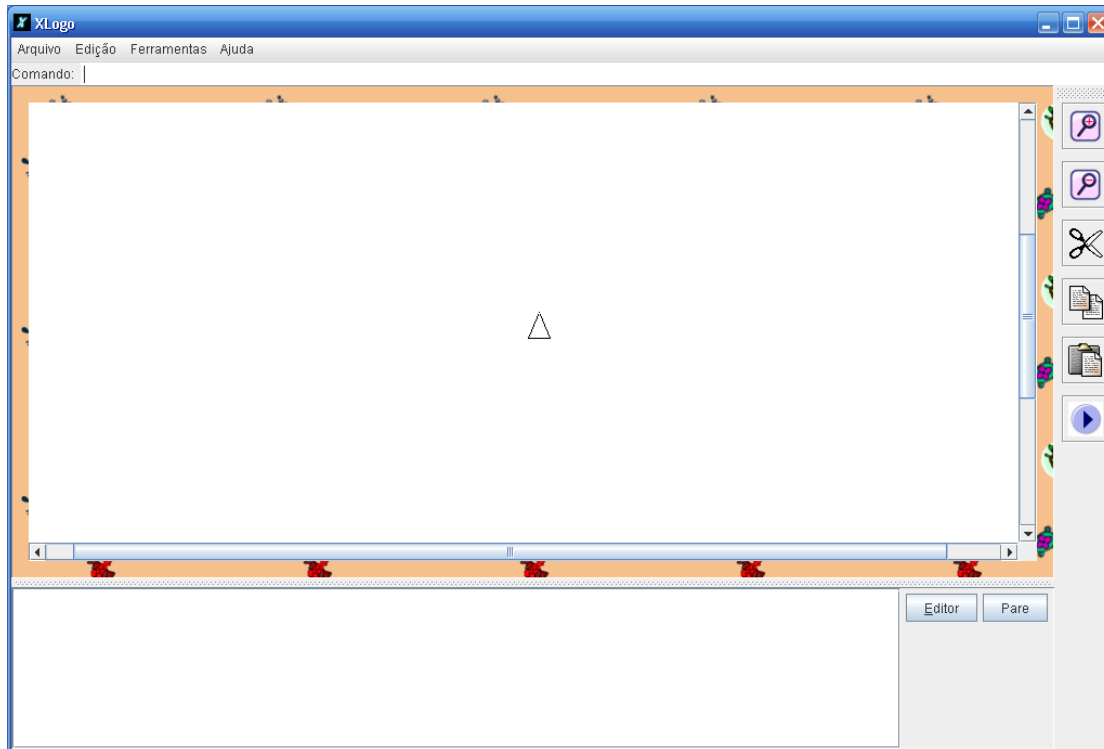
Baixe o ambiente XLogo indo ao endereço <http://xlogo.tuxfamily.org/pt/telechargements.html> e baixando o arquivo xlogo.jar. Arquivos de extensão jar são conhecidos como executáveis Java. Para executá-lo no sistema operacional Windows basta um duplo clique sobre ele.

Quando executarmos pela primeira vez o arquivo xlogo.jar (executável JAVA) aparecerá a janela de seleção de idioma abaixo:



Para selecionar a língua portuguesa clique na bandeira do Brasil e depois clique no

botão OK. Esta seleção só será feita na primeira execução. Será então apresentada a janela principal do ambiente XLogo, mostrada abaixo.



No topo encontra-se a barra de menus com os itens: Arquivo Edição Ferramentas e Ajuda. Logo abaixo aparece a **linha de comando**, onde são escritos os comandos (instruções). Depois de escrito o comando basta dar "enter" para que o comando seja executado.

No meio, temos a **área de desenho**. Na área de desenho vive um bichinho virtual chamado tartaruga. Por isso, a linguagem Logo é, também, conhecida como a linguagem da tartaruga. A tartaruga responde a alguns comandos digitados na linha de comandos. Sua forma inicial é a de um triângulo, mas ela pode trocar de roupa usando o menu Ferramentas – Preferências e na aba Roupa da Tartaruga selecionamos uma roupa e clicamos em OK.

Na parte de baixo encontra-se o histórico de comandos. Ele mostra os comandos já executados bem como as respectivas respostas. Ela mostra também eventuais mensagens de erro que aparecem em vermelho. Para chamar um comando já executado basta clicar nele no histórico de comandos, ou usar as setas de teclado (para baixo e para cima) na linha de comando.

No canto direito inferior há dois botões: Editor e Pare. PARE interrompe a execução do programa e EDITOR abre a janela do editor de procedimentos.

No lado direito da janela temos uma barra de ferramentas com botões aumentar ou diminuir o tamanho do desenho e recortar, copiar e colar.

SENTENÇAS NA LINGUAGEM LOGO

A linguagem de programação Logo possui, como todas as linguagens, um vocabulário. As palavras que compõe o vocabulário de uma linguagem de programação são chamadas de *palavras primitivas* ou de *comandos primitivos*.

Exemplos de palavras primitivas da linguagem Logo: **para frente**, **para trás**, **para direita**, **para esquerda**.

As regras sintáticas para formação de frases na linguagem Logo são muito simples. Todas as frases da linguagem Logo apresentam a seguinte estrutura sintática:

<palavra> *<parâmetro(s) de entrada>*

Ou seja, uma frase na linguagem Logo sempre se inicia por uma palavra do vocabulário da tartaruga seguida de uma lista de *parâmetros de entrada*.

Esta lista pode ser vazia. Os parâmetros de entrada desempenham o mesmo papel dos objetos direto e indireto, da língua portuguesa. Por exemplo, a sentença “**para frente** 100” é composta da palavra **para frente**, que pertence ao vocabulário da tartaruga, seguida do número 100, que é o *parâmetro de entrada* necessário para completar o sentido da frase. Se digitarmos apenas “**para frente**”, apesar da palavra ser reconhecida pela tartaruga como pertencente ao seu vocabulário ela responde que “*Faltam dados para para frente*”. Assim como os verbos transitivos necessitam de complementos (objetos diretos ou indiretos) para fazerem sentido, alguns comandos da linguagem Logo precisam de parâmetros de entrada para formar uma frase completa.

Como sabemos, para nos comunicarmos em uma linguagem qualquer não é necessário que conheçamos todas as palavras do seu vocabulário. Assim, de início, para nos comunicarmos com a tartaruga é necessário que saibamos algumas poucas palavras. As primeiras palavras da linguagem da tartaruga que aprenderemos são **para frente**, **para trás**, **para direita**, **para esquerda** e **limpe desenho**, cujas definições mostramos a seguir.

para frente
Sintaxe : para frente <i>número</i> pf <i>número</i>
Descrição : movimenta a tartaruga para frente o <i>número</i> de passos, ou seja, desloca a tartaruga no sentido em que ela estiver apontando.
para trás
Sintaxe: para trás <i>número</i> pt <i>número</i>
Descrição: movimenta a tartaruga para trás o <i>número</i> de passos, ou seja, desloca a tartaruga no sentido oposto ao que ela estiver apontando.

<p>paradireita</p> <p>Sintaxe: paradireita <i>número</i> pd <i>número</i></p> <p>Descrição: gira a tartaruga para a direita o <i>número</i> especificado em graus.</p>
<p>paraesquerda</p> <p>Sintaxe : paraesquerda <i>número</i> pe <i>número</i></p> <p>Descrição: gira a tartaruga para a esquerda o <i>número</i> especificado em graus.</p>
<p>limpedesenho</p> <p>Sintaxe: limpedesenho ld</p> <p>Descrição: Limpa todos os desenhos na tela e restaura a ld (coloca-a no centro da tela olhando para cima).</p>

Por exemplo, se digitarmos na linha de comandos: **para frente 100**, seguido de <ENTER>, a tartaruga respondera se deslocando 100 passos para frente deixando um traço. Ver figura ao lado

Observação: Um passo equivale a um pixel (ponto na tela do monitor).



Se digitarmos, em seguida, **paradireita 90**, a tartaruga girará 90 graus no sentido do ponteiro do relógio. Ver figura ao lado.



Se digitarmos agora **paratrás 50** veremos a tartaruga se deslocar 50 passos para trás deixando um traço no caminho percorrido. Ver figura ao lado.



Se digitarmos **limpedesenho** o desenho será apagado e a tartaruga será reposicionada no centro da área de desenho, olhando para cima.

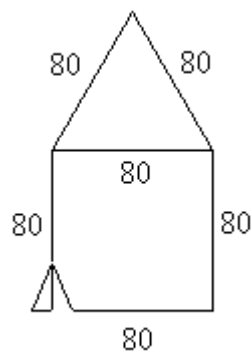
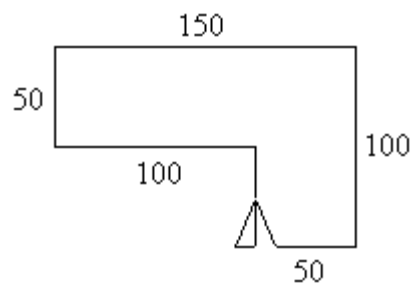
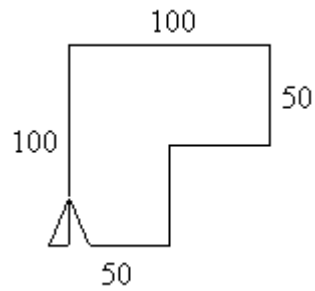
Podemos digitar de uma só vez vários comandos: **pf 50 pd 90 pf 100 pd 90 pf 50 pd 90 pf 100** desenha o retângulo mostrado na figura ao lado.

Observação: **pf** é a abreviatura de **para frente** e **pd** é a de **paradireita**.



Exercícios:

1. Escreva uma sequência de comandos que faça a tartaruga desenhar um quadrado de lado 100?
2. Escreva uma sequência de comandos que faça a tartaruga desenhar as figuras abaixo. Os números indicam o tamanho de cada reta e não precisam ser desenhados.



AULA PRÁTICA I - Conversando com a Tartaruga na linguagem Logo

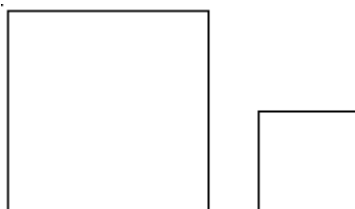
Objetivo principal: Aprender novas palavras do vocabulário da linguagem *Logo* utilizando-as para fazer desenhos variados.

Temos abaixo mais algumas palavras do vocabulário da tartaruga.

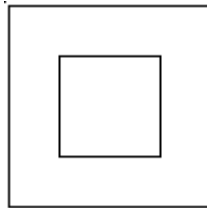
usenada Sintaxe : usenada un Descrição: retira o lápis ou a borracha da tartaruga sem alterar sua posição e direção, possibilitando à tartaruga se movimentar sem desenhar linhas.
uselápis Sintaxe : uselápis ul Descrição : coloca um lápis sob a tartaruga sem alterar sua posição e direção, possibilitando desenhar linhas por onde a tartaruga passar.
useborracha Sintaxe : useborracha ub Descrição : coloca uma borracha na tartaruga. A partir deste comando a tartaruga anda sem riscar e, se passar sobre linhas já traçadas, estas linhas serão apagadas.
escondetat Sintaxe : escondetat dt Descrição: faz a tartaruga desaparecer da tela, tornando-a invisível.
mostretat Sintaxe : mostretat at Descrição: faz com que a tartaruga apareça na tela, na sua última posição.

Usando estas novas palavras tente fazer os desenhos seguintes:

1. Escreva uma seqüência de comandos que faça a tartaruga desenhar cada figura abaixo:
 - a) quadrado maior de lado 100 e menor de lado 50 e a distância entre os quadrados 30



b) quadrado maior de lado 100 e menor de lado 50



Vejamos palavras que permitem modificar as cores do desenho.

mudecordolápis

Sintaxe: **mudecordolápis** <cor>

mudecl <cor>

Descrição: muda a cor do lápis de acordo com o conteúdo de <cor> que pode ser um número de 0 a 16, uma palavra primitiva ou uma lista composta de três números, os quais indicam a combinação das tonalidades das cores vermelho, verde e azul. Ver tabela de cores abaixo.

Exemplos: Os três comandos abaixo darão o mesmo efeito.

mudecordolápis laranja

mudecordolápis 13

mudecordolápis [255 200 0]

TABELA DE ESPECIFICAÇÃO DE CORES NO XLOGO

número	primitiva	[R G B]	Cor	número	primitiva	[R G B]	Cor
0	preto	[0 0 0]		9	cinzaclaro	[192 192 192]	
1	vermelho	[255 0 0]		10	vermelhoescuro	[128 0 0]	
2	verde	[0 255 0]		11	verdeescuro	[0 128 0]	
3	amarelo	[255 255 0]		12	azulescuro	[0 0 128]	
4	azul	[0 0 255]		13	laranja	[255 200 0]	
5	magenta	[255 0 255]		14	rosa	[255 175 175]	
6	ciano	[0 255 255]		15	violeta	[128 0 255]	
7	branco	[255 255 255]		16	marrom	[153 102 0]	
8	cinza	[128 128 128]					

As cores são definidas no XLogo com a ajuda de 3 números entre 0 e 255. Esse é o sistema de código RGB (Red, Green, Blue). Cada número corresponde respectivamente a uma intensidade de vermelho (Red), de verde (Green) e de azul (Blue) para a cor considerada. Podemos obter dessa maneira 16.777.216 de cores. Uma vez que esse sistema não é intuitivo o XLogo proporciona 17 cores pré-definidas acessíveis por um número ou um nome. Ver a tabela de especificação de cores.

mudecordofundo

Sintaxe: **mudecordofundo** <cor>

mudecf <cor>

Descrição: muda a cor do fundo da tela de acordo com o conteúdo de <cor> (ver tabela de cores).

Obs: este comando deve ser usado no início do procedimento, antes de ter qualquer desenho na tela pois este comando preenche toda tela com a nova cor, limpando tudo o que havia anteriormente.

As palavras seguintes mudam a espessura e o acabamento do lápis.

mudeespessuradolápis

Sintaxe: **mudeespessuradolápis** num

mudeel num

Descrição: muda as características do lápis de acordo com num.



mudepontadolápis

Sintaxe: **mudepontadolápis** num

mudepl num

Descrição: Altera a ponta do lápis. 0 (quadrada) e 1 (redonda).

EXEMPLOS:

<p>Os comandos limpedesenho mudeespessuradolápis 10 para frente 100 terão como resposta o desenho ao lado.</p>	
<p>Os comandos limpedesenho mudeespessuradolápis 12 mudepontadolápis 1 para frente 100 terão como resposta o desenho ao lado.</p>	

PALAVRAS PARA PINTAR O DESENHO

As palavras **pinte** e **pintezona** permitem preencher com cores o seu desenho. A tartaruga não deve estar sobre um pixel (ponto) de cor da figura que se deseja preencher (se quiser pintar de vermelho, não deverá estar sobre o vermelho).

pinte

Sintaxe: **pinte**

Descrição: A primitiva pinte colorirá todos os pixels da mesma cor daquele pixel em que ela está com a cor do lápis atual até encontrar um limite de cor diferente daquele em que ela está.

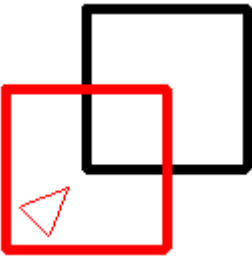
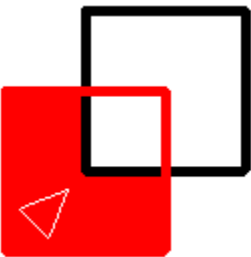
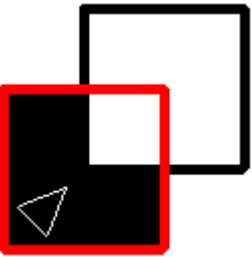
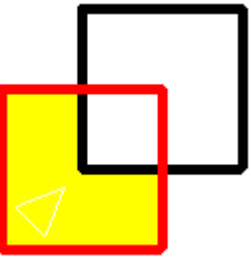
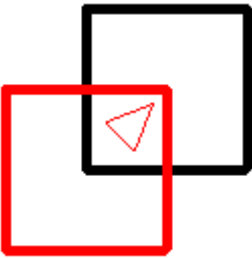
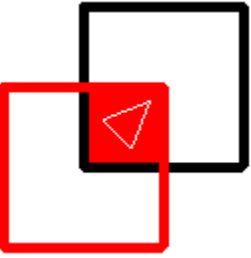
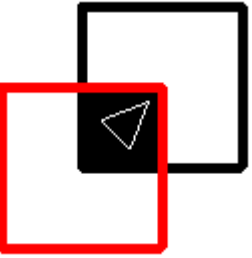
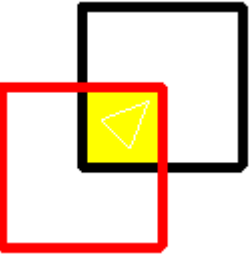
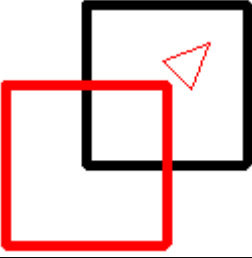
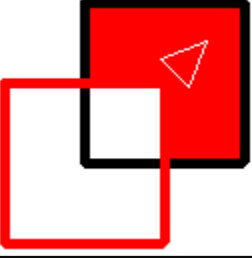
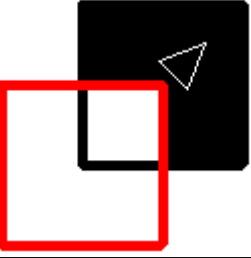
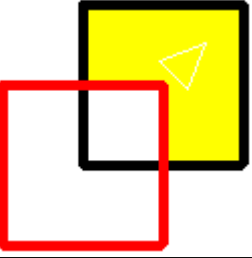
pintezona

Sintaxe: **pintezona**

Descrição: A primitiva pintezona colorirá todos os pixels da mesma cor daquele pixel em que ela está com a cor do lápis atual até encontrar um limite da cor que ela está pintando.

EXEMPLOS: Considere as seguintes situações

USANDO O PINTE

Situação inicial	Pintando de vermelho	Pintando de preto	Pintando de amarelo
			
			
			

USANDO O PINTEZONA

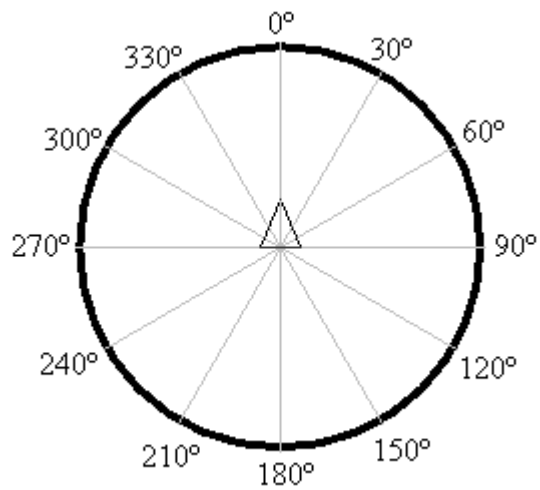
Situação inicial	Pintando de vermelho	Pintando de preto	Pintando de amarelo

Para desenhar curvas temos as palavras arco e círculo

<p>arco</p> <p>Sintaxe: arco <i>raio início fim</i></p> <p>Descrição: Desenha um arco de círculo de raio <i>raio</i> ao redor da <i>ld</i> entre os ângulos <i>início</i> e <i>fim</i> dela (a <i>ld</i> é o centro do círculo)</p>
<p>círculo</p> <p>Sintaxe: círculo <i>raio</i> circ <i>raio</i></p> <p>Descrição: Desenha um círculo de raio <i>raio</i> (a <i>ld</i> é o centro do círculo).</p>

A referência dos ângulos de uma tartaruga no comando `arco` é feita conforme a figura ao lado. Inicia com 0° em cima e cresce no sentido anti-horário.

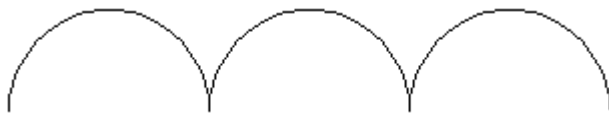
Por exemplo: Se comandarmos **arco 80 30 120** obteremos o desenho abaixo:



EXERCÍCIOS

Escreva uma sequência de comandos que faça a tartaruga desenhar cada figura dada:

a)



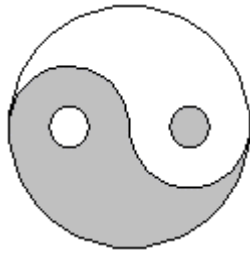
b)



c)



d)



e)



A palavra repita

repita

Sintaxe: **repita** *num* [*lista de comandos*]

Descrição: A *lista de comandos* entre os colchetes será repetida *num* vezes.

Por exemplo: Para desenhar um quadrado precisamos dar os seguintes comandos:

para frente 100 para direita 90
para frente 100 para direita 90
para frente 100 para direita 90
para frente 100 para direita 90

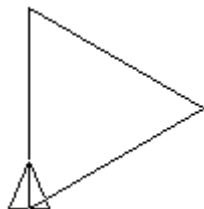
Obteríamos o mesmo resultado se usássemos a palavra **repita**. Ou seja, para desenhar um quadrado basta digitar o comando:

repita 4 [para frente 100 para direita 90]

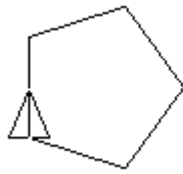
Exercício:

Utilize o comando **repita** para desenhar:

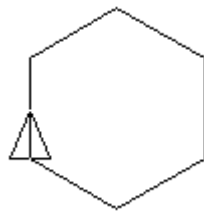
- a) Um triângulo de três lados iguais. (equilátero)



b) Um pentágono.



c) Um hexágono.



4. Programando Computadores

Já sabemos desenhar quadrados, triângulos e outras figuras. Se digitamos quadrado na linha de comandos do XLogo receberemos a seguinte mensagem: **Não sei como fazer quadrado**. Essa mensagem sugere que podemos ensinar ao computador como fazer quadrados.

Sabemos que para desenhar um quadrado de lado igual a 100 podemos digitar os oito comandos abaixo,

```
parafrente 100
paradireita 90
parafrente 100
paradireita 90
parafrente 100
paradireita 90
parafrente 100
paradireita 90
```

ou usar a palavra repita,

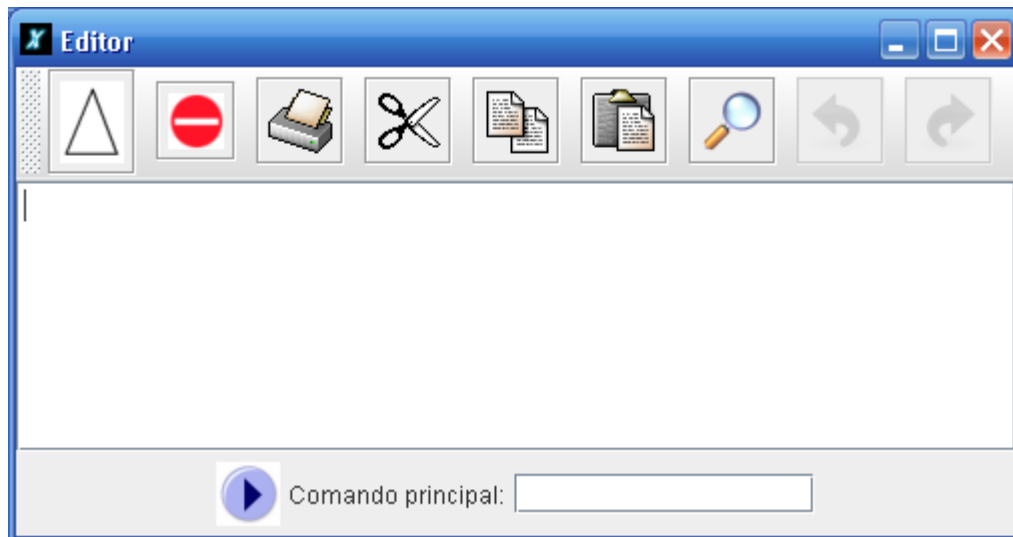
```
repita 4 [parafrente 100 paradireita 90]
```

Quando precisamos desenhar muitos quadrados a tarefa de digitar oito frases ou o comando **repita** para cada quadrado se torna muito cansativa. Podemos, e devemos, ensinar uma nova palavra ao computador: a palavra “quadrado” que significa exatamente a sequência de oito frases acima. Assim toda vez que digitarmos

“quadrado” o computador entenderá que queremos que ele desenhe um quadrado. Assim, podemos dizer que:

Programar o computador é ensiná-lo novas palavras.

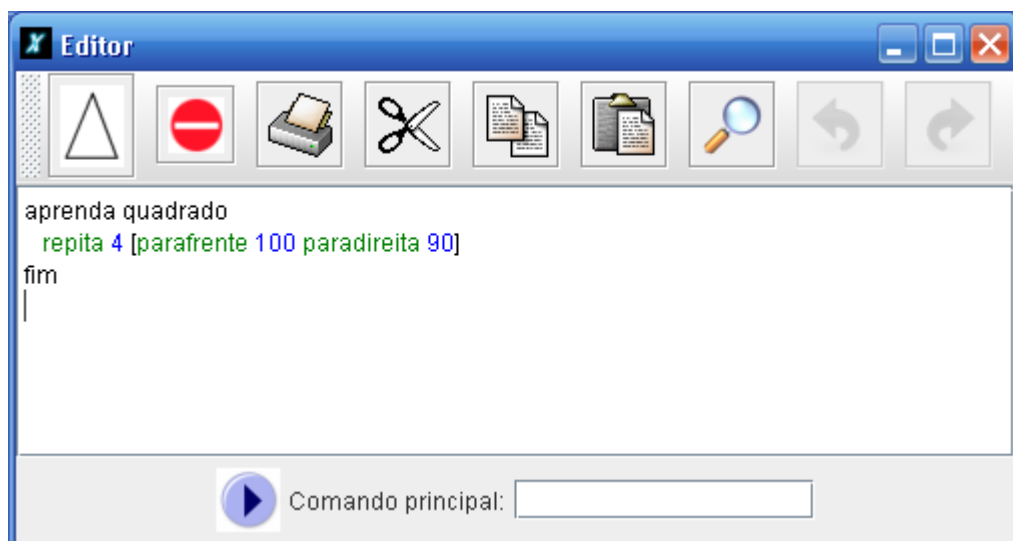
Para ensinar uma nova palavra ao computador precisamos abrir o editor clicando no botão “Editor” no ambiente XLogo. Será aberto um editor de textos simples que servirá para digitar a definição de novas palavras. Veja a figura:




Por exemplo, para ensinar a palavra quadrado devemos digitar o seguinte texto:

```
aprenda quadrado
  repita 4 [para frente 100 para direita 90]
fim
```

Veja como fica no editor:



Clique agora no botão com a imagem da tartaruga () para salvar e sair do editor. O editor será fechado e será exibida a mensagem “*Você definiu quadrado*”.

Agora podemos testar a nova palavra ensinada. Digite quadrado na linha de comandos e você verá que a tartaruga agora sabe desenhar quadrados.

Isto significa que a palavra quadrado passou a fazer parte do *vocabulário* da linguagem da tartaruga. As “novas” *palavras definidas* podem ser usadas da mesma forma que as *palavras primitivas* para definir outras palavras novas. Isto não é incrível. Uma máquina possui capacidade de aprendizagem! Tudo o que soubermos fazer podemos ensinar e o computador aprenderá.

O que chamamos de “ensinar uma nova palavra à tartaruga” é a mesma coisa que, nos livros de computação, se chama de “definir um procedimento” ou “fazer um programa”.

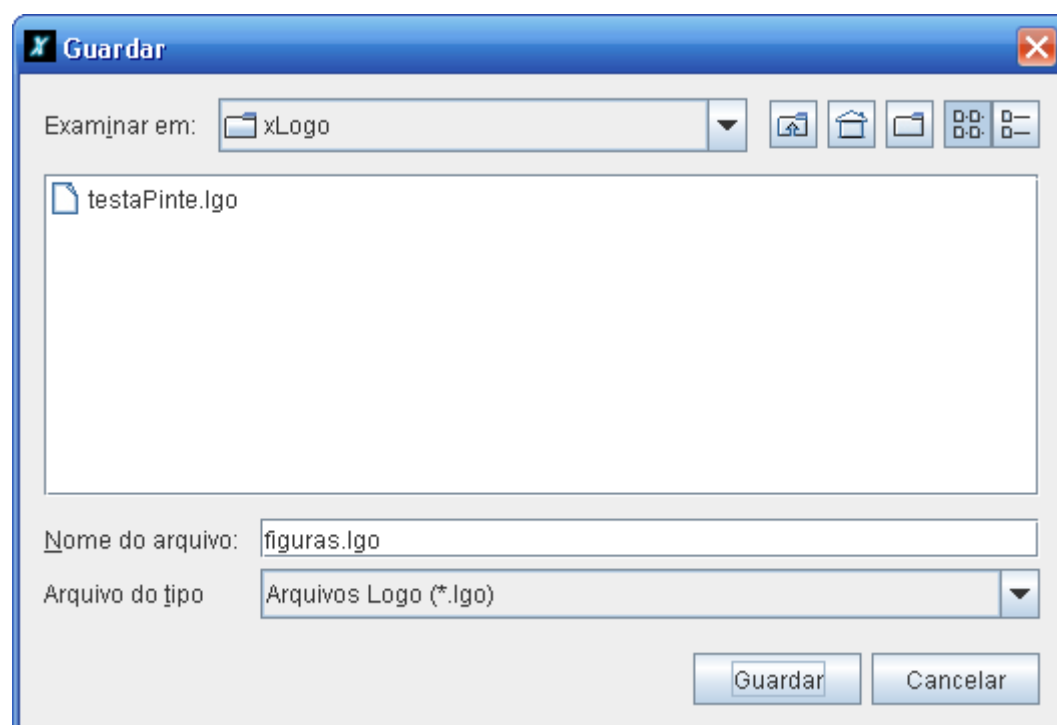
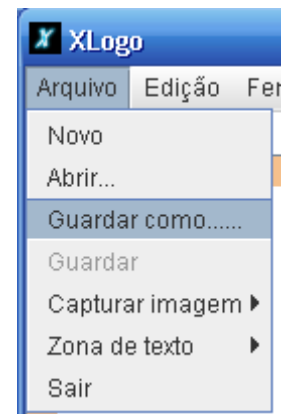
Passamos a ter dois tipos de palavras:

1. As **palavras primitivas** – aquelas que fazem parte do vocabulário da linguagem Logo. Por exemplo, **para frente**, **para direita** e outras.
2. Os **procedimentos** – as novas palavras que ensinamos ao computador. Por exemplo **quadrado**.

É importante saber que as palavras novas, ou procedimentos, precisam ser salvos em um arquivo, pois se não fizermos isso precisaremos ensinar novamente toda vez que precisarmos desenhar quadrados.

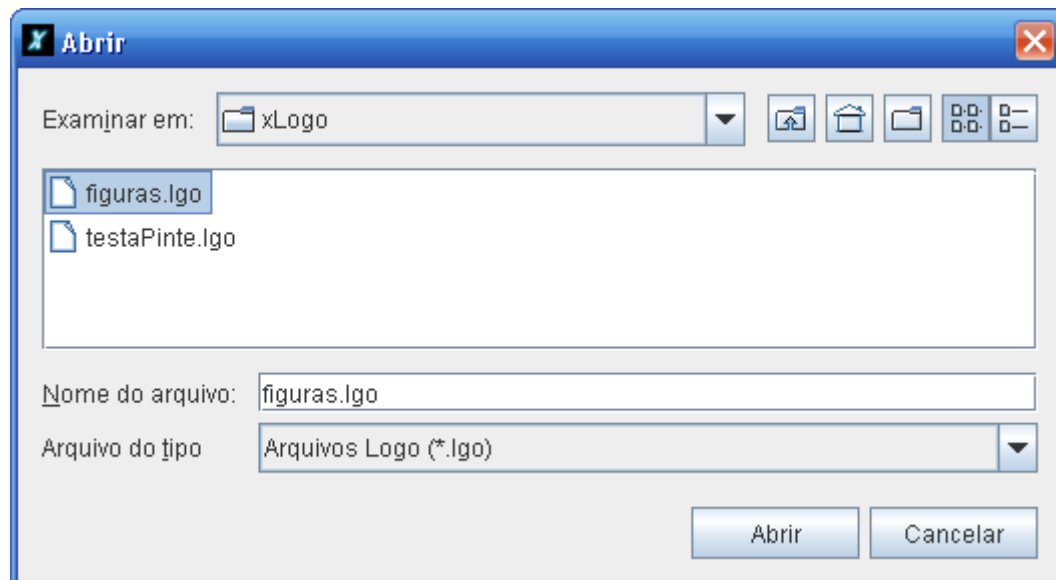
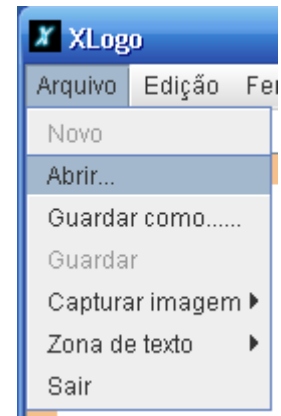
Para salvar os procedimentos que foram ensinados ao computador devemos clicar no menu **Arquivo**, clicando novamente na opção “**Guardar como ...**”. Ver figura ao lado.

Será aberta uma janela de salvamento de arquivos como a mostrada na figura.



Digite um nome para o arquivo com a extensão lgo, por exemplo, figuras.lgo e clique no botão “Guardar”. A extensão lgo identifica arquivos de código Logo assim com arquivos de extensão doc identificam arquivos do Microsoft Word.

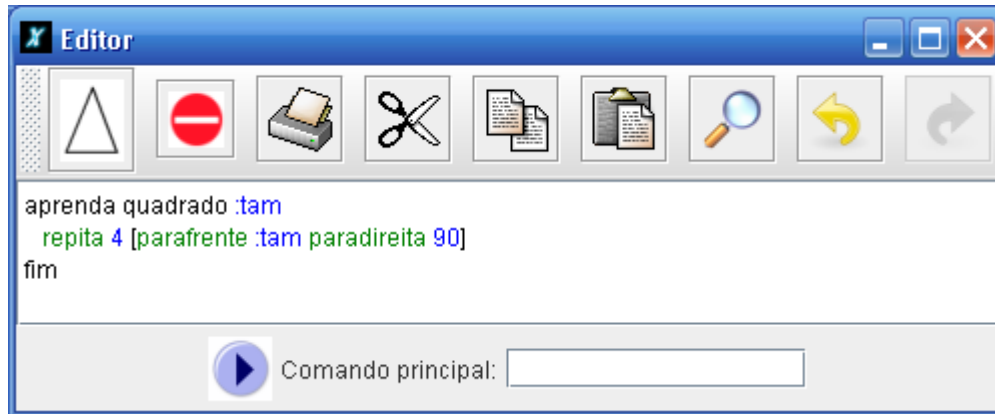
Assim, quando entramos no ambiente XLogo outra vez a palavra quadrado terá sido esquecida. Para carregar na memória a definição de quadrado clicamos no menu em “Arquivo” e selecionamos a opção “Abrir”. Ver figura ao lado. Será aberta uma janela de abertura de arquivos(ver figura) que nos permitirá selecionar o arquivo que desejamos abrir. Selecionamos com o mouse o arquivo figuras.lgo e depois clicamos em “Abrir”. Pronto, agora o computador já sabe desenhar quadrados novamente.



Alterando/Corrigindo procedimentos

Talvez já tenha passado em sua cabeça que o nosso procedimento quadrado tenha uma grave limitação. Ele só desenha quadrados de lado igual a 100. Será que podemos ter parâmetros de entrada em procedimentos como temos nas palavras primitivas? A resposta para essa pergunta é positiva. Podemos definir procedimentos na linguagem *Logo* que possuam parâmetros de entrada como qualquer outra palavra predefinida. Por exemplo, a palavra **para frente** exige um parâmetro que especifique o quanto a tartaruga deve se movimentar para frente. Assim, podemos alterar o procedimento “quadrado” para que ele peça um parâmetro contendo o tamanho do lado do quadrado.

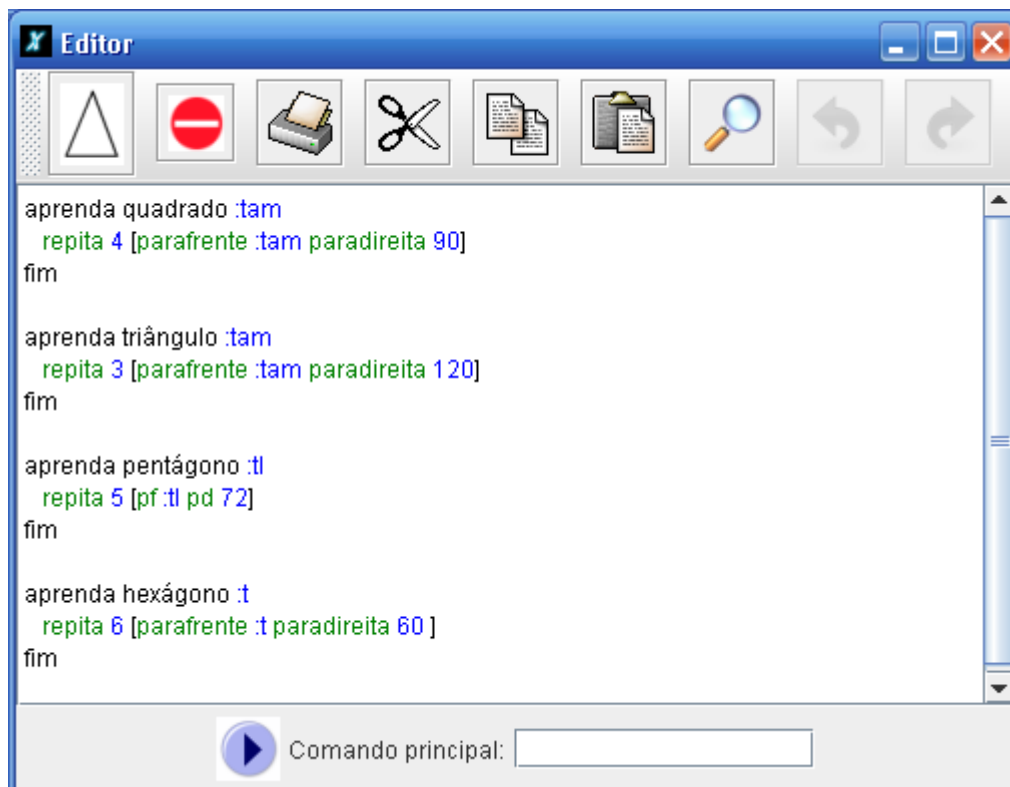
Para alterar o procedimento para que ele tenha um parâmetro de entrada que represente o tamanho do lado do quadrado devemos clicar no botão “Editor” e alterar o código do procedimento quadrado como mostrado na figura abaixo.



Inserimos um parâmetro de entrada com nome `:tam` (tamanho do lado) e substituímos o número 100 pelo parâmetro `:tam`. Na linguagem Logo os parâmetros de entrada têm seus nomes precedidos de dois pontos. Clicando no botão salvar e sair do editor as alterações são salvas e podemos testar as alterações feitas.

Para desenhar quadrados, agora, precisamos digitar um parâmetro de entrada que determina o tamanho do lado do quadrado a ser desenhado. Se esquecermos de digitar o tamanho do lado nos será dada a mesma mensagem que recebemos quando não colocamos, por exemplo, o parâmetro da palavra primitiva **para frente**.

Quando salvamos o procedimento `quadrado` em um arquivo com o nome de `figuras.lgo` e não `quadrado.lgo` tínhamos a intenção de nesse mesmo arquivo salvar outras figuras como o triângulo equilátero, o pentágono e o hexágono. Abra então o editor (clicando no botão “Editor”) e acrescente os procedimentos triângulo, pentágono e hexágono como na figura abaixo.



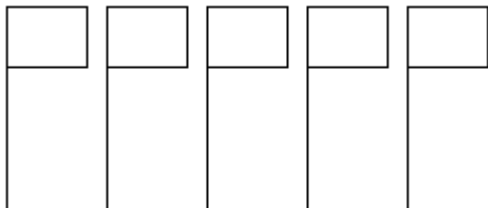


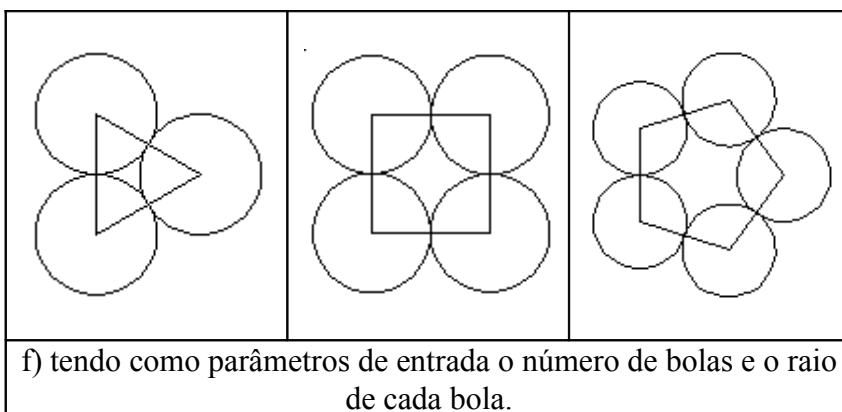
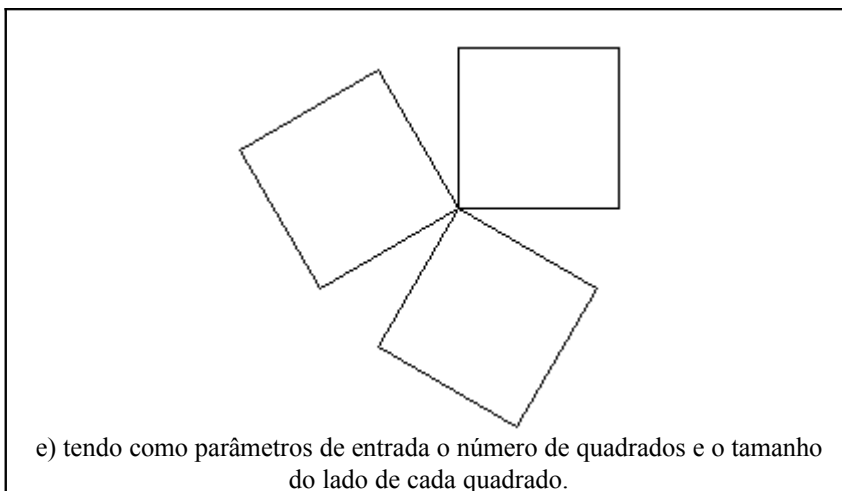
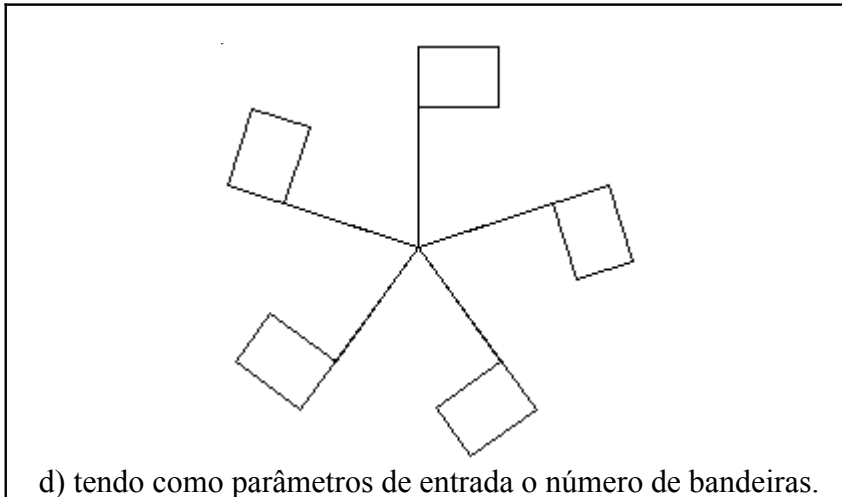
IMPORTANTE: Não esqueça de salvar as alterações no arquivo figuras.lgo clicando no menu “Arquivo” e “Guardar”.

O arquivo figuras.lgo se tornou um biblioteca de figuras. Assim quando abrimos o arquivo figuras.lgo tornamos disponíveis muitos procedimentos úteis para desenhar. Podemos acrescentar muitos outros.

Exercícios

1. Defina um procedimento que desenhe polígonos regulares tendo como parâmetros de entrada o número de lados e o tamanho de cada lado.
2. Defina procedimentos que desenhem as figuras abaixo:


a) tendo como parâmetros de entrada o número de semicírculos e o raio de cada um deles.

b) tendo como parâmetros de entrada o número de ondas.

c) tendo como parâmetros de entrada o número de bandeiras.



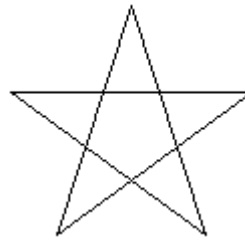
3. Ensine a tartaruga o procedimento mostrado abaixo e teste-o em seguida:

```

Aprenda raios :n :t
  repita :n [pf :t pt :t pd 360/:n]
Fim
  
```

4. Defina um procedimento que desenhe polígonos regulares dados o tamanho do lado e o número de lados como parâmetros de entrada.

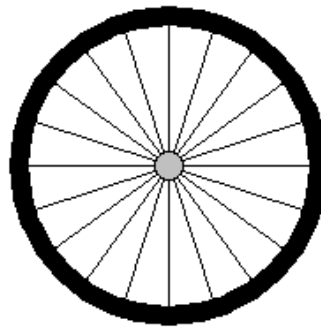
5. Defina um procedimento que desenhe a estrela ao lado tendo como parâmetro de entrada o tamanho de cada “lado” da estrela.



6. Defina um procedimento que desenhe a estrela ao lado tendo como parâmetro de entrada o tamanho de cada “lado” da estrela.

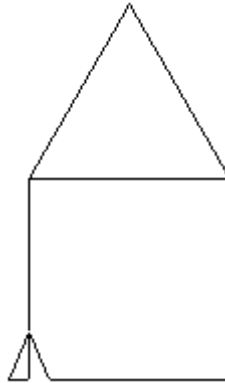


7. Defina um procedimento que desenhe uma roda de bicicleta como parâmetros de entrada a largura do pneu e o raio do aro.



5. Conceito de Subprocedimento

Como a tartaruga já sabe como desenhar quadrados e triângulos equiláteros podemos compor estes procedimentos para desenhar uma casa, como mostra a figura abaixo.

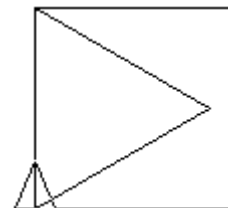


Como a casa é formada por um quadrado e um triângulo poderíamos tentar definir o procedimento casa assim:

```
aprenda casa :lado
  quadrado :lado
  triângulo :lado
fim
```

Procedimento CASA, primeira tentativa.

O resultado obtido não é o esperado como mostra a figura à direita. O procedimento desenhou um quadrado e um triângulo, mas não na posição que desejávamos. Para corrigir este problema precisamos acrescentar mais alguns comandos à definição do procedimento casa. Depois de corrigido ele se apresentaria da seguinte maneira:



```
aprenda casa :lado
  quadrado :lado
  para frente :lado
  para direita 30
  triângulo :lado
  para esquerda 30
  para trás :lado
fim
```

Procedimento CASA definitivo.

Adicionamos os comandos **para frente :lado** e **para direita 30** para posicionar a tartaruga na posição correta para iniciar a desenhar o triângulo e como consequência

da alteração do estado inicial da tartaruga provocada pela introdução destes dois comandos tivemos que introduzir os comandos **paraesquerda 30** e **paratrás :lado** para restaurar o estado original da tartaruga. É interessante que alguns procedimentos definidos não alterem o estado inicial da tartaruga quando executados.

Para definir o procedimento CASA nos utilizamos de palavras predefinidas, como **parafrente**, e dos procedimentos definidos anteriormente, QUADRADO e TRIÂNGULO. Quando um procedimento é usado como parte da definição de um novo procedimento, nós nos referimos a ele como um *subprocedimento*. Assim, os procedimentos QUADRADO e TRIÂNGULO são subprocedimentos utilizados na definição do procedimento CASA.

O procedimento CASA poderá, por sua vez, ser usado como subprocedimento na definição de outros procedimentos. Por exemplo, podemos definir a palavra RUA, procedimento que desenha um conjunto de casas, da seguinte maneira:

```

aprenda RUA :ncasas :tcasa
  repita :ncasas [
    casa :tcasa paradireita 90
    parafrente :tcasa paraesquerda 90]
  paraesquerda 90 parafrente :tcasa*:ncasas paradireita 90
fim

```

Procedimento RUA.

Onde o parâmetro *:ncasas* é igual ao número de casas que desejamos que a RUA tenha e o parâmetro *:tcasa* é o tamanho de cada casa. Vemos na figura seguinte o resultado de RUA 10 40.

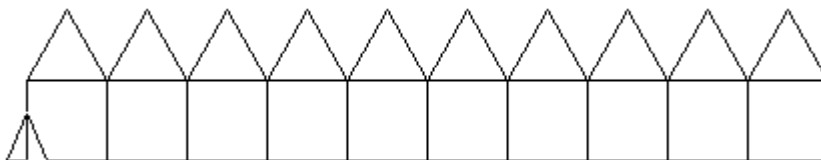


Figura gerada pelo procedimento RUA.

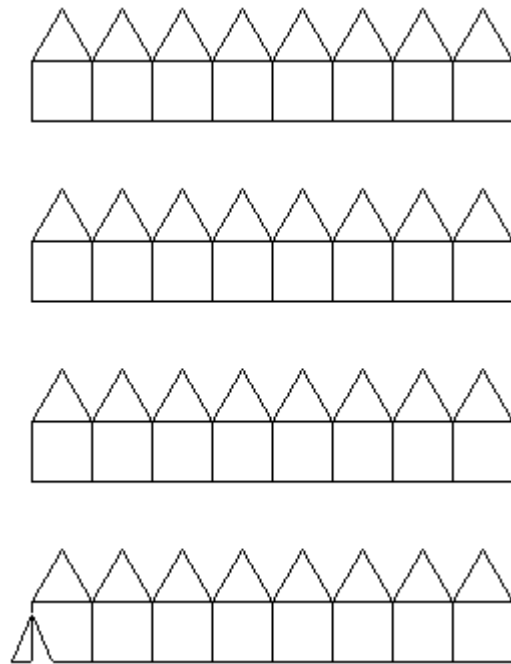
Para concluir vamos definir o procedimento cidade. Como uma cidade é composta por ruas usaremos o comando **repita** para desenhar quantas ruas sejam solicitadas para a cidade. Vejamos como fica o procedimento cidade.


```

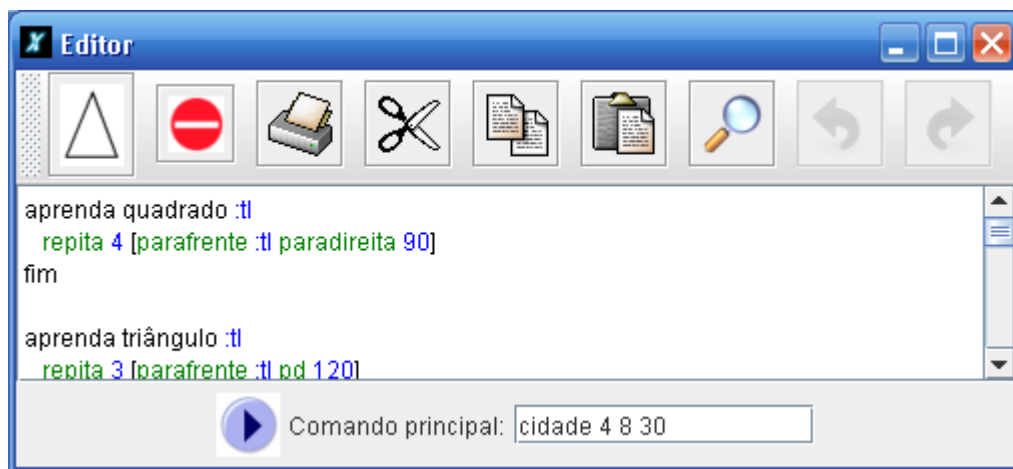
aprenda CIDADE :nrugas :ncasas :tcasa
  repita :nrugas [
    RUA :ncasas :tcasa
    usenada parafrente 3*:tcasa uselápis]
  usenada paratrás 3*:tcasa*:nrugas uselápis
fim

```

Podemos ver na figura abaixo o resultado de CIDADE 4 8 30, ou seja, uma cidade com 4 ruas, cada rua com 8 casas de tamanho 30.



Salve o seu projeto com o nome de cidade.lgo. Você pode utilizar o botão “executar”,  para executar o comando digitado no campo “Comando principal” da janela do editor. Ver figura.



Construir procedimentos usando subprocedimentos é uma das idéias mais importantes que aprendemos com a linguagem *Logo*. O conceito de subprocedimento é muito útil quando precisamos definir procedimentos muito grandes e complexos. Com ele podemos dividir um procedimento qualquer em vários subprocedimentos e trabalhar separadamente em cada um deles. Por exemplo, o procedimento CASA foi dividido em dois subprocedimentos que foram desenvolvidos separadamente e depois ajustados um ao outro para desenhar uma casa. Esta estratégia é muito importante para pensar melhor, aprender mais facilmente e resolver problemas mais eficientemente. *Construir “coisas” usando “subcoisas” é uma das idéias mais importantes e úteis que obtemos a partir do conceito de subprocedimento.*

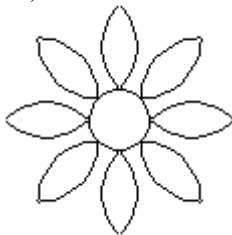
AULA PRÁTICA 3 - Conceitos de Subprocedimento.

1. Ensine à tartaruga os procedimentos quadrado, triângulo, casa, rua e cidade conforme estudados em sala de aula. Salve os procedimentos definidos em um arquivo de nome cidade.lgo.
2. (Desafio) Elabore um projeto que construa um canteiro de flores a partir dos subprocedimentos mostrados abaixo:

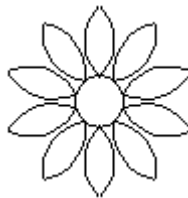
Subprocedimento 1: pétala – desenha uma pétala, como mostrado na figura ao lado. Utilizamos como parâmetro de entrada o raio de cada arco de 90° que compõe a pétala. A tartaruga deve terminar na mesma posição em que iniciou o procedimento.



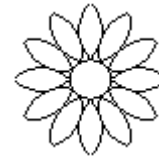
Subprocedimento 2. flor – Utilizando pétala como subprocedimento desenha uma flor, ver exemplos abaixo, tendo como parâmetros de entrada o raio de cada pétala e o número de pétalas na flor. (O raio da circunferência central é a metade do raio da pétala)



Id flor 30 8

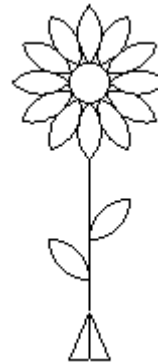


Id flor 25 10

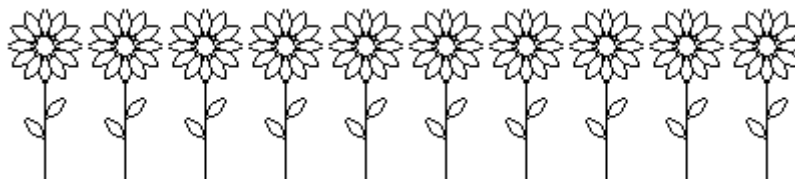


Id flor 20 12

Subprocedimento 3. planta – desenha uma flor com um talo e duas flores, conforme a figura ao lado, tendo como parâmetros de entrada o raio de cada pétala e o número de pétalas na flor. O tamanho do talo é cinco vezes o raio da pétala. A tartaruga inicia e termina na posição indicada na figura.



Procedimento canteiro - Desenha um canteiro, como mostrado abaixo, usando como subprocedimento planta e tendo como parâmetros de entrada o número de flores, o número de pétalas em cada flor e o raio da pétala.



3. Modifique o seu canteiro introduzindo cores nas flores.

4. Torne mais bonita a cidade fazendo com que cada casa tenha um canteiro a sua frente.

6. Conceito de Variável

Através da definição de procedimentos com parâmetros entramos em contato com uma noção que é muito importante na matemática e na computação: o conceito de *variável*. Na matemática o conceito de variável se refere a uma quantidade capaz de assumir qualquer valor dentro de um determinado conjunto. Em computação, *variável* é o nome que é associado a uma porção de memória do computador onde é armazenada uma informação que pode assumir qualquer valor dentro de um determinado domínio. Na linguagem *Logo*, por exemplo, uma variável pode indicar o armazenamento de informações em forma de *números*, *nomes* ou *listas*. Estas informações podem ser, facilmente, consultadas ou alteradas.

Na linguagem da tartaruga, podemos criar variáveis, ou seja, reservar espaço na memória do computador para armazenar informações, de duas maneiras. A primeira é através da definição de um procedimento com parâmetros de entrada, como vimos anteriormente. Cada parâmetro de entrada é uma variável. Por exemplo, o parâmetro *lado* é uma variável criada quando definimos o procedimento QUADRADO :*lado*. Na linguagem da tartaruga, quando queremos nos referir ao valor da variável, escrevemos o nome da variável precedido por dois pontos, como no exemplo dado, onde :*lado* representa o valor contido na variável *lado*. As variáveis criadas pela definição de um procedimento são *locais*, ou seja, elas só podem ser usadas dentro do procedimento que as criou.

A segunda maneira de criarmos variáveis é através da palavra predefinida **atribua**, que tem o seguinte formato:

atribua <nome> <conteúdo>

O comando **atribua** cria uma variável, se ela ainda não existir, designada por <nome> e atribui a ela o valor de <conteúdo>. Se a variável já existir, o comando simplesmente atribui à variável <nome> o valor de <conteúdo>. Por exemplo, a expressão:

atribua "tamanho 100

cria uma variável cujo nome é "tamanho" com valor igual a 100, se a variável tamanho, ainda, não existir, ou, caso contrário, apenas lhe atribui o valor 100.

Na linguagem *Logo* é utilizada a seguinte convenção simbólica para evitar ambigüidade:

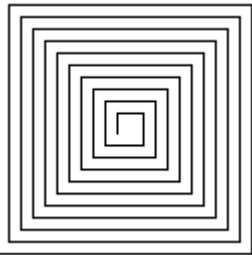
- Um nome precedido por dois pontos significa o valor da variável que possui aquele nome. Por exemplo, se pedirmos à tartaruga **mostre** :*tamanho*, ela mostrará o valor 100.

- Um nome precedido por aspas duplas representa o próprio nome. Por exemplo: Se pedirmos à tartaruga **mostre** “tamanho, ela mostrará a palavra “tamanho”.
- Um nome sem dois pontos ou aspas no início representa um procedimento ou comando da linguagem.

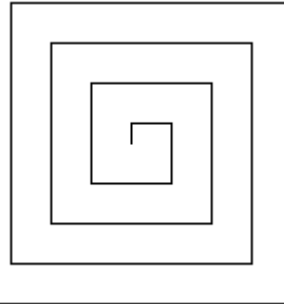
EXEMPLOS DE PROCEDIMENTOS GRÁFICOS COM VARIÁVEIS

O procedimento abaixo desenha espirais quadradas

```
aprenda espiralQuadrada :ns :ts :in
  repita :ns [
    para frente :ts
    para direita 90
    atribua "ts :ts+:in
  ]
fim
```



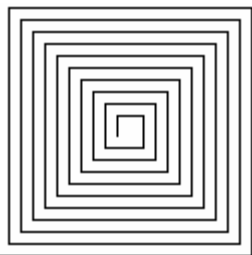
ld espiralQuadrada 40 10 3



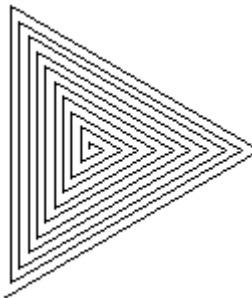
ld espiralQuadrada 16 10 10

O procedimento abaixo faz a tartaruga desenhar espirais poligonais

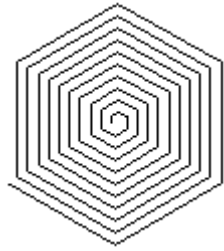
```
aprenda espiral :ns :ts :an :in
  repita :ns [
    para frente :ts
    para direita :an
    atribua "ts :ts+:in
  ]
fim
```



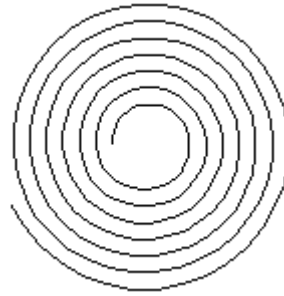
ld espiral 40 3 90 3



ld espiral 30 3 120 5



ld espiral 60 3 60 1

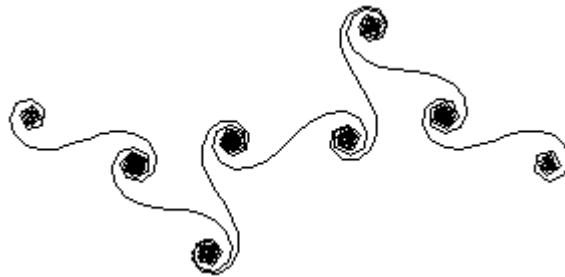


ld espiral 250 3 10 0.04

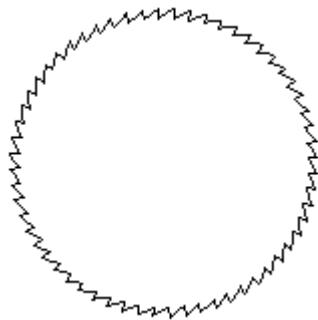
O procedimento abaixo desenha figuras muito interessantes variando o ângulo a cada iteração.

```
aprenda inspi :n :t :a :inc  
  repita :n [para frente :t para direita :a atribua "a :a+:inc]  
fim
```

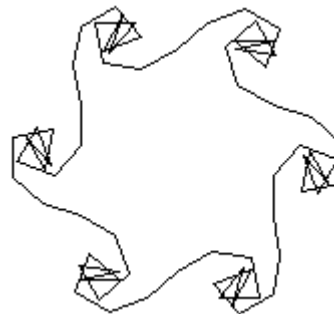
Vejamos alguns exemplos:



ld pd 45 inspi 1000 8 0 7



ld inspi 1000 4 2 120



ld inspi 1000 20 40 30

EXEMPLOS DE PROCEDIMENTOS NUMÉRICOS COM VARIÁVEIS

PROCEDIMENTO 1 - Progressão Aritmética

Descrição: Progressão aritmética (PA) é toda sequência na qual cada termo, a partir do segundo, é a soma do termo anterior com uma constante dada.

```
aprenda progArit :nt :ti :ra
  repita :nt [
    mostre :ti
    atribua "ti :ti+:ra]
fim
```

Onde :nt é o número de termos da PA que se mostrará.
:ti é o primeiro termo da PA
:ra é a razão da PA

Execute o procedimento digitando na linha de comandos, por exemplo,
progArit 10 3 5

obtendo como resposta a seguinte sequência

3
8
13
18
23
28
33
38
43
48

A palavra **mostre**

mostre

Sintaxe: **mostre** *valor*

mo *valor*

Descrição: Mostra *valor* na janela de texto e passa para a próxima linha. Se *valor* for uma expressão, será avaliada e mostrado o seu resultado.

Exemplos:

mostre 5 - mostrará 5
mostre 5+3 - mostrará 8

A palavra **escreva**

escreva

Sintaxe: **escreva** *valor*

esc *valor*

Descrição: Mostra *valor* na janela de texto e NÃO passa para a próxima linha. Se *valor* for uma expressão, será avaliada e mostrado o seu resultado.

PROCEDIMENTO 2 - Progressão Geométrica.

Descrição: Progressão geométrica (PG) é toda sequência na qual cada termo, a partir do segundo, é o produto do termo anterior por uma constante dada (q).

```
aprenda proggeom :nt :ti :ra
  repita :nt-1 [
    escreva :ti
    escreva ", \"
    atribua "ti :ti*:ra]
  mostre :ti
fim
```

Onde :nt é o número de termos da PG que se mostrará.
:ti é o primeiro termo da PG
:ra é a razão da PG

Execute o procedimento digitando na linha de comandos, por exemplo,
progGeom 7 13 7
obtendo como resposta a seguinte sequência
13,91, 637, 4459, 31213, 218491, 1529437

PROCEDIMENTO 3 - Quadrados Perfeitos

Descrição: O procedimento mostra os n primeiros quadrados perfeitos a partir do 1.

```
aprenda quadperf :n
  atribua "x 1
  repita :n [
    mostre :x*:x
    atribua "x :x+1]
  fim
```

Execute o procedimento digitando na linha de comandos, por exemplo,
quadperf 10
obtendo como resposta a seguinte sequência

1 4 9 16 25 36 49 64 81 100

PROCEDIMENTO 4 - Multiplicação por somas sucessivas

Descrição: Podemos fazer o produto de dois números a e b somando a , b vezes ou somando b , a vezes, como mostra a equação abaixo.

$$a \times b = \underbrace{a + a + \dots + a}_{b \text{ vezes}} = \underbrace{b + b + \dots + b}_{a \text{ vezes}}$$

```

aprenda multiplica :a :b
  atribua "resp 0
  repita :a [atribua "resp :resp+:b]
  saída :resp
fim

```

Execute o procedimento digitando na linha de comandos, por exemplo,

mostre multiplica 12 11

obtendo como resposta 132.

É necessário utilizar a palavra **mostre** pois o procedimento multiplica retorna um valor de saída. Se a omitirmos recebemos a mensagem: **Não sei o que fazer com 132 ?**

A palavra **saída**

saída

Sintaxe: **saída** valor_de_saída

Descrição: Caracteriza que um procedimento executa uma tarefa e retorna um valor de saída.

Um procedimento que retorna um valor recebe o nome de **função**. Uma função pode ser usada numa expressão aritmética, um procedimento não. Por exemplo, a expressão aritmética $10+5 \times 3$ pode ser avaliada em Logo por `10+multiplica 5 3`. Teste-a digitando **mostre** `10+multiplica 5 3`.

PROCEDIMENTO 5 - Potência

Descrição: Podemos fazer a operação a^b (a elevado a b) multiplicando a por ele mesmo b vezes.

$$a^b = \underbrace{a \times a \times \dots \times a}_{b \text{ vezes}}$$

1ª versão

```

aprenda pot :a :b
  atribua "resp 1
  repita :b [atribua "resp :resp*:a]
  saída :resp
fim

```

2ª versão

```

aprenda pot2 :a :b
  atribua "resp 1
  repita :b [atribua "resp multiplica :resp :a]
  saída :resp
fim

```

Execute o procedimento digitando na linha de comandos, por exemplo, **mostre pot 3 5** obtendo como resposta 243.

PROCEDIMENTO 6 - Fatorial

Descrição: Fatorial de um número inteiro n é o produto dos números naturais desde 1 até n .

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

1ª versão

```
aprenda fatorial :n
atribua "resp 1
repita :n [atribua "resp :resp*:n atribua "n :n-1]
saída :resp
fim
```

2ª versão

```
aprenda fatorial :n
atribua "resp 1
repita :n [atribua "resp :resp*cv]
saída :resp
fim
```

Execute o procedimento digitando na linha de comandos, por exemplo,
mostre fatorial 4
obtendo como resposta 24.

A palavra **contevezes**

contevezes

Sintaxe: **contevezes**

cv

Descrição: Palavra primitiva que retorna o número de vezes que um repita já executou. Só funciona entre os colchetes da palavra primitiva **repita**.

PROCEDIMENTO 7 - Seqüência de Fibonacci

Descrição: Seqüência de Fibonacci (SF) é toda aquela na qual cada termo, a partir do terceiro, é a soma dos dois termos anteriores. O procedimento seguinte mostra os n primeiros termos da seqüência de Fibonacci.


```
aprenda fibonacci :n
  atribua "p 1
  atribua "s 1
  mostre :p
  mostre :s
  repita :n-2 [
    atribua "t :s
    atribua "s :s+:p
    atribua "p :t
    mostre :s]
fim
```

Execute o procedimento digitando na linha de comandos, por exemplo,
fibonacci 10
obtendo como resposta a seguinte sequência: 1 1 2 3 5 8 13 21 34 55

7. Conceitos de Estado e de Operadores de Mudança de Estado.

Podemos descrever o *estado* da tartaruga através de alguns indicadores. Por exemplo:

1. sua *posição* - onde ela está localizada,
2. sua *orientação* - para que direção ela está voltada, e,
3. a *situação do lápis* - abaixado ou levantado.
4. a *cor do lápis* utilizado pela tartaruga.
5. a *espessura do lápis* usado para desenhar.

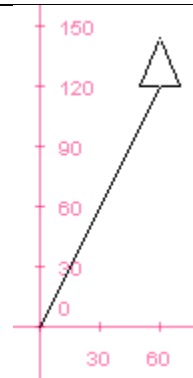
Assim, alguns dos comandos da linguagem da tartaruga, que vimos até agora, podem ser considerados como *operadores de mudança de estado*. Os comandos **para frente** e **para trás** mudam o estado da tartaruga atuando sobre a sua *posição*. Os comandos **para direita** e **para esquerda** alteram o estado da tartaruga modificando a sua *orientação*. E finalmente, os comandos **uselápis**, **usenada**, **mudecl** e **mudeel** modificam a *situação do lápis*, alterando assim, também, o estado da tartaruga. Percebemos claramente que cada comando altera apenas um, e não mais do que um, dos indicadores de estado de cada vez.

Mostramos a seguir outros operadores de mudança de estado:

mudepos

Sintaxe: **mudepos** *lista*

Descrição: movimenta a tartaruga para uma posição absoluta da tela. O argumento é uma *lista* de dois números, onde primeiro representa a coordenada X e o segundo representa a coordenada Y.



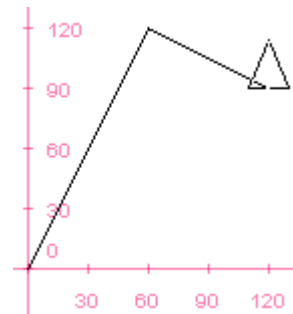
Exemplo: `ld mudepos [60 120]`

Observação: Os eixos cartesianos são exibidos clicando no menu “Ferramentas”, em “Preferências” e na aba “Opções”, na moldura “Eixo no fundo” marcamos as opções “Eixo X” e “Eixo Y”.

mudexy

Sintaxe: **mudexy** *número1* *número2*

Descrição: movimenta a tartaruga para uma posição absoluta de tela. O argumento são dois números onde *número1* representa a coordenada X e *número2* representa a coordenada Y.

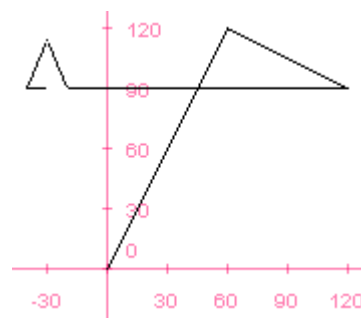


Exemplo: `mudexy 120 90`

mudex

Sintaxe: **mudex** *número*

Descrição: movimenta a tartaruga até o ponto com coordenada X especificado por *número*, o Qual representa a coordenada X, mantendo inalteradas sua coordenada Y e sua direção.

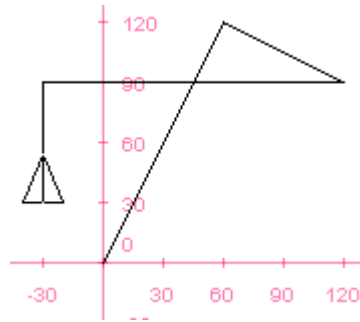


Exemplo: `mudex -30`

mudey

Sintaxe: **mudey** *número*

Descrição: movimenta a tartaruga para o ponto especificado por *número*, o qual representa a coordenada Y, mantendo inalteradas sua coordenada X e sua direção.

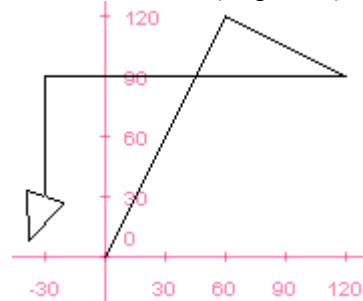


Exemplo: mudey 30

mudedireção

Sintaxe: **mudedireção** *número*
mudedç *número*

Descrição: muda a direção da tartaruga para *número* correspondente em graus. A direção 0 corresponde ao Norte (para cima), 90 corresponde a Leste (direita), 180 ao Sul (para baixo) e 270 ao Oeste (esquerda) .

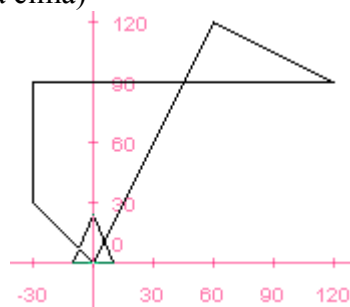


Exemplo: mudedireção 200

centro

Sintaxe: **centro**

Descrição: movimenta a tartaruga para o centro da tela (posição [0 0]) e muda sua direção para 0 (olhando para cima)



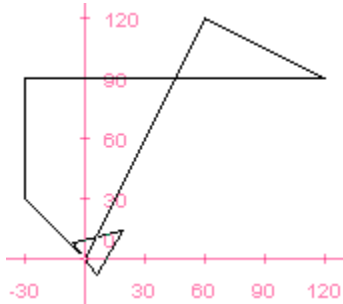
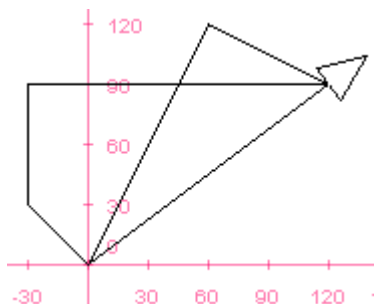
Exemplo: centro

direçãopara

Sintaxe: **direçãopara** *lista*
Dçpara *lista*

Descrição: Retorna a direção da tartaruga para o ponto indicado em *lista*.

Exemplo: mostre direçãopara [120 90]
53.13010235415598

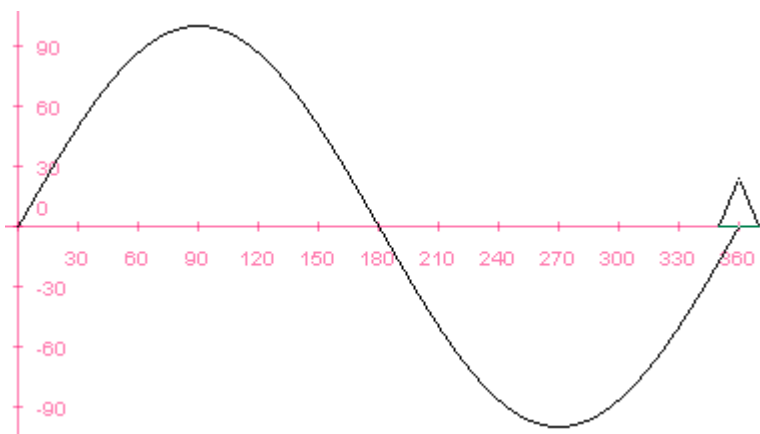
 <p>mude direção direção para [120 90]</p>	<p>distância</p> <p>Sintaxe: distância <i>lista</i></p> <p>Descrição: Retorna a distância entre a posição da tartaruga e o ponto indicado em <i>lista</i>.</p> <p>Exemplo: mostre distância [120 90] 150</p>  <p>para frente distância [120 90]</p>
<p>pos</p> <p>Sintaxe: pos</p> <p>Descrição: Retorna a posição da tartaruga.</p> <p>Exemplo: mostre pos 120 90</p>	<p>direção</p> <p>Sintaxe: direção dc</p> <p>Descrição: Retorna a direção da tartaruga.</p> <p>Exemplo: mostre direção 53.13010235415595</p>

Exemplos

1. O procedimento abaixo desenha a curva da função seno.

```

aprenda grafseño
atribua "x 1
usenada centro uselápis
repita 360 [
  mudexy :x 100*sen :x
  atribua "x :x+1]
fim
  
```



2. Podemos inserir alguns parâmetros de entrada para o procedimento `grafseno` no procedimento. O parâmetro `:amp` é a amplitude, `:freq` é a frequência e `:fase` é a fase da senoide.

```

aprenda grafseno :amp :freq :fase
atribua "x 0
usenada mudexy :x :amp*sen :freq*:x+:fase uselápis
repita 361 [
  mudexy :x :amp*sen :freq*:x+:fase
  atribua "x :x+1]
fim
  
```

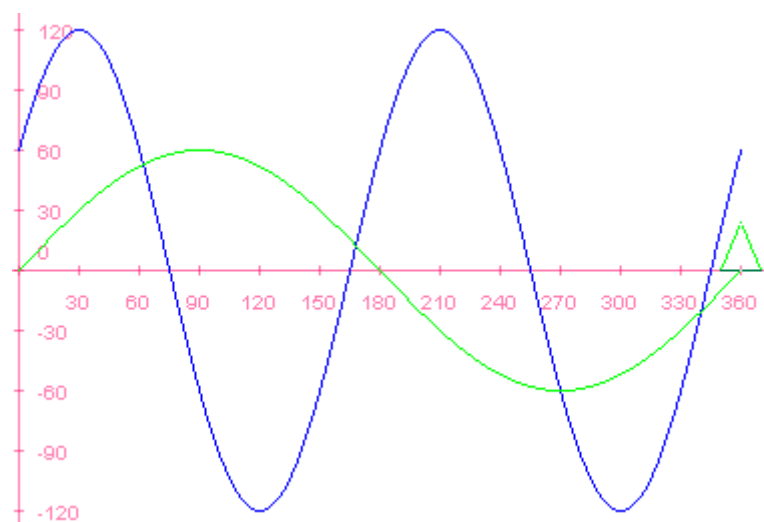
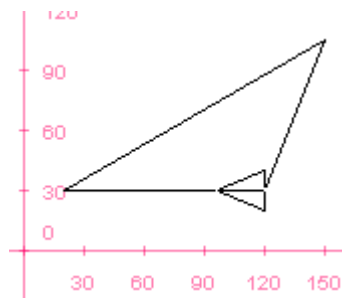


Gráfico obtido com **ld mudecl azul grafseno 120 2 30 mudecl verde grafseno 60 1 0**

3. O procedimento abaixo desenha um triângulo qualquer dados o tamanho de dois de seus lados e o ângulo formado por eles.

```
aprenda triângulo :lado1 :lado2 :ang
  atribua "posini pos
  atribua "dçini dç
  para frente :lado1 para direita 180-:ang para frente :lado2
  mude pos :posini
  mude dç :dçini
fim
```

Exemplo: `ld usenada mudexy 120 30 mudedç 270 uselápis triângulo 100 150 30`



Exercício

Defina um procedimento que desenhe parábolas tendo como parâmetros de entrada os coeficientes da equação do segundo grau (a, b e c).

8. EXPRESSÕES ARITMÉTICAS

Uma expressão aritmética é formada pela combinação de operadores, funções aritméticas e operandos que podem ser constantes e/ou variáveis numéricas.

Operadores aritméticos da linguagem Logo.

Os operadores aritméticos realizam as operações aritméticas básicas. Para cada operador aritmético existe uma função equivalente.

Operação	Operador	Função	Exemplos
Adição	+	soma	mostre 3+4 \equiv mostre soma 3 4
Subtração	-	diferença	mostre 3-4 \equiv mostre diferença 3 4
Multiplificação	*	produto	mostre 3*4 \equiv mostre produto 3 4
Divisão	/	quociente (divisão inteira)	O operador / e a função quociente não são equivalentes. O operador realiza a divisão real e a função realiza a divisão inteira. Por exemplo: mostre 13/5 mostrará 2.6 mostre quociente 13 5 mostrará 2

Funções matemáticas predefinidas da linguagem Logo.

Na formação de uma expressão matemática podemos usar funções matemáticas predefinidas da linguagem *Logo*. Listamos abaixo essas dessas funções:

Função	Sintaxe	Descrição	Exemplo
absoluto	absoluto a abs a	Retorna o valor absoluto de a	mostre abs -23 23
arredonde	arredonde a	Retorna a parte inteira de a arredondando	mostre arredonde 21.6 22
inteiro	inteiro a	Retorna a parte inteira de a	mostre inteiro 21.6 21
log10	log10 a	Retorna o logaritmo de a na base 10	mostre log10 100 2
menos	menos a	Retorna a de sinal trocado	mostre menos 23 -23
pi	pi	Retorna o valor de π	mostre pi 3.141592653589793
potência	potência a b	Retorna a elevado à potência b	mostre potência 2 3 8
raizq	raizq a	Retorna a raiz quadrada de a	mostre raizq 9 3

resto	resto a b	Retorna o resto de a dividido por b	mostre resto 10 3 1
sorteie	sorteie a	Retorna um número inteiro positivo aleatório entre 0 e a-1	mostre sorteie 10 (mostrará um número aleatório entre 0 e 9)
coseno	coseno a cos a	Retorna o coseno de a	mostre coseno 30 0.8660254037844387
seno	seno a sen a	Retorna o seno de a	mostre seno 30 0.49999999999999994
tangente	tangente a tan a	Retorna a tangente de a	mostre tangente 45 0.9999999999999999
arccoseno	arccoseno a acos a	Retorna o ângulo em graus cujo coseno é a	mostre arccoseno 0.8 36.86989764584401
arcseno	arcseno a asen	Retorna o ângulo em graus cujo seno é a	mostre arcseno 0.5 30.0000000000000004
atan	atan a	Retorna o ângulo em graus cuja tangente é a	mostre atan 0.8 38.659808254090095

Exercícios

Determine o resultado de cada expressão matemática:

- atribua "a 5 atribua "b 3 mostre :a+:b
- atribua "a 5 atribua "b 3 mostre :a+:b*:b
- atribua "a 5 atribua "b 3 mostre :a*:a+:b
- atribua "a 5 atribua "b 3 mostre (:a+:b)*:b
- atribua "a 5 atribua "b 3 mostre :a*(:a+:b)
- mostre inteiro 2/3
- mostre inteiro 2/3*100
- mostre (inteiro 2/3*100)/100
- mostre arredonde 2/3
- mostre arredonde 2/3*100
- mostre (arredonde 2/3*100)/100
- mostre resto 18 5
- mostre resto 5 18
- mostre raizq absoluto menos -9

9. EXPRESSÕES LÓGICAS.

Uma expressão lógica é aquela formada por operadores lógicos ou relacionais, funções lógicas e operandos que podem ser variáveis e/ou constantes do tipo lógico. O resultado de uma expressão lógica é sempre *verd* ou *falso*.

Operadores Relacionais da linguagem Logo.

Utilizamos os operadores relacionais para realizar comparações entre dois valores. A tabela abaixo mostra os operadores relacionais da linguagem *Logo*.

Operação de comparação	Operador	Exemplos
É igual?	=	mostre 3=4 falso
É diferente?	não ... = ...	mostre não 3=4 verd
É maior que?	>	mostre 3>3 falso
É maior ou igual a?	>=	mostre 3>=3 verd
É menor que?	<	mostre 3<3 falso
É menor ou igual a?	<=	mostre 3<=3 verd

Operadores Lógicos da linguagem Logo.

Os operadores lógicos da linguagem *Logo* são:

Operação Lógica	Operador	Função	Exemplos
Conjunção (e)	&	e	mostre 3>2 & 4=4 verd mostre e 3>2 4=4 verd
Disjunção (não exclusiva)		ou	mostre 3<2 4=5 falso mostre ou 3<=2 4=5 falso
Negação		não	mostre não 3=4 verd

Abaixo listamos algumas das funções lógicas primitivas da linguagem XLogo. Uma função lógica ou booleana retorna *verd* ou *falso*.

Função	Sintaxe	Descrição	Exemplo
éinteiro?	éinteiro a	Retorna verd se a for um número inteiro; falso, se não	mostre éinteiro? 5 verd
éprimitiva?	éprimitiva? "a eprim? "a	Retorna verd se "a for uma primitiva do XLogo.	mostre éprimitiva? "pf verd
éprocedimento?	éprocedimento? "a eproc? "a	Retorna verd se "a for um procedimento.	mostre eproc? "quadrado verd

éuselápis?	éuselápis?	Informa verd se o lápis estiver sendo usado	un mostre éuselápis? falso ul mostre éuselápis? verd
visível?	visível?	Retorna verd se a tartaruga estiver visível	dt mostre visível? falso

Exercícios:

Determine o resultado de cada expressão lógica dada:

- a) ou $2=3$ não $2=3$
- b) ou $2=3$ não $3=3$
- c) ou ou $2=3$ não $3=3$ $3=3$
- d) ou & $2=3$ não $3=3$ $3=3$
- e) & ou $2=3$ não $3=3$ $3=3$
- f) atribua "a 3 atribua "b 2 mostre não :a>:b
- g) atribua "a 2 atribua "b 3 mostre não :a>:b
- h) atribua "a 2 atribua "b 3 mostre e ou não :a>:b :a=:b :b>:a
- i) atribua "a 2 atribua "b 3 mostre ou e não :a>:b :a=:b :b>:a

10. ESTRUTURAS DE CONTROLE

As linguagens de programação de alto nível, como por exemplo *Logo* e *Java*, possuem comandos que controlam se um grupo de instruções deve ou não ser executado e o número de vezes que será executado. Já estamos familiarizados com uma dessa estruturas de controle: o comando **repita**.

Existem dois tipos de estruturas de controle:

1. Estruturas de controle de decisão
2. Estruturas de controle de repetição

As do primeiro tipo decidem se um grupo de instruções será ou não executado dependendo do resultado de uma expressão lógica. As do segundo controlam o número de vezes que uma lista de instruções será executado.

ESTRUTURAS DE CONTROLE DE DECISÃO

Uma estrutura de controle de decisão avalia uma expressão lógica e dependendo do resultado (falso ou verdadeiro) decide se um grupo de instruções será executado ou não.

Na linguagem do ambiente XLogo existem dois tipos de estruturas de controle de decisão: Simples e Composta

ESTRUTURA DE CONTROLE DE DECISÃO SIMPLES

Sintaxe:

```
se <expressão lógica> [<lista de instruções>]
```

A <lista de instruções> será executada se o resultado da <expressão lógica> for verdadeiro.

EXEMPLO: Considere esta nova versão do procedimento fatorial.

```
aprenda fatorial :n
  se :n<0 [saída [Entre com um número positivo]]
  atribua "resp 1
  repita :n [atribua "resp :resp*:n atribua "n :n-1]
  saída :resp
fim
```

Antes de fazer o cálculo do fatorial o procedimento verifica se o parâmetro de entrada é negativo. Se for negativo será retornada uma mensagem ao usuário.

ESTRUTURA DE CONTROLE DE DECISÃO COMPOSTA

Sintaxe:

```
se <expressão lógica> [<lista 1>] [<lista 2>]
```

A lista de instruções <lista 1> será executada se o resultado da <expressão lógica> for verdadeiro. Se for falso será executada <lista 2>.

EXEMPLO:

```
aprenda multiplica :a :b
  atr "r 0
  se :a>:b [repita :b [atr "r :r+:a]] [repita :a [atr "r :r+:b]]
  saída :r
fim
```

Para melhorar a performance do procedimento a estrutura de controle de decisão verifica se o parâmetro a é maior que b para obter o menor número de repetições no cálculo da multiplicação.

EXEMPLO 2: Raízes de uma equação do segundo grau:

```
aprenda raízes :a :b :c
  atribua "d :b*b-4*:a*:c
  se :d<0 [mostre [não existem raízes reais]]
  se :d=0 [mostre (menos :b)/(2*:a)]
  se :d>0 [
    mostre ((-:b)+raizq :d)/(2*:a)
    mostre ((-:b)-raizq :d)/(2*:a)]
```

fim

Exercício:

Tendo como parâmetros de entrada a altura e o sexo de uma pessoa, defina um procedimento que calcule o seu peso ideal, utilizando as seguintes fórmulas:

- para homens: $72,7 \times h - 58$;
- para mulheres $62,1 \times h - 44,7$.

onde h representa a altura em metros.

ESTRUTURAS DE CONTROLE DE REPETIÇÃO.

As estruturas de controle de repetição controlam o número de vezes que um grupo de instruções será executado. Estas estruturas podem ser de dois tipos:

1. Com número de repetições conhecido previamente.
2. Com número de repetições dependente de um teste.

Já estamos familiarizados com uma delas: o **repita**, cuja sintaxe é mostrada abaixo:

repita <nº de repetições> [<lista de instruções>]

A <lista de instruções> será executada o número de vezes especificado em <nº de repetições>. Percebe-se claramente que o comando **repita** pertence ao primeiro tipo, ou seja, ele se aplica quando conhecemos antecipadamente o número de vezes que a lista de instruções deverá ser repetida.

EXEMPLO 1

```
aprenda somapg :n :t :r
  atribua "resp 0
  repita :n [atribua "resp :resp+:t atribua "t :t*:r]
  saída :resp
fim
```

Este procedimento calcula a soma dos n primeiros termos de uma progressão geométrica dados o primeiro termo (:t) e a razão (:r).

EXEMPLO 2

```
aprenda polígono :n :t
  repita :n [para frente :t para direita 360/:n]
fim
```

O procedimento do exemplo 2 desenha um polígono regular de :n lados de tamanho :t.

O Comando **para**

O segundo tipo de estrutura de controle de repetição do primeiro tipo é o comando **para**. Sua sintaxe é mostrada abaixo:

para (lista “<var> <vini> <vfin> <inc>”) [<lista de instruções>]

onde <var> é o nome de uma variável que será usada para controlar o número de repetições da seguinte maneira:

Caso 1 - <vini> é menor ou igual a <vfin>

1. Inicialmente a variável <var> recebe o valor calculado por <vini>.
2. Se o valor de <vini> for menor ou igual a <vfin> então a <lista de instruções> será executada. Caso contrário o comando será encerrado.
3. A variável <var> é incrementada do valor calculado em <inc>, que deve ser maior que zero. Se <inc> for omitida seu valor será igual a um.
4. Repita os passos de 2 a 4.

Caso 2 - <vini> é maior ou igual a <vfin>

1. Inicialmente a variável <var> recebe o valor calculado por <vini>.
2. Se o valor de <vini> for maior ou igual a <vfin> então a <lista de instruções> será executada. Caso contrário o comando será encerrado.
3. A variável <var> é decrementada do valor calculado em <inc> que deve ser menor que zero. Se <inc> for omitida seu valor será igual a menos um.
4. Repita os passos de 2 a 4.

EXEMPLO 1

Considere o procedimento abaixo que tem como parâmetros de entrada :vini, :vfin, e :inc e mostra a variação da variável de controle x.

```
aprenda contagem :a :b :c
  para (lista "x :a :b :c) [mostre :x]
fim
```

Vemos abaixo alguns resultados apresentados por ele:

contagem 4 9 1	contagem 4 9 2	contagem 4 9 menos 2
4	4	
5	6	
6	8	
7		
8		
9		

contagem 9 4 menos 1	contagem 9 4 menos 2	contagem 9 4 2
9	9	
8	7	
7	5	
6		
5		
4		

EXEMPLO 2 : O procedimento abaixo calcula a soma dos n primeiros termos de uma progressão aritmética dados n , o primeiro termo (t) e a razão (r).

```

aprenda somapa :n :t :r
  atribua "resp 0
  para (lista "x 1 :n) [atribua "resp :resp+:t atribua "t :t+:r]
  saída :resp
fim

```

EXEMPLO 3: O procedimento seguinte mostra os n primeiros múltiplos de um número a .

```

aprenda múltiplos :n :a
  para (lista "x :a :n*:a :a) [mostre :x]
fim

```

O Comando **enquanto**

O comando **enquanto** pertence ao grupo das estruturas de controle de repetição do segundo tipo, ou seja, daqueles cujo número de repetições depende de um teste. Sua sintaxe é mostrada abaixo:

```
enquanto [<expressão lógica>] [<lista de comandos>]
```

A <lista de comandos> será executada repetidamente *enquanto* o resultado da <expressão lógica> for verdadeiro.

EXEMPLO 1

O procedimento abaixo calcula o fatorial de n .

```

aprenda fatorial :n
  atribua "resp 1
  enquanto [:n>1] [atribua "resp :resp*:n atribua "n :n-1]
  saída :resp
fim

```

EXEMPLO 2

Euclides, o famoso geômetra grego descobriu um procedimento para encontrar o MDC (máximo divisor comum) de dois números inteiros.

ALGORITMO DE EUCLIDES para calcular o MDC (máximo divisor comum) de dois números.

INSTRUÇÃO 1 - Se os dois números são iguais, então pare pois o MDC é igual a eles próprios.

INSTRUÇÃO 2 - Subtraia o menor número do maior e descarte o maior.

INSTRUÇÃO 3 - Repita as instruções 1 e 2 usando como dois números o menor e o resultado da subtração da instrução 2.

EXEMPLOS: Vemos abaixo as etapas para o calculo do MDC de 360 e 144 e de 943 e 437.

360	144
216	144
72	144
72	72

943	437
506	437
69	437
69	368
69	299
69	230
69	161
69	92
69	23
46	23
23	23

O procedimento abaixo implementa o algoritmo de Euclides:

```

aprenda mdc :a :b
  enquanto [não :a=:b] [se :a>:b [atr "a :a-:b] [atr "b :b-:a]]
  saída :a
fim

```

EXEMPLO 3

O procedimento abaixo calcula a raiz cúbica de um número n com nc casas decimais:

```

aprenda raizcub :n :nc
  atr "r 1
  atr "inc 1
  repita :nc+1 [
    enquanto [não (potência :r 3) >:n] [atr "r :r+:inc]
    atr "r :r-:inc
    atr "inc :inc/10]
  saída :r
fim

```

11. RECURSIVIDADE

Uma *definição recursiva* é aquela em que o item que está sendo definido é usado na sua própria definição. A princípio, pode parecer muito estranho que algo possa ser definido em termos de si próprio. Mostraremos alguns exemplos de definições recursivas que encontramos na matemática e na computação para tornar mais claro o conceito de definição recursiva.

Seqüências definidas recursivamente.

Uma *seqüência* é uma lista de objetos dispostos segundo uma determinada ordem. Ou seja, numa seqüência existe um primeiro elemento, um segundo e assim por diante. Podemos definir recursivamente uma seqüência explicitando o valor do(s) primeiro(s) elemento(s) e definindo os demais elementos a partir deles. Por exemplo, a seqüência S que tem como seis primeiros elementos os números 2, 4, 8, 16, 32, 64 ... Percebemos que existe uma lei de formação recursiva, pois um elemento qualquer da seqüência é igual ao dobro do elemento anterior, com exceção do primeiro que não tem anterior. Podemos expressar essa definição da seguinte forma.

$$S(n) = \begin{cases} 2 & \text{para } n = 1 \\ 2 * S(n - 1) & \text{para } n \geq 2 \end{cases}$$

Por exemplo, para encontrar o quarto elemento da seqüência passaremos pelas etapas seguintes:

$$\begin{aligned} S(4) &= 2 * S(3) \\ S(4) &= 2 * 2 * S(2) \\ S(4) &= 2 * 2 * 2 * S(1) \\ S(4) &= 2 * 2 * 2 * 2 = 16 \end{aligned}$$

A função que retorna o enésimo elemento da seqüência é:

```
aprenda seqüência :n
se :n=1 [saída 2]
saída 2*seqüência :n-1
fim
```

A conhecida seqüência de Fibonacci, cujos primeiros termos são 1, 1, 2, 3, 5, 8, 13, 21 é definida recursivamente da seguinte maneira:

$$F(n) = \begin{cases} 1 & \text{para } n = 1 \\ 1 & \text{para } n = 2 \\ F(n - 2) + F(n - 1) & \text{para } n > 2 \end{cases}$$

Para encontramos o quarto termo da seqüência fazemos:

$$F(4) = F(2) + F(3) = 1 + F(1) + F(2) = 1 + 1 + 1 = 3$$

A função que retorna o n ésimo elemento da sequência de Fibonacci é mostrada abaixo:

```

aprenda fibo :n
se :n=1 | :n=2 [saída 1]
saída (fibo :n-2) + fibo :n-1
fim

```

Operações definidas recursivamente

Na matemática, uma operação é qualquer ação que transforma uma ou várias entidades matemáticas de um domínio em outra do mesmo domínio. Algumas operações podem ser definidas recursivamente, como por exemplo, a operação de potenciação a^n , mostrada abaixo:

$$a^n = \begin{cases} 1 & \text{para } n = 0 \\ (a^{n-1}) \times a & \text{para } n \geq 1 \end{cases}$$

onde a é um número real diferente de zero e n é um inteiro não negativo. A função que retorna a elevado a n é mostrada a seguir.

```

aprenda pot :a :n
se :n=0 [saída 1]
saída :a*pot :a :n-1
fim

```

Podemos, também, dar uma definição recursiva para a operação de multiplicação da seguinte maneira:

$$m \times n = \begin{cases} m & \text{para } n = 1 \\ m \times (n - 1) + m & \text{para } n \geq 2 \end{cases}$$

onde m e n são números inteiros positivos.

A função que retorna o resultado da multiplicação de m por n é:

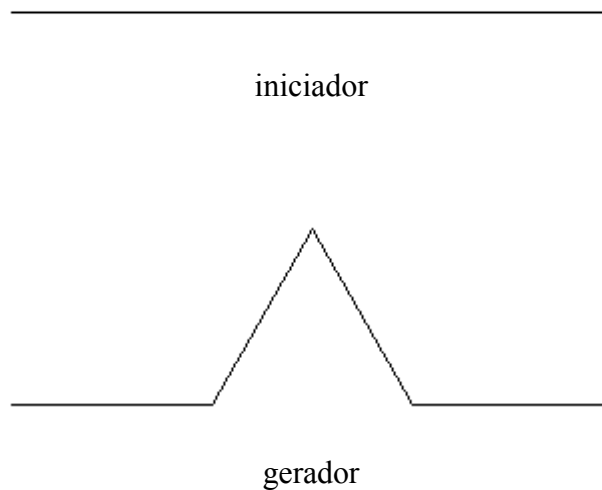
```

aprenda multiplica :m :n
se :n=0 [saída 0]
saída :m+multiplica :m :n-1
fim

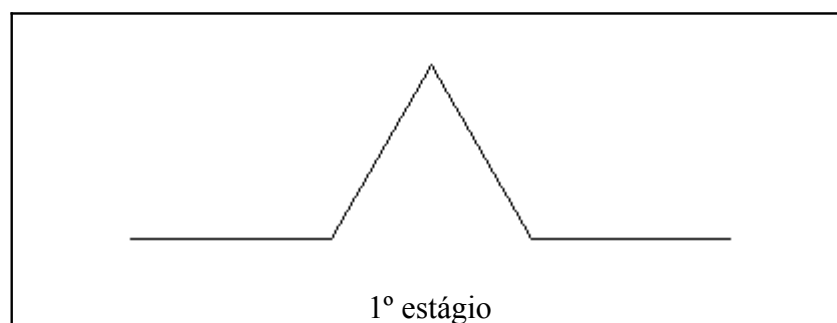
```

Figuras definidas recursivamente

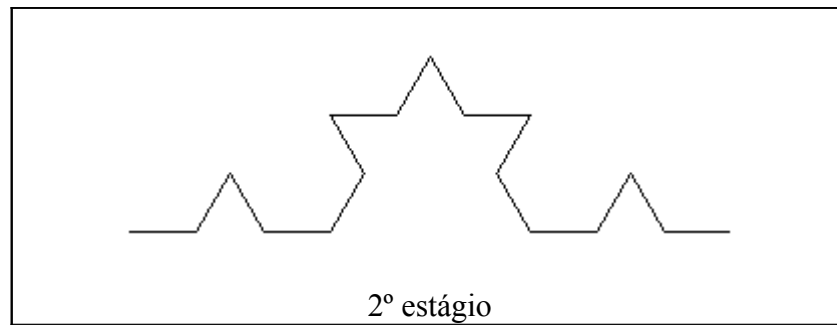
Algumas curvas da geometria podem ser mais facilmente definidas pela estratégia recursiva. O exemplo clássico é o da famosa curva de Koch descoberta em 1904. Ela é gerada a partir de duas formas que chamamos de *iniciador* e *gerador* mostradas abaixo:



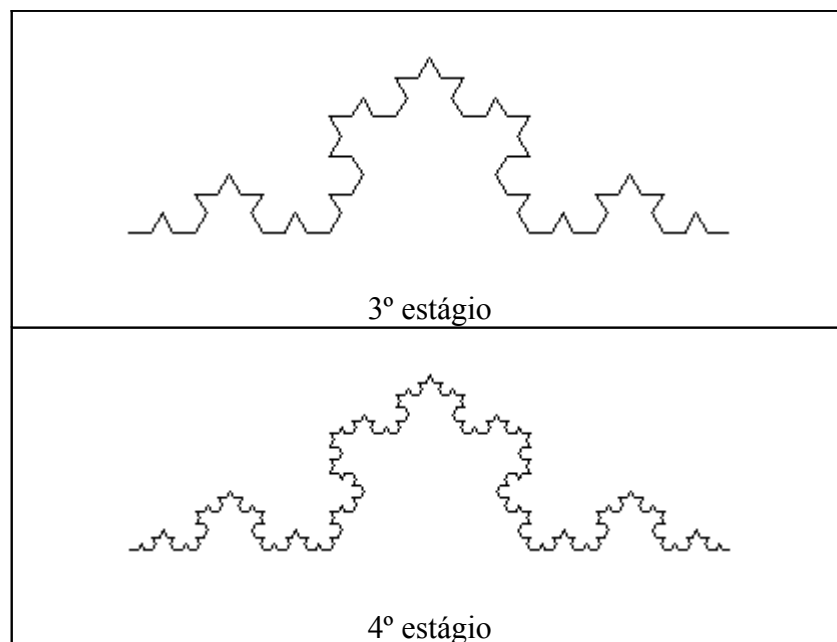
O processo de construção se inicia pela substituição de cada segmento de reta do iniciador, no caso apenas um, pelo gerador. Obtemos, após esta substituição, o primeiro estágio da construção da curva de Koch, mostrado na figura seguinte, que, nesse caso se assemelha ao gerador.



O segundo estágio é obtido pelo mesmo processo aplicado agora à curva obtida no 1º estágio. Ou seja, cada segmento de reta do estágio 1 é substituído por uma cópia do gerador na devida escala para que sejam mantidos os mesmos pontos inicial e final. A curva obtida no segundo estágio da geração da curva de Koch é mostrada abaixo.



Repetindo-se o mesmo processo indefinidamente geramos a curva de Koch. Nas figuras seguintes, mostramos os estágios 3 e 4 da geração da curva de Koch.



Para definir um procedimento que desenhe a curva de Koch podemos adotar a seguinte estratégia. Inicialmente definimos um procedimento que desenhe o gerador, como mostrado a seguir:

```

aprenda koch :t
  para frente :t/3
  para esquerda 60
  para frente :t/3
  para direita 120
  para frente :t/3
  para esquerda 60
  para frente :t/3
fim
  
```

Depois de testar o procedimento definido para verificar se ele realmente desenha o gerador substituiremos cada segmento de reta do nosso procedimento (pf :t/3) por uma chamada recursiva (koch :t/3 :n-1) tomando o cuidado de estabelecer a condição de parada do nosso procedimento, como podemos ver a seguir:

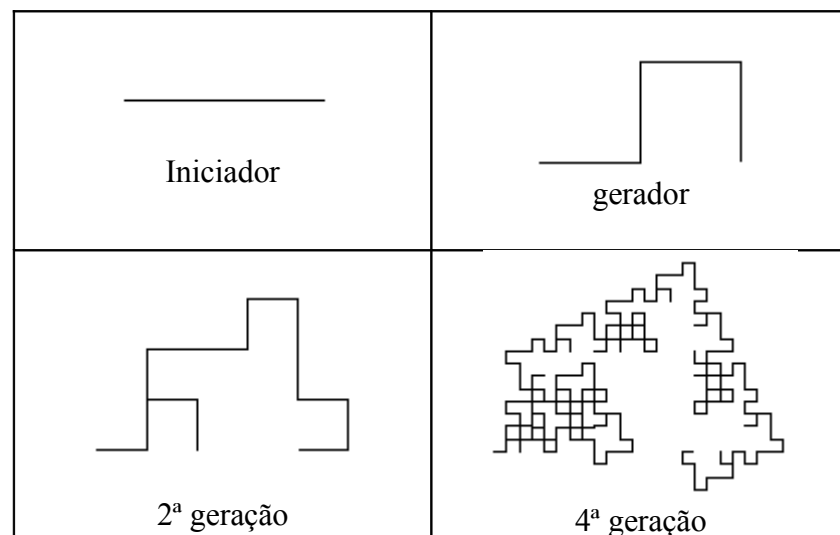
```

aprenda koch :t :n
se :n=0 [para frente :t pare]
koch :t/3 :n-1
paraesquerda 60
koch :t/3 :n-1
paradireita 120
koch :t/3 :n-1
paraesquerda 60
koch :t/3 :n-1
fim
    
```

A curva de Koch recebeu este nome por ter sido proposta, em 1904, pelo matemático sueco *Helge von Koch*. Na época, a curva de Koch foi rejeitada pelos matemáticos, pois durante séculos, eles consideravam como legítimas somente as curvas que possuísem tangentes bem definidas em cada ponto. Além disso, consideravam, platonicamente, que as coisas da natureza deveriam se aproximar desse tipo de curva suave, sem pontos de fratura. Mas, a curva que acabamos de construir tem infinitos pontos de fratura. Por isso, a curva de Koch foi chamada de “monstro matemático” e de “curva patológica”. Mandelbrot foi o primeiro a questionar esta posição platônica, afirmando que objetos matemáticos complexos, como a curva de Koch, que ele chamou de curvas fractais, são a regra e não a exceção na matemática e na natureza.

Outros exemplos de figuras geradas como a curva de Koch

EXEMPLO 1.

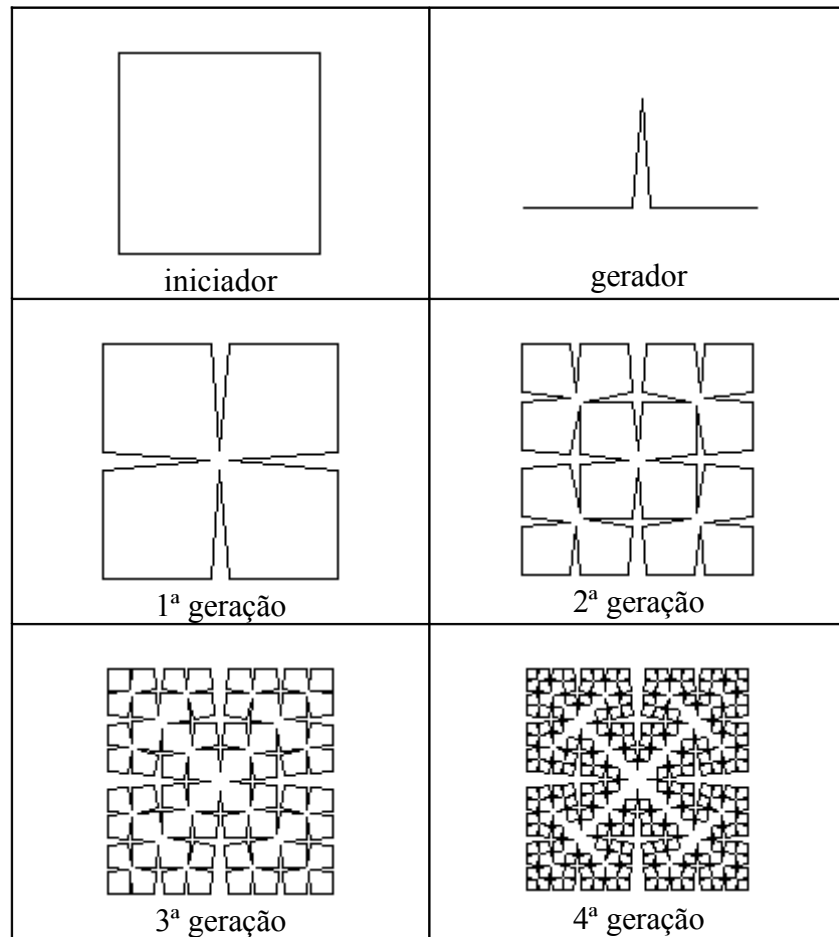


Vemos abaixo o procedimento que desenha a figura de koch do exemplo 1.

```

aprenda koch2 :t :n
  se :n=0 [pf :t pare]
  koch2 :t/2 :n-1
  paraesquerda 90
  koch2 :t/2 :n-1
  paradireita 90
  koch2 :t/2 :n-1
  paradireita 90
  koch2 :t/2 :n-1
  paraesquerda 90
fim
  
```

EXEMPLO 2



Vemos abaixo os procedimentos que geram o iniciador e o gerador da figura de Koch do exemplo 2:

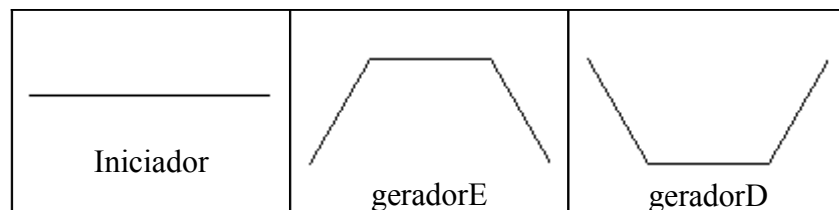
```

aprenda cesaro :t :n
  repita 4 [cesaro2 :t :n paraesquerda 90]
fim

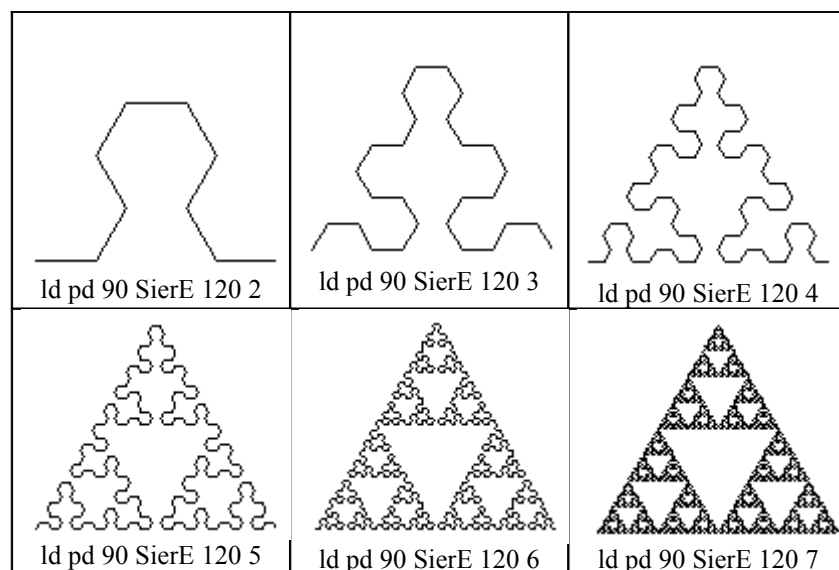
aprenda cesaro2 :t :n
  se :n=0 [pf :t pare]
  cesaro2 0.45*:t :n-1
  paraesquerda 85
  cesaro2 0.45*:t :n-1
  paradireita 170
  cesaro2 0.45*:t :n-1
  paraesquerda 85
  cesaro2 0.45*:t :n-1
fim
  
```

EXEMPLO 3 - RECURSÃO MÚTUA

Temos recursão mútua quando dois procedimentos se chamam recursivamente, ou seja, o primeiro procedimento chama o segundo que por sua vez faz referência ao primeiro chamando-o também. Podemos gerar figuras de Koch que tenham dois geradores que se chamam recursivamente. Por exemplo:



Vemos abaixo algumas gerações desse exemplo:



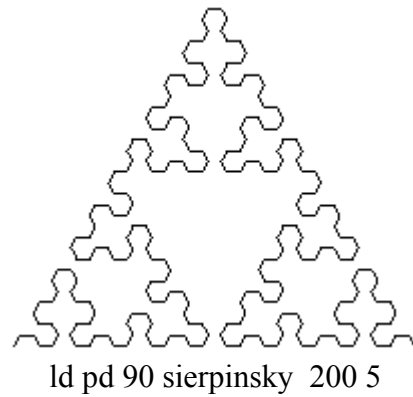
Essa figura gerada pelos mesmos princípios da curva de Koch recebe o nome de triângulo de Sierpinsky em homenagem ao seu criador.

Vemos abaixo a listagem dos dois geradores dessa figura onde podemos observar a existência da recursão mútua.

```

aprenda gerae :t :n
  se :n=0 [pf :t pare]
  paraesquerda 60
  geraD :t/2 :n-1
  paradireita 60
  geraE :t/2 :n-1
  paradireita 60
  geraD :t/2 :n-1
  paraesquerda 60
fim

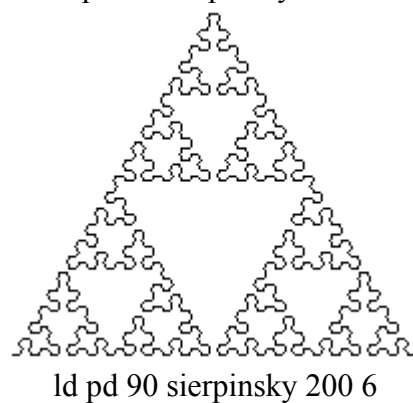
```



```

aprenda gerad :t :n
  se :n=0 [pf :t pare]
  paradireita 60
  geraE :t/2 :n-1
  paraesquerda 60
  geraD :t/2 :n-1
  paraesquerda 60
  geraE :t/2 :n-1
  paradireita 60
fim

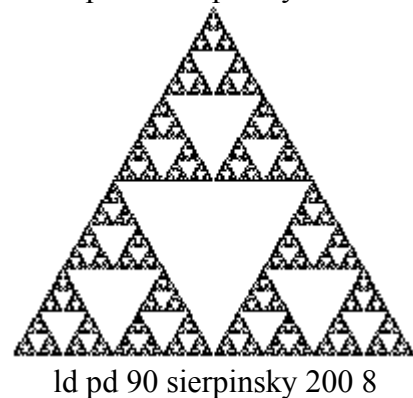
```



```

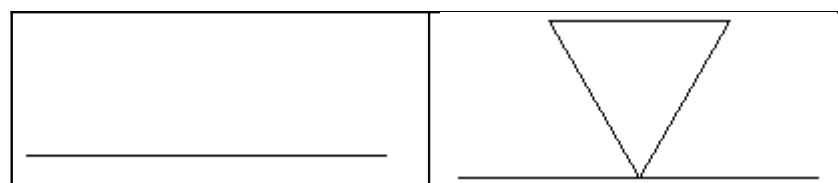
aprenda sierpinsky :t :n
  geraE :t :n
fim

```

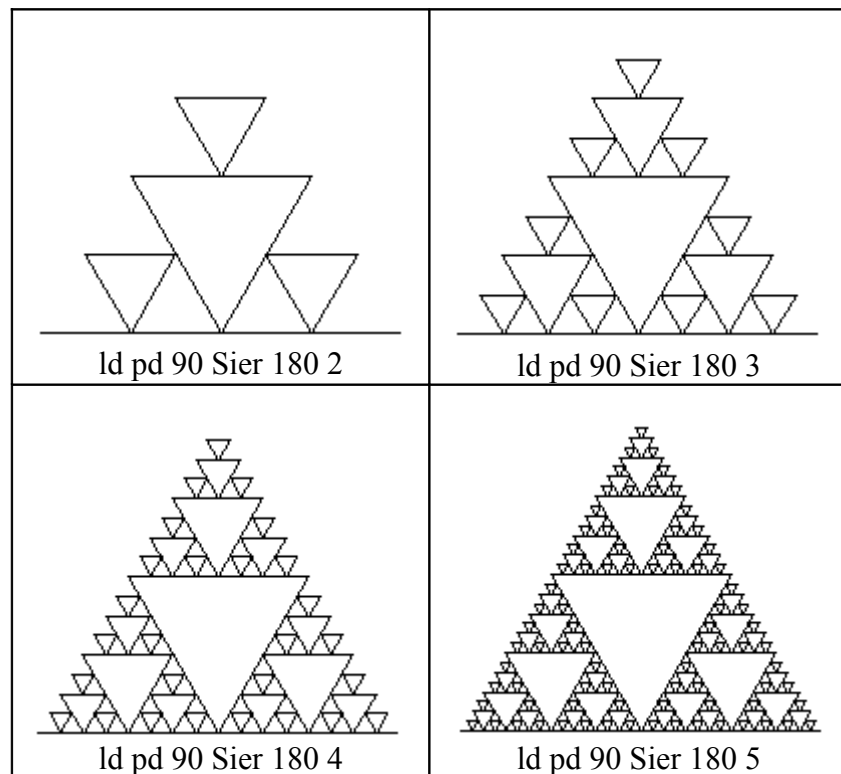


EXEMPLO 4

Podemos construir o triângulo de Sierpinsky de outra maneira usando o iniciador e o gerador mostrados abaixo:



Vemos a seguir várias gerações da figura



Vemos a seguir o procedimento que gera as figuras acima:


```

aprenda sier :t :n
se :n=0 [para frente :t pare]
Sier :t/2 :n-1
paraesquerda 120
para frente :t/2
paradireita 120
Sier :t/2 :n-1
paradireita 120
para frente :t/2
paraesquerda 120
Sier :t/2 :n-1
fim
    
```

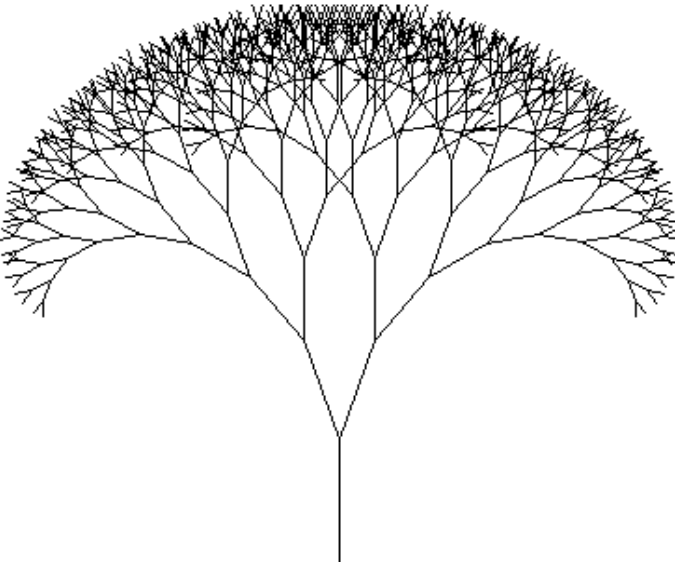
Podemos, então, vislumbrar que existem infinitas possibilidades de geração de figuras quando nos utilizamos da definição recursiva. Tentemos pois criar as nossas próprias figuras.

EXEMPLO 5 – ÁRVORES

Podemos gerar figuras em forma de árvore mais facilmente se usarmos definições recursivas e a idéia de reescrita de termos. Imaginemos pois que temos como iniciador uma bifurcação como a mostrada abaixo que pode ser gerada pelo procedimento ao lado..

	<pre> aprenda árvore :t para frente :t para esquerda 20 para frente 0.8*:t para trás 0.8*:t para direita 40 para frente 0.8*:t para trás 0.8*:t para esquerda 20 para trás :t fim </pre>
---	--

A idéia é substituir cada um dos dois ramos por uma cópia da própria bifurcação recursivamente. Podemos obter o resultado esperado substituindo os comandos pf 0.8*:t e pt 0.8*:t, que são os ramos, por chamadas recursivas ao procedimento. Ver procedimento árvore abaixo.

<pre> aprenda árvore :t :n se :n=0 [pare] para frente :t para esquerda 20 árvore 0.8*:t :n-1 para direita 40 árvore 0.8*:t :n-1 para esquerda 20 para trás :t fim </pre>	
--	--

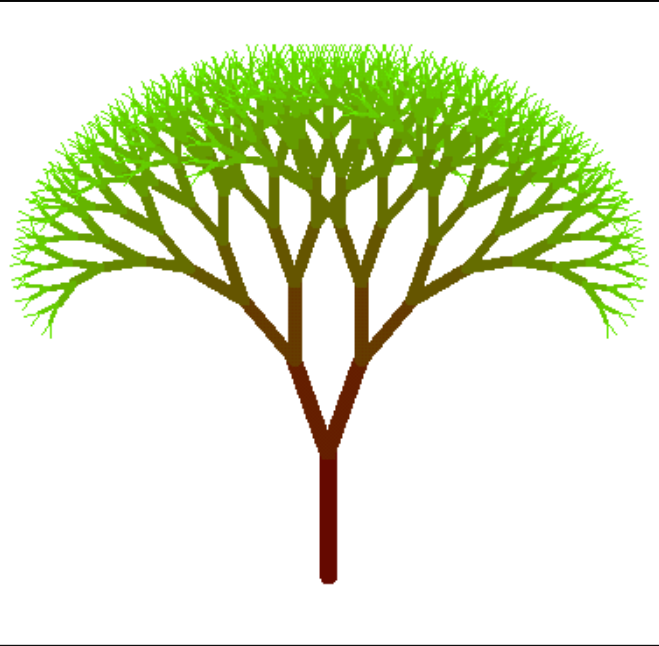
Vemos na figura acima a forma arbórea obtida pelo procedimento árvore

Acrescentando algumas linhas ao código podemos dar à nossa árvore uma aparência mais natural através da espessura e cor diferenciada dos ramos. Ver figura e o código que a gerou:

```

aprenda árvore :t :n
  atr "m inteiro 255/:n
  ramo :t :n
  fim

aprenda ramo :t :n
  se :n=0 [pare]
  mudeespessuradolápis lista :n :n
  mudecl (lista 100 255-:n*:m 0)
  pf :t
  pe 20
  ramo :t*0.8 :n-1
  pd 40
  ramo :t*0.8 :n-1
  pe 20
  un pt :t ul
  fim
  
```

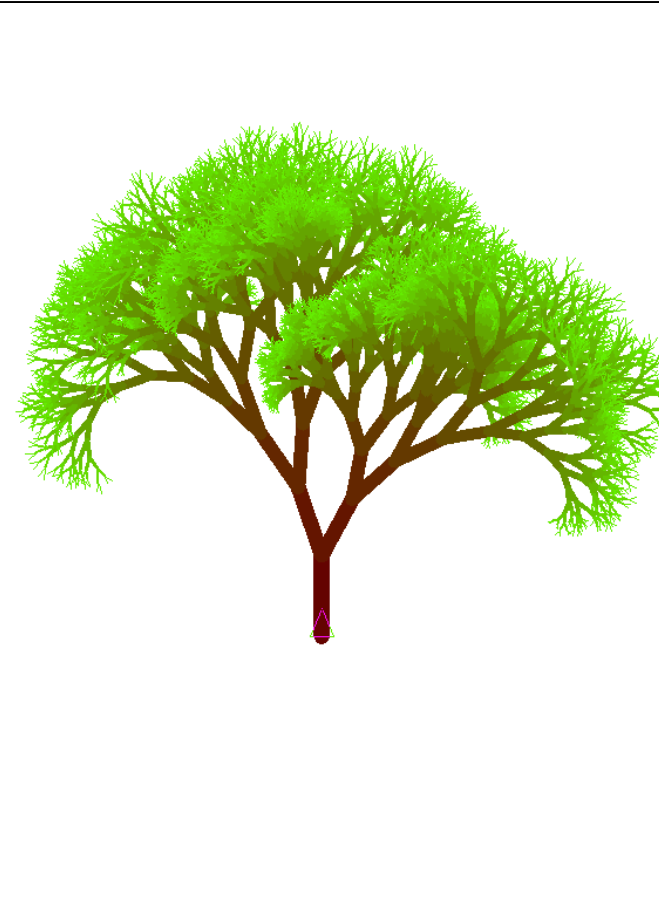


Podemos melhora ainda mais a aparência de nossa árvore se introduzirmos um pouco de aleatoriedade nos ângulos e tamanhos dos galhos. Ver procedimentos abaixo:

```

aprenda árvorenatural :t :n
  atr "m inteiro 255/:n
  ramonat :t :n
  fim

aprenda ramonat :t :n
  local "ae
  local "ad
  local "tam
  se :n=0 [pare]
  atribua "ae 10+sorteie 21
  atribua "ad 10+sorteie 21
  atribua "tam (85+sorteie
  31)/100*:t
  mudeespessuradolápis :n
  mudecl (lista 100 255-:n*:m 0)
  pf :tam
  pe :ae
  ramonat :tam*0.8 :n-1
  pd :ae+:ad
  ramonat :tam*0.8 :n-1
  pe :ad
  un pt :tam ul
  fim
  
```



12. ESTRUTURAS DE DADOS

Até agora trabalhamos com dados simples, ou seja, cada variável armazenava um único valor. Muitas vezes precisamos armazenar dados que possuam múltiplos valores como, por exemplo, uma fração que possui um numerador e um denominador. Para armazenar uma fração em uma variável é preciso que esta variável possua uma estrutura especial que permita que múltiplos valores sejam armazenados com um mesmo nome. Essa técnica chama-se estruturação de dados.

As informações armazenadas em um meio qualquer podem ser estruturadas ou organizadas de várias maneiras diferentes dependendo da aplicação que lhes daremos. As principais estruturas de dados são:

1. Listas
2. Palavras
3. Filas
4. Pilhas
5. Vetores e Matrizes
6. Registros
7. Classes

Listas e Palavras

A linguagem Logo, por ser um dialeto da linguagem LISP, possui naturalmente um conjunto completo de comandos para realizar operações sobre o primeiro e mais básico tipo de estrutura de dados: as listas.

Uma *lista* é uma coleção $L: [a_1, a_2, \dots, a_n]$, $n \geq 0$, cuja propriedade estrutural baseia-se apenas na posição relativa dos elementos, que são dispostos linearmente. Se $n=0$, dizemos que a lista é vazia; caso contrário, são válidas as seguintes propriedades:

- a_1 é o primeiro elemento de L ;
- a_n é o último elemento de L ;
- a_k , $1 < k < n$, é precedido pelo elemento a_{k-1} e seguido por a_{k+1} em L .

Os elementos de uma lista podem ser números, palavras ou até mesmo listas.

Uma palavra é uma seqüência de caracteres. Na linguagem *Logo* uma palavra é identificada por possuir aspas duplas em seu início. Esta convenção torna possível diferenciar uma palavra de um nome de procedimento. Podemos armazenar uma palavra “Escola na memória do computador (na variável NomeCliente) através do comando abaixo:

atribua “NomeCliente “Ricardo

O Caracter Especial "\"

O caracter «\» (barra invertida ou "backslash") permite em particular colocar espaço entre palavras ou ainda uma nova linha. «\n» faz a quebra de linha enquanto que «\» seguido de um espaço indica espaço numa palavra.

Exemplo:

```
mostre "Ricardo\ Guedes  
Ricardo Guedes
```

```
mostre "Ricardo\nGuedes  
Ricardo  
Guedes
```

Se desejar escrever o caracter «\» será necessário escrevê-lo duas vezes («\\»).

```
mostre "\\xlogo  
\xlogo  
mostre "\\  
\
```

Da mesma forma, os caracteres «() [] # » são reservados para a linguagem Logo, sendo assim, se necessitar representá-los bastará colocar um caractere «\» antes.

```
mostre "(xlogo\  
(xlogo)
```

Todos os caracteres «\» são ignorados. Este aspecto é especialmente importante em particular para o gerenciamento de arquivos

Exemplo no Windows: Para definir (mudar) o diretório atual para C:\Meus Documentos, escreva:

```
mudediretório "c:\\Meus\ Documentos
```

Exemplo no Linux: Para definir (mudar) o diretório atual para /local/aluno/Meus Documentos, escreva:

```
mudediretório "/local/aluno/Meus\ Documentos
```

Note o uso do «\» para indicar o espaço entre «Meus » e «Documentos ». No exemplo do Windows, se você omitisse a dupla barra invertida, o caminho seria definido como c:Meus Documentos e o xLogo devolveria uma mensagem de erro (diretório inválido).

Criação de Listas

Podemos criar listas de três maneiras:

Usando a colchetes como delimitadores

A primeira, atribuindo a uma variável os elementos da lista colocados entre colchetes. Cada elemento é separado do outro por um espaço.

Exemplo 1:

```
atribua "lista [1 5 3 8 6]
```

Nesse exemplo é criada uma lista com cinco elementos numéricos.

Exemplo 2:

```
atribua "nome [Centro Federal de Educação Tecnológica]
mostre :nome
Centro Federal de Educação Tecnológica
```

A variável "nome conterà um lista composta de cinco palavras que podem ser exibidas pelo comando `mostre`.

Usando a palavra `lista`

A segunda maneira de criarmos uma lista é através do comando **lista** cuja sintaxe mostramos abaixo:

lista

Sintaxe : lista objeto1 objeto2
 (lista objeto1 objeto2...objeto n)

Descrição:

O comando `lista` retorna uma lista constituída por `objeto1` e `objeto2`. Um objeto pode ser um número, uma palavra ou até mesmo uma lista. Se houver mais de dois parâmetros de entrada devemos usar parênteses.

Exemplos:

```
>mostre lista "a "b
a b
```

```
>atribua "letras [a b]
>atribua "letras lista :letras "c
>mostre :letras
[a b] c
```

```
>mostre (lista "L "O "G "O)
L O G O
```

```
>mostre lista [1 2 3] [a b c]
[1 2 3][a b c]
```

Usando a palavra `senteça`

A terceira maneira de criarmos uma lista é através do comando **sentença** cuja sintaxe mostramos abaixo:

sentença

Sintaxe : sentença objeto1 objeto2
 sn objeto1 objeto2

Descrição :

Retorna uma lista formada pela concatenação de objeto1 e objeto2.

Observação :

Se houver mais de dois parâmetros de entrada deve-se usar parênteses.

Exemplo :

```
>mostre sn "cachorro "quente
cachorro quente
>mostre (sentença 1 2 3)
1 2 3
```

Existe uma sutil diferença entre os comandos **lista** e **sentença** que percebemos através dos exemplos abaixo:

```
>atribua "frase (sn [criar] "com "LOGO)
>esc :frase
criar com LOGO
```

```
>atribua "frase (lista [criar] "com "LOGO)
>esc :frase
[criar] com LOGO
```

Operações sobre Listas e Palavras

Sendo um dialeto da linguagem LISP a linguagem Logo herdou um conjunto completo de operadores que atuam sobre as listas. Descreveremos cada um destes operadores:

OPERADORES DE INSERÇÃO

junteno início

Sintaxe : junteno início objeto lista
 ji objeto lista

Descrição :

Retorna a lista do parâmetro de entrada acrescida do objeto no seu início.

Exemplo :

```
>esc junteno início "l [o g o]
l o g o
>mostre ji 1 [2 3 4]
[1 2 3 4]
```

juntanofim

Sintaxe : juntanofim objeto lista
 jf objeto lista

Descrição :

Retorna a lista do parâmetro de entrada acrescida do objeto no seu final.

Exemplo :

```
>atribua "vogais juntanofim "u [a e i o]
>esc :vogais
a e i o u
>mostre jf "5 [1 2 3 4]
[1 2 3 4 5]
```

palavra

Sintaxe: palavra palavra1 palavra2
 pal palavra1 palavra2

Descrição

Retorna uma palavra composta pela concatenação de palavra1 e palavra2.

Obs. se houver mais de dois parâmetros de entrada, deve-se usar parênteses.

Exemplos:

```
>esc palavra "LO "GO
LOGO
>esc (pal "a "pren "der)
aprender
```

OPERADORES DE CONSULTA

primeiro

Sintaxe : primeiro objeto
 pri objeto

Descrição :

Retorna o primeiro elemento de objeto. Se objeto for uma palavra, retorna o primeiro caracter da palavra. Se objeto for uma lista, retorna o primeiro elemento da lista.

Exemplos :

```
>escreva primeiro [a b c d]
a
>esc pri "LOGO
L
>escreva pri [[L][O][G][O]]
L
```

último

Sintaxe : último objeto
 ult objeto

Descrição :

Retorna o último elemento do objeto. Se objeto for uma palavra, retorna o último

caracter da palavra. Se objeto for uma lista, retorna o último elemento da lista.

Exemplo :

```
>esc último [1 2 3]
3
>esc ult "aqui
i
```

OPERADORES DE EXCLUSÃO

semprimeiro

Sintaxe : semprimeiro objeto
 sp objeto

Descrição :

Retorna objeto sem seu primeiro elemento.

Exemplo :

```
>esc sp "LOGO
OGO
>atribua "x [1 2 3]
>esc sp :x
2 3
```

semúltimo

Sintaxe : semúltimo objeto
 su objeto

Descrição :

Retorna objeto sem seu último elemento.

Exemplo :

```
>esc su "LOGO
LOG
>coloque [1 2 3] "x
>esc su :x
1 2
```

EXEMPLOS DE PROCEDIMENTOS

Vemos a seguir duas versões (iterativa e recursiva) de um procedimento que envia como resposta o número de elementos de uma lista.

```
aprenda contalista :l
  atribua "resp 0
  enquanto [não :l=[]) [
    atribua "l sp :l
    atr "resp :resp+1]
  saída :resp
fim
```

```
aprenda contalista2 :lis
```



```
se :lis=[] [saída 0]
saída 1+contalista2 sp :lis
fim
```

A linguagem logo já possui uma palavra primitiva com a mesma função.

conte

Sintaxe: conte objeto

Descrição:

retorna o número de elementos que compõem objeto. Se objeto for uma palavra, retorna o número de caracteres da palavra. Se objeto for uma lista retorna o número de elementos da lista.

Exemplo:

```
>esc conte [renato e alice]
3
>esc conte "flávio
6
```

EXERCÍCIO:

Elabore um procedimento que envie como resposta a combinação de duas listas. Por exemplo:

```
mostre comb [3 6 2] [8 5]
[3 6 2 8 5]
```

Outros Operadores

elemento

Sintaxe : elemento <número> <objeto>
 elem <número> <objeto>

Descrição:

Retorna o elemento de posição número no objeto. Se objeto for uma palavra, retorna caracter que está na posição número da palavra. Se objeto for uma lista, retorna o elemento que está na posição número da lista.

Exemplos :

```
>escreva elemento 4 "praia
i
>esc elemento 3 [rosa cravo violeta]
violeta
>atr "x [azul verde]
>esc elemento 2 :x
verde
```

émembro?

Sintaxe: émembro? objeto1 objeto2
 emembro? objeto1 objeto2

Descrição:

se objeto2 for uma lista ou retorna a palavra *verd* se objeto1 for igual a um membro ou a um elemento de objeto2; caso contrário, retorna a palavra *falso*. Se

objeto2 for uma palavra, retorna verd se o caractere objeto1 estiver contido em objeto2; caso contrário, retorna falso.

Exemplos:

```
> mostre émembro? 1 [1 2 3]
verd
> mostre éelem "a [a b c]
verd
> mostre éelemento "x [ e r t]
falso
> mostre émembro? "F "CEFET
verd
```

escolhe

Sintaxe : escolhe objeto

Descrição :

Retorna aleatoriamente um elemento de um parâmetro dado, o qual pode ser uma lista ou uma palavra.

Exemplo :

```
> mostre escolhe [1 2 3 4 5 6 7 8 9]
2
> mostre escolhe [1 2 3 4 5 6 7 8 9]
6
> mostre escolhe "escola
o
```

inverte

Sintaxe: inverte lista

Descrição :

Retorna uma lista invertida ou seja, uma lista com os mesmos elementos da lista de entrada, porém em ordem invertida.

Exemplo :

```
> mostre inverte[1 2 3]
[3 2 1]
> mostre inverte [R O M A]
A M O R
```

EXERCÍCIOS

1. Defina um procedimento que envie como resposta uma lista sem os seus primeiros n elementos tendo como parâmetros de entrada a lista original e o número de elementos a serem excluídos no início da lista.
2. Defina um procedimento que envie como resposta uma lista sem os seus últimos n elementos tendo como parâmetros de entrada a lista original e o número de elementos a serem excluídos no fim da lista.

3. Defina um procedimento que insira um elemento *ele* na posição *pos* de uma lista *lis* e retorne a nova lista tendo como parâmetros de entrada a lista *lis*, o elemento *ele* e posição *pos* do elemento
4. Defina um procedimento que exclua um elemento de uma lista tendo como parâmetros de entrada a própria lista e o índice do elemento a ser excluído e enviando como resposta a nova lista.
5. Defina um procedimento que substitua um elemento de uma lista por outro elemento tendo como parâmetros de entrada a lista, o índice do elemento a ser substituído e o valor do novo elemento.
6. Defina um procedimento que receba como parâmetros de entrada uma lista numérica e retorne a posição do menor elemento.
7. Defina um procedimento que receba como parâmetro de entrada uma lista e retorne uma lista com mesmos elementos da lista original ordenados em ordem crescente.