

ACADEMIA DA TARTARUGA

Programação em Logo

PLATAFORMA ONLINE
Web



JOSÉ AUGUSTO N. G. MANZANO

Augusto N. G. Manzano



ORCID: 0000-0001-9248-7765

Academia da Tartaruga

Programação em Logo



São Paulo
2021 - Propes Vivens

ISBN 978-65-00-26398-5

9 786500 263985

© Copyright 2021 by José Augusto N. G. Manzano / Propes Vivens.

Todos os direitos reservados. Proibida a reprodução total ou parcial, por qualquer meio ou processo, especialmente por sistemas gráficos, microfilmicos, fotográficos, reprogramáticos, fonográficos, videográficos, internet, e-books. Vedada a memorização e/ou recuperação total ou parcial em qualquer sistema de processamento de dados e a inclusão de qualquer parte da obra em qualquer programa jus cibérnético existentes ou que a venham existir. Essas proibições aplicam-se também às características gráficas da obra e à sua editoração, exceto pelo exporto no próximo parágrafo. A violação dos direitos autorais é punível como crime (art. 184 e parágrafos, do Código Penal, conforme Lei nº 10.695, de 07.01.2003) com pena de reclusão, de dois a quatro anos, e multa, conjuntamente com busca e apreensão e indenizações diversas (artigos 102 e 103 parágrafo único, 104, 105, 106 e 107 itens 1, 2 e 3 da Lei nº 9.610, de 19.06.1998, Lei dos Direitos Autorais).

Esta obra é distribuída gratuitamente em formato digital (somente em PDF) apenas e tão somente no sítio do autor (www.manzano.pro.br) e na forma impressa comercialmente e disponibilizada nas plataformas **Clube de Autores** e **Agbook**. Nenhum outro local da Internet ou fora dela está autorizado a distribuir, seja gratuitamente ou comercialmente este material. Não é permitido o compartilhamento deste material em qualquer lugar ou por qualquer meio exceto o exposto neste parágrafo, bem como outro formato digital. Os infratores estão sujeitos a processo judicial.

O Autor acredita que as informações aqui apresentadas estão corretas e podem ser utilizadas para qualquer fim legal. Entretanto, não existe qualquer garantia, explícita ou implícita, de que o uso de tais procedimentos conduzirá sempre ao resultado desejado. O autor e a editora não poderão ser responsabilizados civilmente ou criminalmente. Os nomes de sítios e empresas, mencionados, foram utilizados apenas como ilustração, não havendo nenhum vínculo com a obra, não garantindo a sua existência nem divulgação a posteriori.

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Manzano, Augusto N.G. Academia da tartaruga [livro eletrônico] : programação em logo / Augusto N.G. Manzano. -- 1. ed. -- São Paulo : Ed. do Autor, 2021. PDF ISBN 978-65-00-26398-5 1. Algoritmos de computadores 2. Ciência da computação - Estudo e ensino 3. Programação (Computadores eletrônicos) I. Título.	21-72157	CDD-005.1
--	----------	-----------

Índices para catálogo sistemático:

1. Programação de computadores 005.1

Aline Graziele Benitez - Bibliotecária - CRB-1/3129

Produção e Editoração: José Augusto Navarro Garcia Manzano
Capa: canva.com / Espiral hexagonal (veja apêndice)

Edição: Propes Vivens

FERRAMENTA UTILIZADA

Produto: **Academia da Tartaruga**

Sítio: https://turtleacademy.com/?lang=pt_BR

REQUISITOS DE HARDWARE E SOFTWARE

- ◆ Qualquer Sistema Operacional;
- ◆ Monitor com 1024 x 768 pontos ou superior para melhor visualização;
- ◆ Mouse ou outro periférico de apontamento;
- ◆ Acesso à Internet.

CARTA AO ESTUDANTE

Olá, estudante de programação.

Espero que você esteja bem e com excelente animo para desenvolver habilidades na arte da programação de computadores a partir de uma linguagem de programação que segue o estilo declarativo de definição. Este livro vem de encontro a sua aprendizagem fornecendo diversos aspectos práticos no uso da linguagem Logo como suporte as aulas de lógica de programação, podendo ser usado por pessoas de todas as idades.

A linguagem Logo é uma ferramenta que proporciona a aprendizagem de detalhes lógicos de forma divertida e descontraída partindo-se de um ponto aparentemente inocente e permitindo ao estudante de programação desenvolver aplicações robustas e complexas. Neste livro se faz uma introdução a linguagem e seus principais recursos de operação desde o uso da geometria da tartaruga até o desenvolvimento de programas independentes da operação gráfica.

O livro encontra-se dividido em cinco capítulos, mais alguns apêndices complementares que abordam diversas características de uso da linguagem, onde alguns capítulos possuem como reforço um conjunto de exercícios de fixação. São fornecidas orientações desde a obtenção e instalação da fermenta de trabalho *FMSLogo* a apresentação de diversas ações que podem ser produzidas com a linguagem, como: funções; operadores aritméticos, lógicos e relacionais; ações de entrada e saída; decisões; recursividades; escopo e visibilidade de variáveis; uso de procedimentos, entre outros. São demonstradas ações que produzem formas geométricas a partir de diversas primitivas (comandos da linguagem) e outras operações.

Cabe destacar que este livro é distribuído gratuitamente em formato eletrônico PDF no site do autor ou disponibilizado comercialmente em sua forma impressa a partir das plataformas de publicação Clube de Autores e Agbook para quem desejar ter o material no formato livro impresso.

Este trabalho não passou por revisões de língua portuguesa e está em sua forma bruta, versão pré-alpha. Por isso, poderão ser encontrados erros de escrita, concordância ou outros que passaram “batidos”, exceto os códigos de programas apresentados os quais foram exaustivamente testados. Auxílios, neste sentido, são sempre bem vindos, desde que sejam realizados sem nenhum interesse financeiro ou comercial.

Espero que este conteúdo possa lhe ser bastante útil.

Com grande abraço.
Augusto Manzano.

AGRADECIMENTOS

À minha esposa Sandra e à minha filha Audrey, motivos de constante inspiração ao meu trabalho. Pela paciência nos momentos de ausência que tenho, quando estou absorto em escrever, dedicando-me ao ensino.

Há você que me lê neste livro e a todos os estudantes que passaram e passam por minhas mãos, que acreditaram e acreditam na minha pessoa e seguiram e seguem as orientações passadas; por me incentivarem continuamente quando me questionam sobre temas que ainda não conheço, por me levarem a um patamar maior, por exigirem assim que eu pesquise mais e retorno a você conhecimentos na forma de livros.

Vida longa e próspera.

SUMÁRIO

1.	INTRODUÇÃO	
1.1.	Linguagem Logo.....	xx
1.2.	Academia da Tartaruga	xx
1.3.	O ambiente operacional	xx
2.	AÇÕES BÁSICAS	
2.1.	Primitivas iniciais	xx
2.2.	Outras interações	xx
2.3.	Primitivas complementares	xx
3.	AÇÕES ESPECIALIZADAS	
3.1.	Repetições	xx
3.2.	Procedimentos	xx
3.3.	Sub-rotinas	xx
3.4.	Variável	xx
3.5.	Decisão	xx
3.6.	Recursão	xx
3.7.	Exercícios de fixação	xx
4.	AÇÕES ESPECÍFICAS	
4.1.	Entrada de dados	xx
4.2.	Randomização	xx
4.3.	Coordenadas	xx
4.4.	Cores	xx
4.5.	Fractais	xx
4.6.	Outras repetições	xx
4.7.	Exercícios de fixação	xx
5.	AÇÕES COMPLEMENTARES	
5.1.	Funções matemáticas	xx
5.2.	Funções lógicas	xx
5.3.	Algumas funções complementares	xx
5.4.	Programação sem figuras	xx
5.5.	Manipulação básica de dados	xx
5.6.	Escopo e visibilidade de variáveis	xx
5.7.	Exercícios de fixação	xx
	APÊNDICES	
A.	Exemplos geométricos	xx
B.	Espiral hexagonal (imagem da capa)	xx
C.	Gabarito	xx
	REFERÊNCIAS BIBLIOGRÁFICAS	xx

CAPÍTULO 1 - Introdução

A programação declarativa caracteriza-se por ser um paradigma de programação onde se diz a um computador o que deve ser feito e não como ser feito como ocorre no paradigma imperativo. Neste sentido, este capítulo apresenta a linguagem de programação Logo baseada na linguagem Lisp que é o primeiro exemplar de uma linguagem de programação declarativa.

1.1 - Linguagem Logo

A linguagem Logo foi desenvolvida durante a década de 1960, por uma equipe multidisciplinar dirigida pelo Filósofo, Matemático, Pesquisador e Professor Seymour Papert, com coautoria de Cynthia Solomon no *Massachusetts Institute of Technology* (MIT) com a participação direta do Professor Marvin Minsky e participação indireta de Daniel Bobrow e Wally Feurzeig.

Em 1960 o Professor Papert conhece na Universidade de Sorbonne (França) Jean Piaget e inicia com este um trabalho relacionado a teoria de aprendizagem. Deste momento histórico veio a influência para seus estudos nas áreas de inteligência artificial e robótica educacional, dando origem a linguagem Logo (LOGO FOUNDATION, 2015).

Em 1961 em uma conferência na Inglaterra o Professor Seymour Papert conhece o Professor Marvin Minsky do MIT um dos grandes expoentes da Inteligência Artificial (IA). Nessa ocasião eles apresentaram artigos muito semelhantes sobre o estudo de IA e o uso dessa tecnologia por crianças. Isso os aproxima e leva em 1964 o Professor Papert a integrar o Grupo de Inteligência Artificial do MIT. No MIT Papert conhece Daniel Bobrow, ex-aluno do Professor Minsky que foi trabalhar em uma empresa de pesquisa e desenvolvimento chamada *Bolt, Beranek and Newman* (BBN). Na BBN Daniel Bobrow conhece Wally Feurzeig e põe Feurzeig e Papert em contato. Nesta ocasião Bobrow, Papert e Feurzeig conversam sobre a linguagem que Papert deseja criar para crianças chamada *Mathland*, que posteriormente tornou-se *Logo* como um dialeto da linguagem Lisp (PALEOTRONIC, 2021).

Logo é uma linguagem declarativa com toques de programação imperativa. É fundamentada sobre os princípios da programação lógica e funcional tendo sido direcionada inicialmente a crianças. Quando a linguagem surgiu não existiam microcomputadores e o acesso a essa tecnologia era extremamente restrito. No entanto, após o ano de 1975 com o surgimento dos microcomputadores e o barateamento da tecnologia computacional o uso da linguagem se tornou mais popular. O ambiente de trabalho é baseado principalmente sobre uma interface plana com um ícone (cursor central) chamado tartaruga que tem por objetivo percorrer o plano desenhando imagens baseadas em figuras geométricas a partir de quatro comandos principais, suas primitivas de operação: **PARAFRENTE**, **PARATRÁS**, **PARADIREITA**, **PARAESQUERDA**. Além desses recursos existem outros que complementam a linguagem, tais como: **REPITA**, **SE**, **APRENDA**, **USELÁPIS**, **USEBORRACHA**, entre outros.

Apesar da linguagem ser muito conhecida devido ao modo de operação chamado *geometria da tartaruga*, ela é mais completa do que isso possuindo outros recursos. O efeito do modo *geometria da tartaruga* é apenas uma parte do que a linguagem efetivamente é.

Ser conhecida como a primeira linguagem para crianças também trouxe um pequeno inconveniente. Muitas pessoas deixam de levar a linguagem a sério por pensarem, apenas, que é uma linguagem lúdica para ensinar apenas crianças, quando pessoas de outras idades podem também fazer grande uso desta linguagem.

1.2 - Academia da Tartaruga

Este livro foca o uso da linguagem Logo focada sob a plataforma Web chamada "**Academia da Tartaruga**" que pode ser acessada de qualquer computador com qualquer sistema operacional por ser um serviço *online*, incluindo-se *tablets* e *smartphones*.

Para obter acesso ao serviço no seu programa de navegação entre o endereço do link Web "https://turtleacademy.com/?lang=pt_BR" para acessar o serviço diretamente no idioma português falado no Brasil. A figura 1.1 mostra a tela inicial do ambiente.

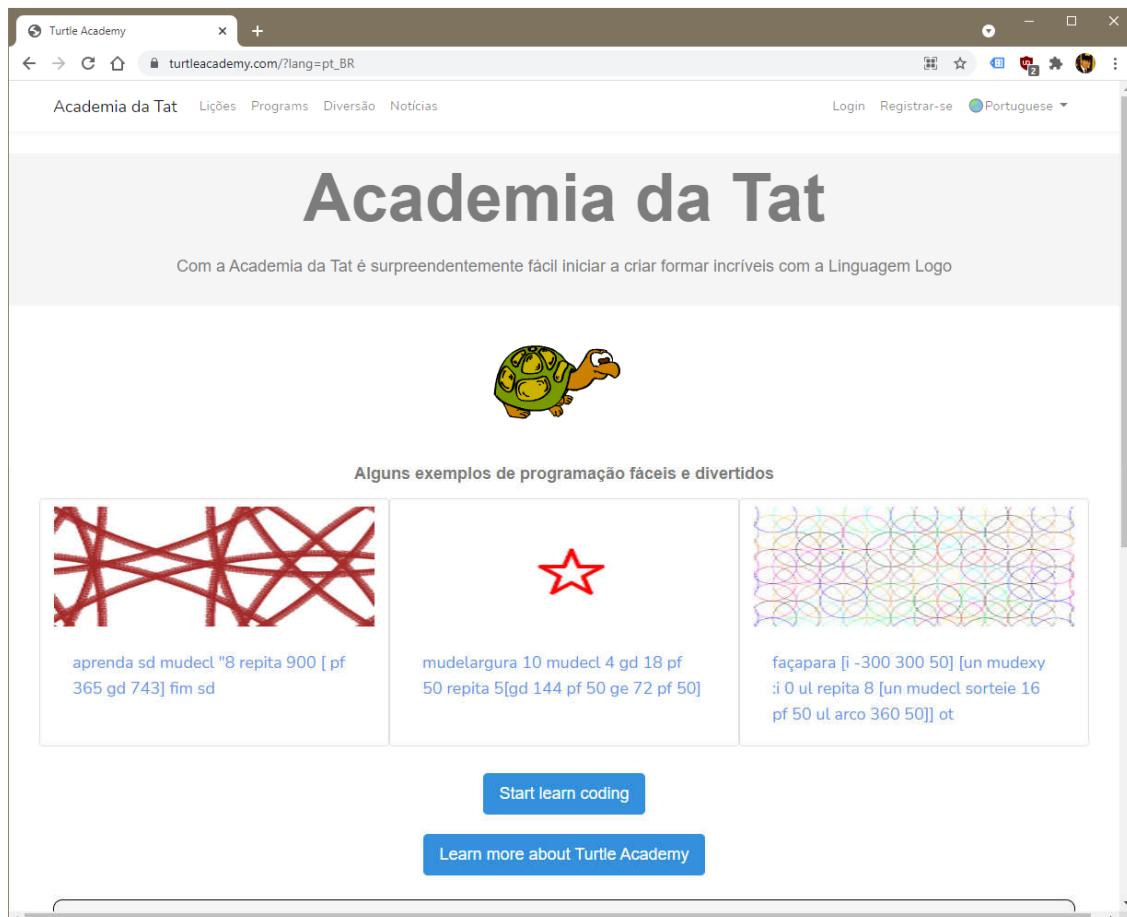


Figura 1.1 - Sítio oficial para uso do serviço "Academia da Tartaruga".

Neste serviço você poderá registrar-se como poderá usá-lo independentemente do registro. Neste livro o uso do ambiente, parte do pressuposto de você não estar registrado no serviço. Para acessar diretamente o ambiente de programação acione no menu superior a esquerda com o ponteiro do *mouse* a opção "**Diversão**" como indicado na figura 1.2.



Figura 1.2 - Seleção da opção "Diversão".

Assim que a opção "**Diversão**" é selecionada é apresentada a página contendo o ambiente operacional de trabalho dimensionado em "**700**" pixels de comprimento (**Width**) e "**500**" pixels de altura (**Height**), sendo "pixels" o menor ponto iluminado no monitor de vídeo para a composição visual de algum elemento (imagem, letras, números, etc.). Veja a figura 1.3.

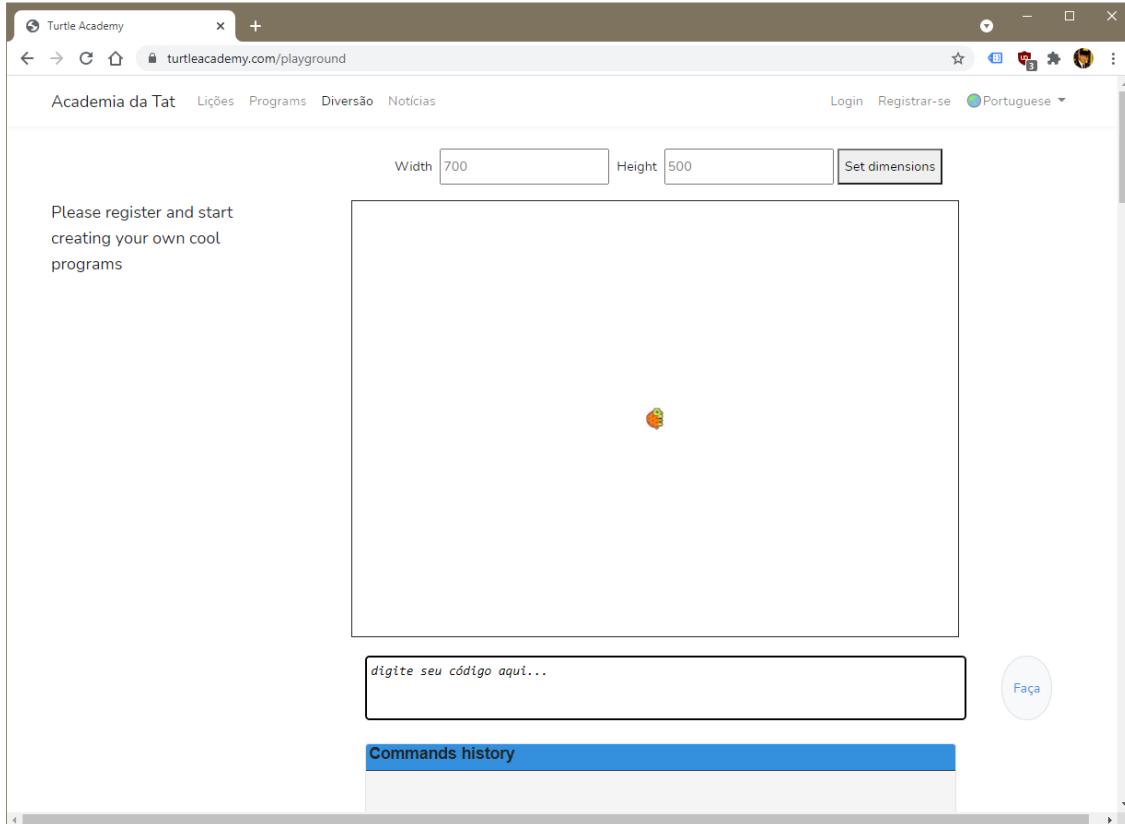


Figura 1.3 - Áreas operacionais de trabalho da "Academia da Tartaruga".

Apesar da entrada no ambiente em português (Brasil) ocorre a apresentação de alguns termos em inglês, ainda não traduzidos. Isso ocorre pelo fato do projeto "*Academia da Tartaruga*" ainda estar em desenvolvimento e adaptação. No entanto, isso não prejudica a aprendizagem, pois quando necessário as informações sobre o que está escrito serão devidamente apontadas.

1.3 - O ambiente operacional

O ambiente "*Academia da Tartaruga*" foi projetado para ser simples, mostrando-se de forma minimalista. Este ambiente de desenvolvimento possui um menu com as opções: **Academia da Tat**, **Lições**, **Programs (Programas)**, **Diversão** e **Notícias** como mostra a figura 1.4.

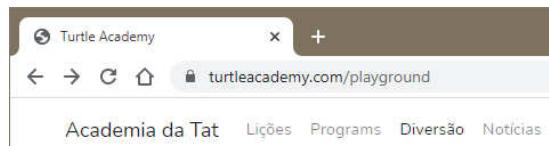


Figura 1.4 - Menu de opções.

A opção "**Academia da Tat**" apresenta a página inicial da plataforma, onde você tem acesso há algumas informações do projeto e alguns exemplos de programas escritos em Logo.

A opção "**Lições**" abre uma página que apresenta 25 lições de aprendizagem que podem ser usadas para o aprofundamento da linguagem suportada na plataforma.

A opção "**Programs**" abre a página da galeria das centenas de milhares de projetos postados pelos usuários registrados da plataforma.

A opção "**Diversão**" abre o ambiente de desenvolvimento operacional.

A opção "**Notícias**" abre a página com diversas informações sobre o projeto (em inglês).

A página mais usada da plataforma é sem dúvida a área de "**Diversão**" composta por alguns elementos estruturais como mostra a figura 1.5.

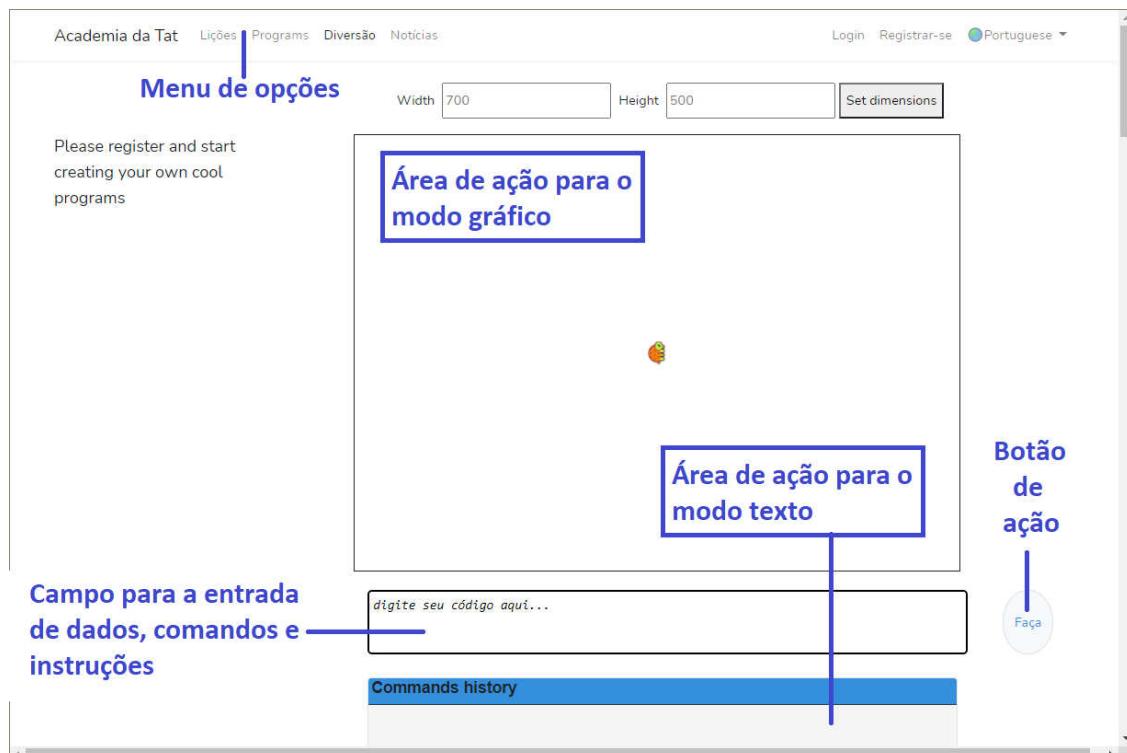


Figura 1.5 - Estrutura de organização do ambiente "Academia da Tartaruga" - área "Diversão".

Na parte inferior da página de "**Diversão**" encontra-se um resumo das principais primitivas (os comandos) da languageLogo suportada pela "*Academia da Tartaruga*".

A mensagem em inglês "*Please register and start creating your own cool programs*" do lado esquerdo superior orienta "*Por favor, registre-se e comece a criar seus próprios programas legais*". Vale ressaltar que não há a necessidade de fazer registro na plataforma.

Os campos superiores "**Width**" (largura) e "**Height**" (altura) permitem que a dimensão da área de ação para o modo gráfico seja alterada para mais ou para menos. Se fizer mudanças nesse campo é necessário acionar o botão "**Set dimensions**" (Define dimensões) para que a sua mudança seja registrada. Vale ressaltar que os exemplos usados neste trabalho foram projetados para operarem em uma dimensão de "**700**" por "**500**" pixels. O valor de alteração destes campos deve ser configurado entre "**100**" por "**999**". Para manter proporcionalidade de tamanho da área gráfica sugere-se alterar os valores usando uma razão percentual. Por exemplo, alterar os valores "**700**" por "**500**" com mais "**12%**", ficando como "**784**" por "**560**" e assim por diante.

A proposta, por ser o ambiente da "*Academia da Tartaruga*" um projeto em desenvolvimento sua tradução para a língua portuguesa não é completa. Assim sendo, não estranhe em alguns momentos o uso de comandos (primitivas) codificadas em inglês que ficam misturadas com primitivas em português.

CAPÍTULO 2 - Ações básicas

As *primitivas* em Logo caracterizam-se por serem o conjunto de comandos e funções básicas da linguagem e de como esses elementos podem ser usados por estudantes para interagirem com o ambiente como um todo. Os comandos ou *primitivas* são palavras que determinam ações a serem executadas pela linguagem como **PARAFRENTE** e **GIRADIREITA** entre outros. As funções por sua vez caracterizam-se por serem recursos operacionais que devolvem uma resposta a sua operação como **PI**, **SOMA**, **INTEIRO** e etc.

2.1 - Primitivas iniciais

O conjunto de comandos na linguagem Logo é extenso. No entanto, não é necessário conhecer todos as primitivas para poder usufruir da linguagem, pois a partir de um pequeno conjunto de ações já é possível realizar algumas ações divertidas.

Antes de começar é importante ter em mente que um comando é uma ação a ser realizada no computador pela linguagem e que este pode ser usado de forma isolada ou acompanhado de um parâmetro. Um comando escrito com ou sem parâmetro pode com o acionamento da tecla <Enter> passar ao computador uma instrução de ação a ser realizada.

Veja o que é preciso para se fazer o desenho de um **quadrado**.

No campo para a entrada de comandos, dados e instruções escreva tanto em letras minúsculas quanto em letras maiúsculas a instrução seguinte e acione após escrever-la a tecla <Enter>:

PARAFRENTE 80

Após executar a instrução anterior formada pelo comando **PARAFRENTE** e pelo parâmetro "80" ocorre o desenho de uma linha de baixo para cima na posição central da tela com 80 pixels (*pixel* é o menor ponto luminoso imprimível na tela do monitor de vídeo - ecrã). A instrução após sua execução é apresentada dentro da área de ação do modo texto, como indicado na figura 2.1.

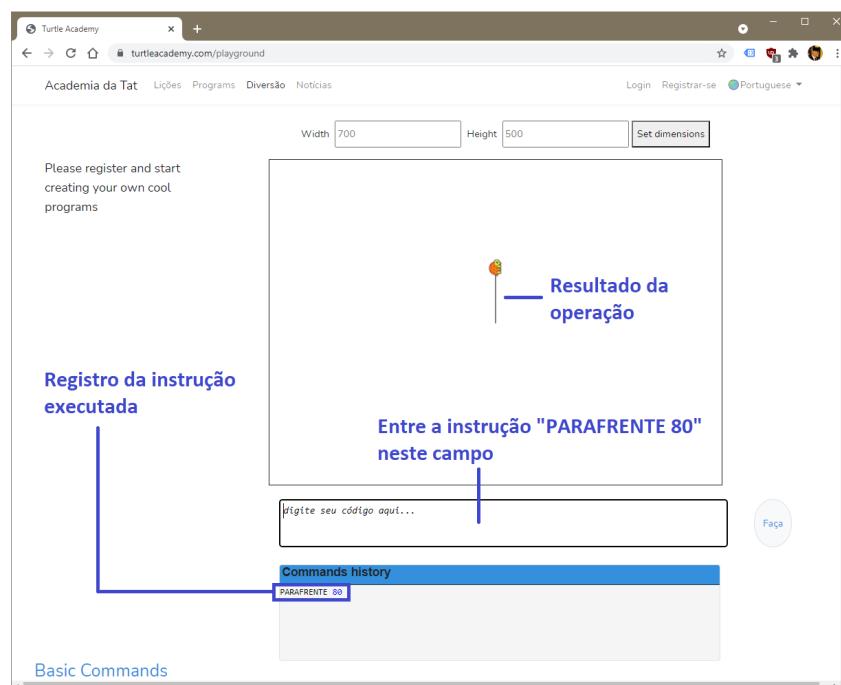


Figura 2.1 - Resultado da ação para a instrução "PARAFRENTE 80".

Além da primitiva **PARAFRENTE**, há sua inversa **PARATRÁS** que faz com que a **TARTARUGA** ande de cima para baixo. Os valores de deslocamento para esses parâmetros será neste obra definido entre "0" e "500" pixels.

O primeiro traço do que deverá ser um quadrado já está definido. Agora é necessário fazer com que o próximo traço seja desenhado em um sentido lateral. Observe que o desenho do primeiro traço ocorreu de baixo para cima, ou seja, ocorreu no sentido **NORTE**. Considere que se deseja fazer com que o próximo traço seja desenhado no sentido **LESTE** ou seja a direita do ponto em que a **TARTARUGA** se encontra. Para este caso, use o comando **GIRADIREITA** com o parâmetro "**90**" a partir da instrução:

GIRADIREITA 90

Veja que a **TARTARUGA** é apontada para a direção **LESTE** como mostra a figura 2.2.

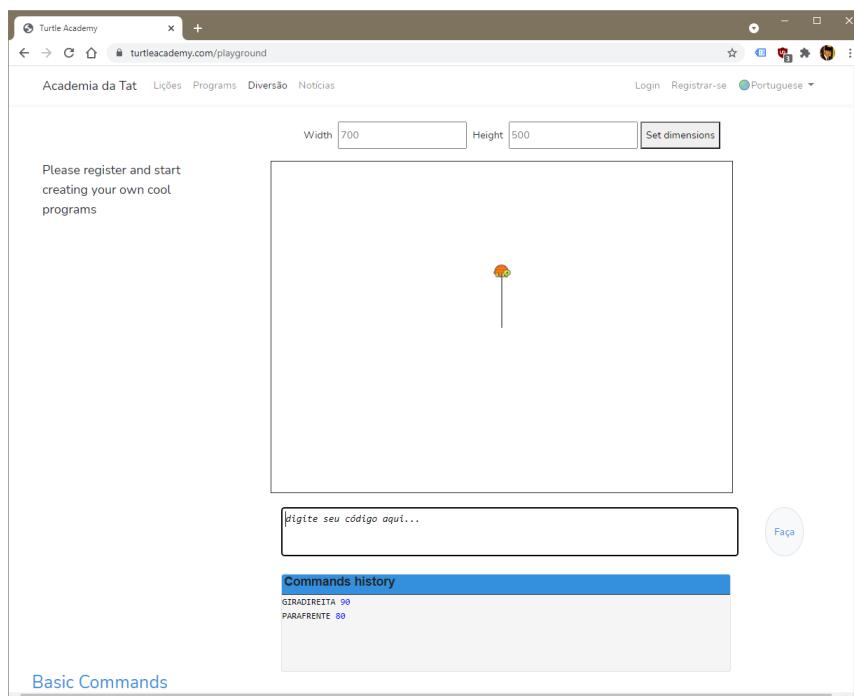


Figura 2.2 - Resultado da tartaruga no sentido LESTE após instrução "GIRADIREITA 90".

Além do comando **GIRADIREITA** que altera o sentido de giro da **TARTARUGA** de **NORTE** para **LESTE**, há seu inverso **GIRAESQUERDA** que altera o sentido de giro da **TARTARUGA** de **NORTE** para **OESTE**. Os valores padrão de parâmetros permitidos para esses comandos são de "0" a "360" graus.

A partir da definição da nova direção a ser seguida basta repetir por mais três vezes a execução das instruções:

PARAFRENTE 80
GIRADIREITA 90

Veja junto a figura 2.3 a apresentação do conjunto de instruções e a imagem do quadrado desenhada. Note que para apresentar o conjunto de instruções o tamanho da janela foi ajustado. Este ajuste pode ser feito com o posicionamento do ponteiro do *mouse* sobre a linha que divide ás áreas de ação do modo gráfico e do modo texto.

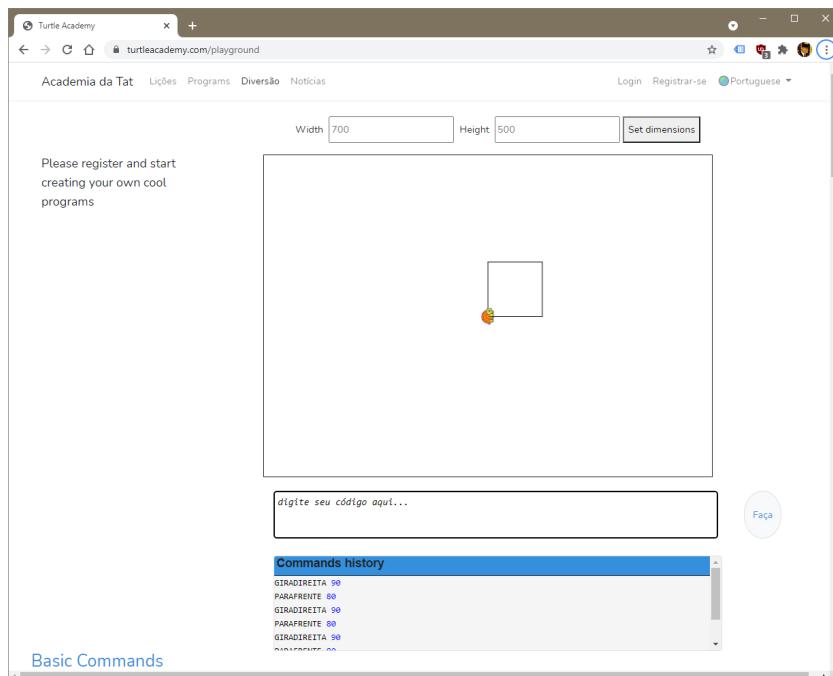


Figura 2.3 - Apresentação de um quadrado.

Note que a sobreposição da TARTARUGA sobre a figura desenhada pode atrapalhar um pouco sua visualização. Neste sentido, é possível pedir que a TARTARUGA seja ocultada a partir do uso da instrução:

OCULTETAT

Observe junto a figura 2.4 a apresentação da imagem do quadrado sem a sobreposição da TARTARUGA.

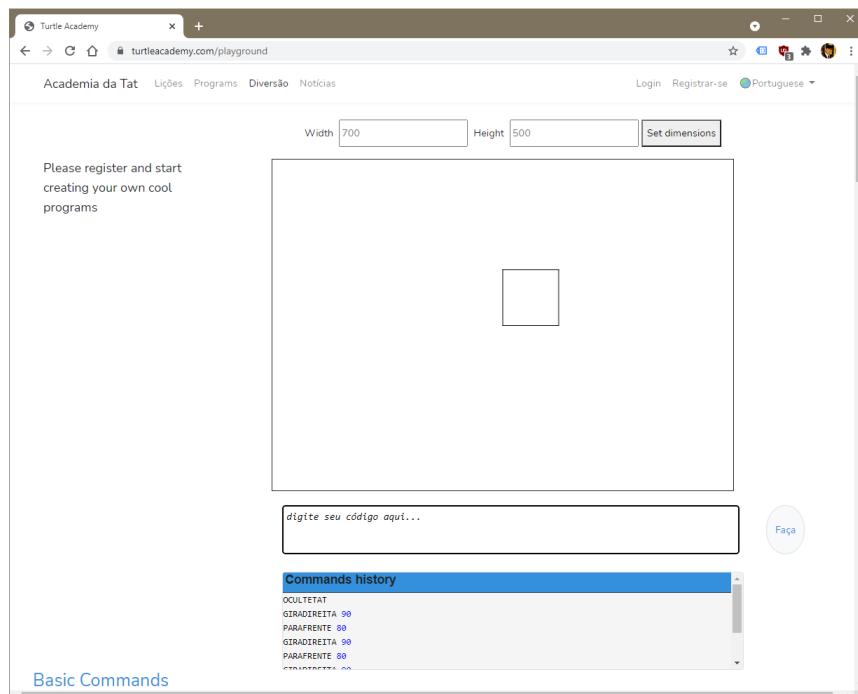


Figura 2.4 - Apresentação de um quadrado sem sobreposição da tartaruga.

Para retornar a apresentação da TARTARUGA use a instrução:

MOSTRETAT

Uma das ações mais importantes é um comando especial que efetua a limpeza da área de ação do modo gráfico e coloca a TARTARUGA na posição central apontando para o norte. Neste momento, execute a instrução:

TARTARUGA

Veja que até este ponto você já aprendeu que uma instrução pode ser definida a partir do uso de um comando (**OCULTETAT**) ou de um comando com parâmetro (**GIRADIREITA 90**) e que a partir dessas diretivas básicas já é possível fazer diversos desenhos geométricos interessantes, ou seja, de um triângulo até uma circunferência. Mas antes de partir para outras criações é importante conhecer mais alguns detalhes da linguagem Logo.

OBS:

Caso queira limpar toda a tela, ou seja, limpar indiscriminadamente as áreas de ação para o modo gráfico e texto, além de tudo que estiver na memória acione o comando de menu "Diversão".

2.2 - Outras interações

Logo é um ambiente interativo que além de desenhar pode realizar diversas operações, como cálculos aritméticos. Antes de mais nada selecione a opção de menu "**Diversão**". A partir do uso da primitiva **ESCREVA** veja algumas operações aritméticas simples interessantes a partir das seguintes instruções:

ESCREVA 5 + 2

ESCREVA 5 - 2

ESCREVA 5 * 2

ESCREVA 5 / 2

Veja que as instruções anteriores são formadas pelo uso de um comando com um parâmetro identificado pela definição de um cálculo aritmético. A figura 2.5 mostra os resultados das operações aritméticas estabelecidas dentro na área de ação do modo gráfico.

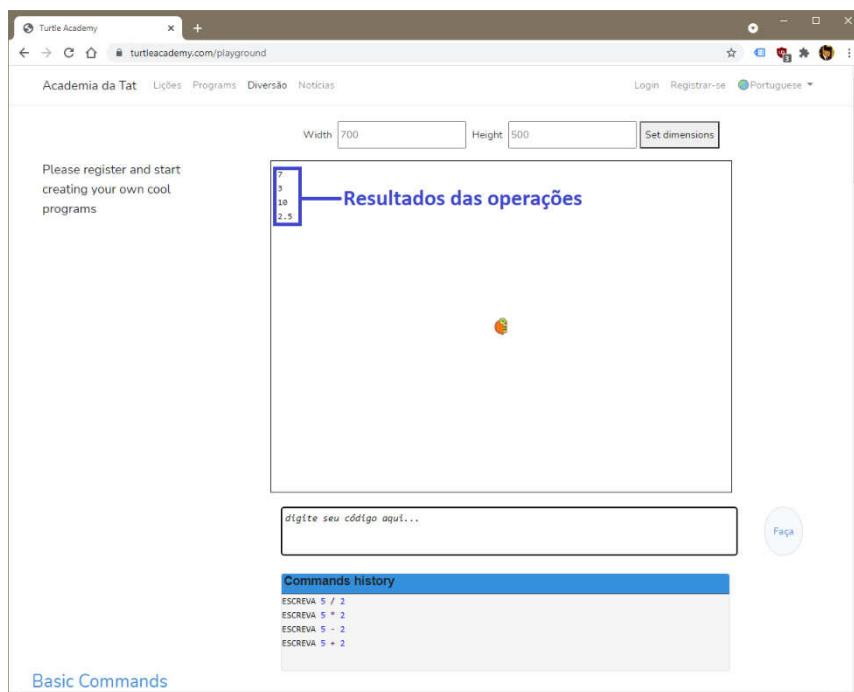


Figura 2.5 - Resultado de operações aritméticas.

O comando **ESCREVA** também pode ser usado para apresentar mensagens. Mas, neste caso é importante ter atenção no que se deseja apresentar. Se for uma mensagem simples, basta após o comando indicar como parâmetro a palavra precedida de aspa inglesa ("), mas se for uma frase é importante que está esteja definida entre colchetes ([e]). Observe os exemplos para apresentação da palavra "Logo" e da frase "Linguagem Logo".

```
ESCREVA "Logo
ESCREVA [Linguagem Logo]
```

Veja o resultado das duas apresentações na figura 2.6.

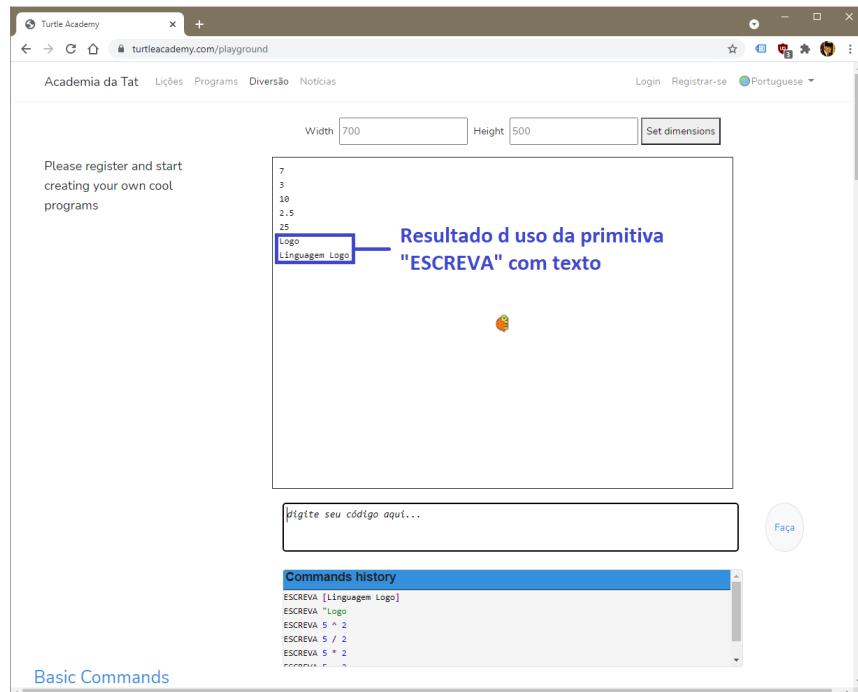


Figura 2.6 - Resultado da apresentação de texto.

Além das operações aritméticas básicas é possível fazer a apresentação de outros resultados baseando-os no uso de *funções*. Veja a seguir alguns exemplos no uso de algumas funções integradas com o uso do comando **ESCREVA**. Antes dos testes selecione a opção de menu "**Diversão**" e acompanhe as seguintes instruções:

```
ESCREVA ABS -5
ESCREVA INT 5 / 2
ESCREVA INT QUOTIENT 5 2
ESCREVA MODULO 5 2
ESCREVA POWER 5 2
ESCREVA 5 ^ 2
ESCREVA PRODUCT 5 2
ESCREVA QUOTIENT 5 2
ESCREVA SQRT 25
ESCREVA SUM 5 2
```

Veja os resultados do uso de funções na figura 2.7. É importante esclarecer que a linguagem Logo do ambiente da "Academia da Taratura" não está totalmente traduzida para o português e é por esta razão que o uso das funções **ABS** (valor positivo), **INT** (valor inteiro), **MODULO**

(resto de divisão), **POWER** (potência), **PRODUCT** (produto), **QUOTIENT** (quociente), **SQRT** (raiz quadrada) e **SUM** (somatório, podendo-se usar **SOMA**) estão grafadas em inglês.

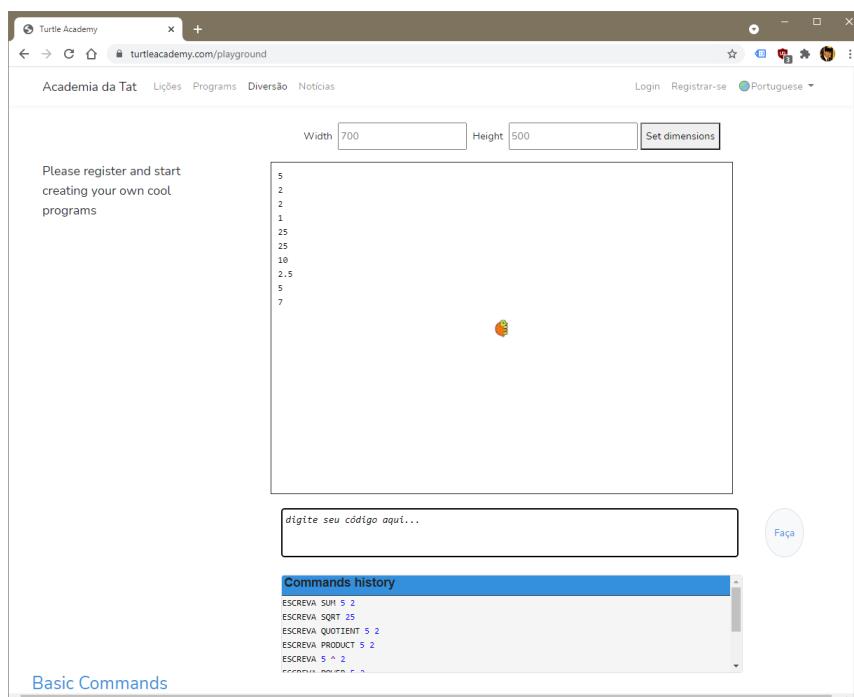


Figura 2.7 - Apresentação dos resultados no uso de funções.

Note que as funções apresentadas são operadas com um ou dois parâmetros. Tomando por base a função **SUM** imagine o desejo de realizar a apresentação da soma de três argumentos. Veja o que acontece:

ESCREVA SUM 1 2 7

O desejo desta ação é, de fato, obter o resultado "10" como resposta a soma de "1 + 2 + 7". No entanto, surpreendentemente ocorre a apresentação do valor "3" que é a soma dos valores "1" e "2" e a indicação da mensagem de erro "**Don't know what to do with 7**" (*Não sei o que fazer com 7*) informando que o ambiente não sabe o que fazer com o valor a mais, neste caso 7. Isto ocorre devido ao fato da função **SUM** (e de outras funções) fazer uso de dois parâmetros e não de três parâmetros como pretendido. Mas há uma maneira de fazer esta ocorrência funcionar, basta executar a instrução:

ESCREVA (SUM 1 2 7)

Veja que ao colocar a função **SUM** e os parâmetros "1", "2" e "7" dentro de parênteses consegue-se obter o resultado da operação pretendida, ou seja, obter o valor "10".

Usar parênteses entre uma função e seus parâmetros é uma maneira de contornar a limitação no uso de parâmetros. Então, mantenha atenção sobre esse detalhe.

A apresentação de elementos em tela também pode ser produzida na área de ação do modo gráfico a partir do uso do comando **ROTULE**. No entanto, é importante considerar que o comando escreve na direção da tartaruga e não na direção da linha. Para ver o texto linearmente é ideal executar antes um giro para o **LESTE**. Observe as instruções seguintes após selecionar a opção de menu "**Diversão**":

GIRADIREITA 90

ROTULE [Línguagem Logo]

A figura 2.8 mostra o resultado da apresentação da frase "**Linguagem Logo**" dentro da área de ação do modo gráfico.

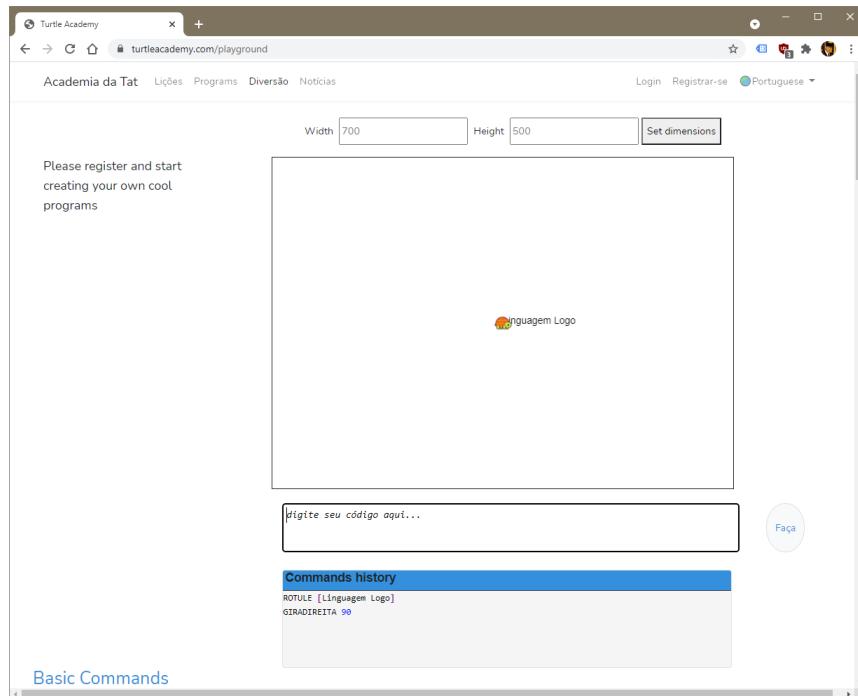


Figura 2.8 - Apresentação de mensagem na área gráfica.

Até este ponto todos as instruções estabelecidas foram executadas pelo Logo. Mas, o que acontece se o Logo não conseguir executar uma instrução. Para fazer um teste escreva no campo de entrada de comandos, dados e instruções a palavra **QUADRADO** e acione <Enter>. Veja a apresentação da mensagem de erro "**Don't know how to QUADRADO**" em branco com fundo vermelho na área de ação do modo gráfico como indica a figura 2.9.

Observe que Logo disse-lhe "*não sei como fazer*". O que isso significa? Significa que Logo ainda não aprendeu a fazer o que lhe foi pedido, apesar de você ter desenhado um quadrado, Logo não sabe o que é, de fato, um quadrado.



Figura 2.9 - Apresentação de mensagem de erro.

Mas, nem tudo é perdido, pois ao dizer que ainda não sabe fazer, Logo abre espaço para que você ensine a tartaruga a fazer um quadrado. Este é um assunto que será visto no próximo capítulo.

2.3 - Primitivas complementares

Além do que foi apresentado, há outras ações básicas a serem conhecidas que ajudarão você a fazer diversas operações.

Quando a tartaruga anda, ela por padrão desenhar, pois carrega com ela um lápis posicionado sobre o ambiente de trabalho. Mas nem sempre se deseja que a tartaruga ande desenhando. Por vezes é interessante que a tartaruga ande sem desenhar.

Para andar sem desenhar, ou seja, com a lápis erguido é necessário antes do movimento pedir a execução da instrução:

USENADA

Para voltar a desenhar basta usar a instrução:

USELÁPIS

A fim de demonstrar o uso dos comandos **USENADA (UN)** e **USELÁPIS (UL)** considere as seguintes instruções:

```
TARTARUGA  
PARAFRENTE 40  
GIRAESQUERDA 90  
USENADA  
PARAFRENTE 40  
GIRADIREITA 90  
USELÁPIS  
PARAFRENTE 40
```

Após executar as instruções anteriores ter-se-á na figura 2.10 a imagem de duas linhas verticais deslocadas.

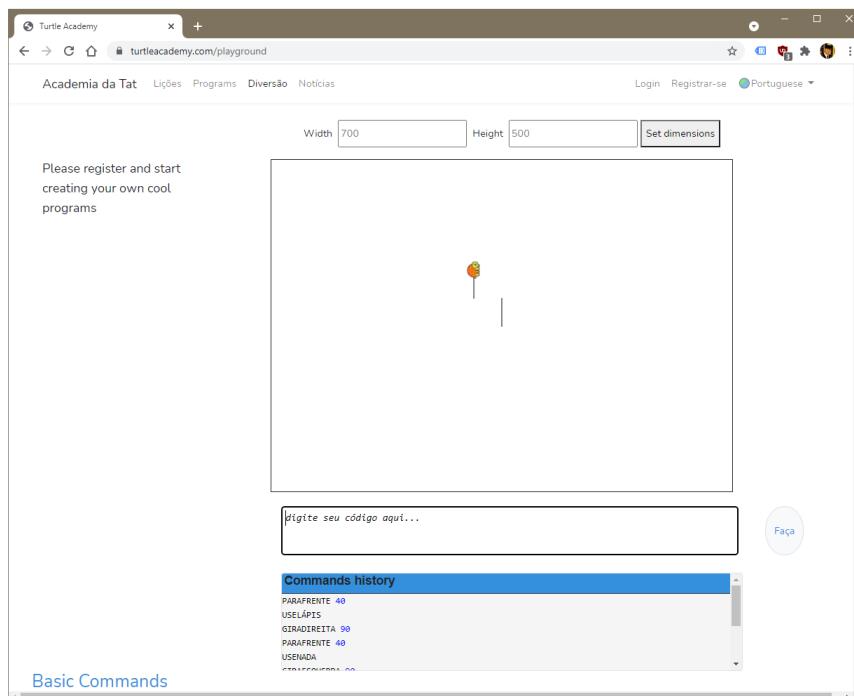


Figura 2.10 - Apresentação de linhas verticais deslocadas.

A partir das primitivas apresentadas você tem em mãos algumas ferramentas básicas para o desenvolvimento de diversas operações e a possibilidade de desenhar diversas formas. Mas antes de sair desenhando é importante ter noção da dimensão, do tamanho, do universo Logo.

Anteriormente foi comentado que o tamanho de *pixels* usado com os comandos **PARAFRENTE** e **PARATRÁS** será entre "1" e "500". Mas, o que aconteceria se fosse definido um valor acima de 500? Assim sendo, após executar o comando **TARTARUGA** execute a instrução seguinte:

PARAFRENTE 300

Se você esperava um erro, ficou semvê-lo, pois a instrução foi processada e executada, como mostra a figura 2.11.

O que aconteceu então? O universo Logo é, na verdade, uma esfera. Isto posto, se executada uma instrução **PARAFRENTE 1050** ocorrerá a sobreposição do movimento em si mesmo.

Há um provérbio chinês que diz que "*se você não mudar a direção, terminará exatamente onde partiu*". Veja que isso ocorre exatamente em Logo.

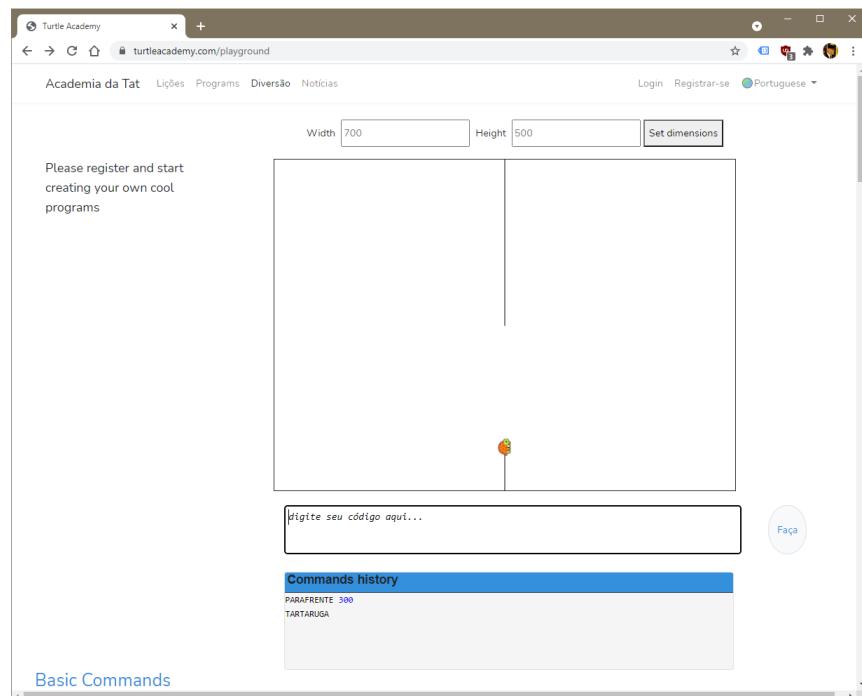


Figura 2.11 - Apresentação de traço além do limite entre 1 e 500.

O plano de ação que permite desenhar na área de ação do modo gráfico pode ser esquematizado segundo a estrutura indicada na figura 2.12 que demonstra de forma simplificada a dimensão padrão da tela de trabalho para deslocamento e graus de giro para locomoção da tartaruga.

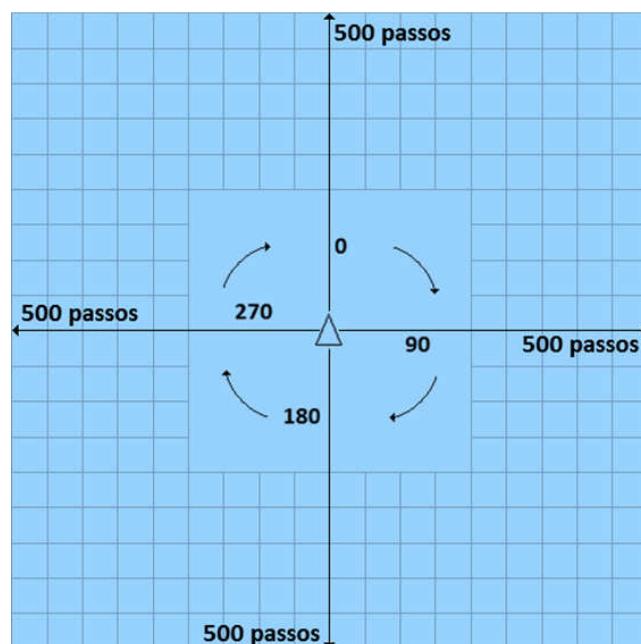


Figura 2.12 - Dimensão padrão do universo Logo no FMSLogo.

Das diretivas apresentadas há alguns comandos que podem ser usados de forma simplificada a partir de siglas de identificação, seus mnemônicos.

É importante considerar que nem todos os comandos possuem este recurso siglas de simplificação ou identificação. Veja na tabela 2.1 a indicação de alguns dos comandos apresentados nesta obra que possuem siglas de identificação.

Mas tenha cuidado no uso de primitivas a partir das siglas de simplificação. Somente as use quando você tiver certeza absoluta de seu significado.

COMANDO	SIGLA
PARAFRENTE	PF
PARATRÁS	PT
GIRADIREITA	GD
GIRAESQUERDA	GE
OCULTETAT	OT
MOSTRETAT	MT
TARTARUGA	TAT
USENADA	UN
USELÁPIS	UL

Tabela 2.1 - Primitivas simplificadas.

CAPÍTULO 3 - Ações especializadas

As operações em Logo, no que tange, ao universo da *geometria da tartaruga* vão além das primitivas apresentadas. Neste contexto, desenhar um quadrado ou triângulo não é difícil apesar de maçante, mas desenhar figuras geométricas com maior número de lados poderá se tornar inviável. É neste sentido que entram outras primitivas de apoio especializadas em facilitar o uso de certos recursos da linguagem, as quais são apresentadas ao longo deste capítulo.

3.1 - Repetições

O desenho de um quadrado pode ser produzido com uso básico da combinação das primitivas **PARAFRENTE**, **PARATRÁS**, **GIRADIREITA** ou **GIRAESQUERDA** repetidos quatro vezes. A combinação de uso das primitivas é de cunho pessoal ou da necessidade do que se deseja efetivamente fazer.

Para facilitar repetições Logo possui uma primitiva chamada **REPITA** com a finalidade de repetir um grupo de instruções definidas entre colchetes um certo número de vezes. O comando (primitiva) **REPITA** para ser usado deve seguir a seguinte estrutura sintática:

REPITA <n> [<instruções>]

Onde, o indicativo "**n**" refere-se a quantidade de repetições e o indicativo "**instruções**" refere-se as ações que se deseja executar. Para desenhar um quadrado, por exemplo, use a instrução:

REPITA 4 [PF 80 GD 90]

A figura 3.1 mostra a apresentação de um quadrado desenhado a partir do uso da instrução "**REPITA 4 [PF 80 GD 90]**" formada pelas primitivas **REPITA**, **PF** e **GD**.

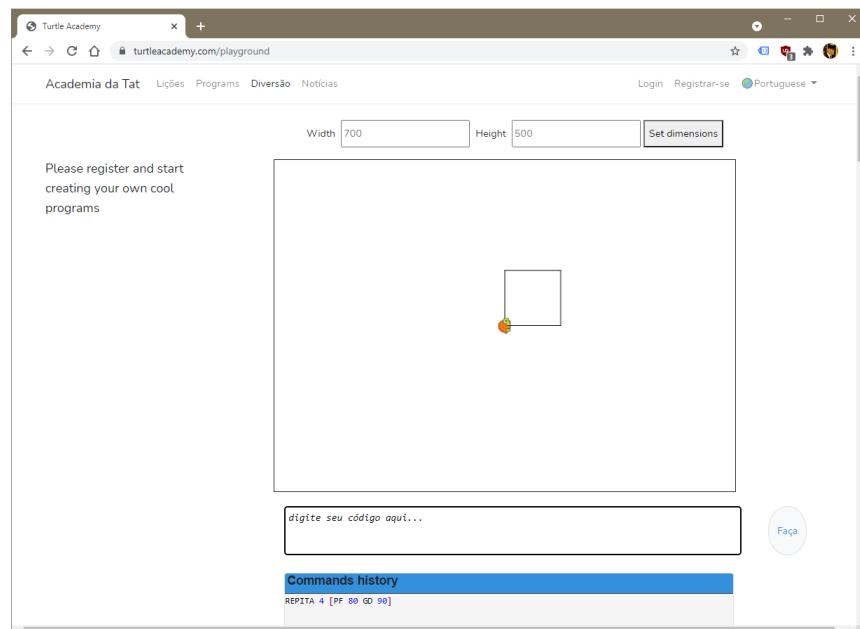


Figura 3.1 - Quadrado apresentado a partir do uso do comando "REPITA".

Veja, por exemplo, a apresentação de um triângulo equilátero com o uso do comando **REPITA**:

TAT
REPITA 3 [PF 80 GD 120]

A figura 3.2 mostra a apresentação de um triângulo equilátero desenhado a partir do uso do comando **REPITA**.

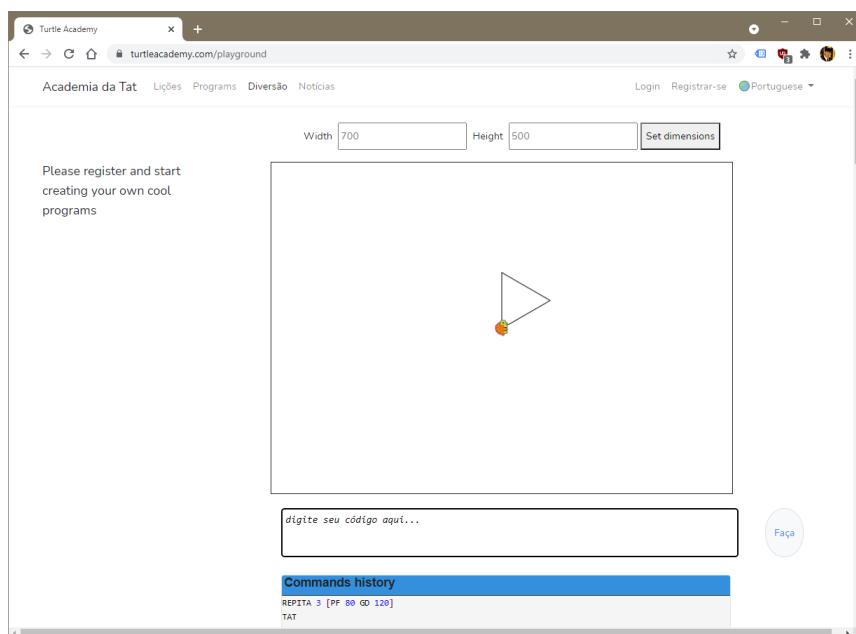


Figura 3.2 - Triângulo equilátero apresentado a partir do uso do comando "REPITA".

Note que para desenhar um triângulo equilátero usou-se a medida de graus como "**120**". No ambiente Logo usa-se a medida externa da figura. Se fosse usado a medida interna o valor para um triângulo equilátero seria "**60**".

Agora veja o desenho de um pentágono. Assim sendo, execute a instrução:

**TAT
REPITA 5 [PF 80 GD 72]**

A figura 3.3 mostra o resultado obtido.

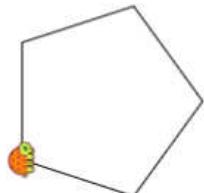


Figura 3.3 - Pentágono.

Agora pode estar em sua mente a pergunta que não quer calar. *Como saber exatamente o valor dos graus a ser usado para a definição de certa figura geométrica?* Recorde que os comandos **GIRADIREITA (GD)** e **GIRAESQUERDA (GE)** podem ser usados com valores entre "**0**" e "**360**". No universo Logo a menor figura geométrica possui três lados, que é um triângulo, por sua vez a maior figura geométrica é a circunferência com trezentos e sessenta lados.

Veja a instrução:

**TAT
REPITA 360 [PF 1 GD 1]**

A figura 3.3 mostra o resultado obtido.

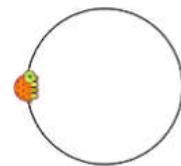


Figura 3.4 - Circunferência.

A resposta à pergunta é que basta você pegar o número de lados desejados e dividi-los por "**360**" para obter o valor exato de graus a ser usado para desenhar determinada figura geométrica.

A partir dessas orientações fica fácil desenhar qualquer figura geométrica plana compreendida entre "**3**" e "**360**" lados, ou seja, desenhar um polígono. Como ilustração veja a tabela 3.1 com

os nomes de alguns polígonos, seus respectivos lados e o valor calculado dos ângulos de rotação.

POLÍGONO	LADOS	360 / LADOS	ÂNGULO
TRIÂNGULO	3	360 / 3	120.00
QUADRILÁTERO	4	360 / 4	90.00
PENTÁGONO	5	360 / 5	72.00
HEXÁGONO	6	360 / 6	60.00
HEPTÁGONO	7	360 / 7	51.43
OCTÓGONO	8	360 / 8	45.00
ENEÁGONO	9	360 / 9	40.00
DECÁGONO	10	360 / 10	36.00
UNDECÁGONO	11	360 / 11	32.73
DODECÁGONO	12	360 / 12	30.00
TRIDECAZAGONO	13	360 / 13	27.69
TETRADECÁGONO	14	360 / 14	25.71
PENTADECÁGONO	15	360 / 15	24.00
HEXADECÁGONO	16	360 / 16	22.50
HEPTADECÁGONO	17	360 / 17	21.18
OCTODECÁGONO	18	360 / 18	20.00
ENEADECÁGONO	19	360 / 19	18.95
ICOSÁGONO	20	360 / 20	18.00
TRIACONTÁGONO	30	360 / 30	12.00
TETRACONTÁGONO	40	360 / 40	9.00
PENTACONTÁGONO	50	360 / 50	7.20
HEXACONTÁGONO	60	360 / 60	6.00
HEPTACONTÁGONO	70	360 / 70	5.14
OCTOCONTÁGONO	80	360 / 80	4.50
ENEACONTÁGONO	90	360 / 90	4.00
HECTÁGONO	100	360 / 100	3.60
CIRCUNFERÊNCIA	360	360 / 360	1.00

Tabela 3.1 - Algumas medidas de ângulos.

Além da produção de figuras geométricas planas há outras possibilidades de geração de imagens na linguagem. Mas este é um assunto a ser visto um pouco mais adiante.

3.2 - Procedimentos

Já foi comentado que o fato de Logo apresentar uma figura não significa em absoluto que Logo saiba fazer a figura. Para que Logo aprenda a fazer uma figura é necessário que você ensine Logo a fazer.

A ideia de "ensinar" um computador a realizar certa tarefa, de modo que o computador aprenda e tenha esse "conhecimento" armazenado para uso futuro chamassem comumente de "inteligência artificial".

A "inteligência artificial" é a junção de duas palavras distintas com significados absolutos: *inteligência* é a faculdade de conhecer, compreender e aprender a resolver problemas e de adaptá-los a novas situações e *artificial* significa aquilo que não revela naturalidade, sendo produzido pelo ser humano e não pela natureza. Desta forma, pode-se entender que a "inteligência artificial" é a fabricação de conhecimentos específicos processados por computadores e controlados dentro das bases da Ciência da Computação.

Uma forma de "ensinar" a linguagem Logo a realizar tarefas de modo que se desenvolva a ideias de "inteligência artificial" é fazer uso de rotinas de scripts de programas chamadas de **procedimentos**.

A criação de procedimentos no ambiente da "Academia da Tartaruga" é produzida com o uso dos comandos (primitivas) **APRENDA** e **END** (primitiva *FIM* sem tradução) a partir da seguinte sintaxe:

```
APRENDA <nome>
  <instruções>
END
```

Onde, o indicativo "**nome**" refere-se a definição de um nome de identificação para o procedimento a ser usado no campo de comandos e "**instruções**" o conjunto de ações a ser executada dentro do procedimento.

Note que se tem duas categorias de primitivas na linguagem Logo: as primitivas internas (pertencente a linguagem) e as primitivas externas (definidas por humanos) para adaptar a linguagem Logo a necessidades particulares.

Cada procedimento criado é a definição de um grau de conhecimento que o ambiente Logo pode artificialmente obter dando-lhe um nível de inteligência, além do que foi projetado. Assim sendo, veja a definição de um procedimento que desenhe um quadrado chamado "**QUAD1**". Para tanto, coloque no campo de entrada de comandos e instruções o código do procedimento seguinte:

```
APRENDA QUAD1
  REPITA 4 [PF 80 GD 90]
END
```

A partir deste instante o Logo sabe desenhar um quadrado por meio da primitiva derivada chamada "**QUAD1**". Assim sendo, no capo de comandos, dados e instruções execute a instrução:

QUAD1

A figura 3.5 mostra o resultado obtido a partir da definição interna do conhecimento do que é um quadrado para a linguagem Logo.

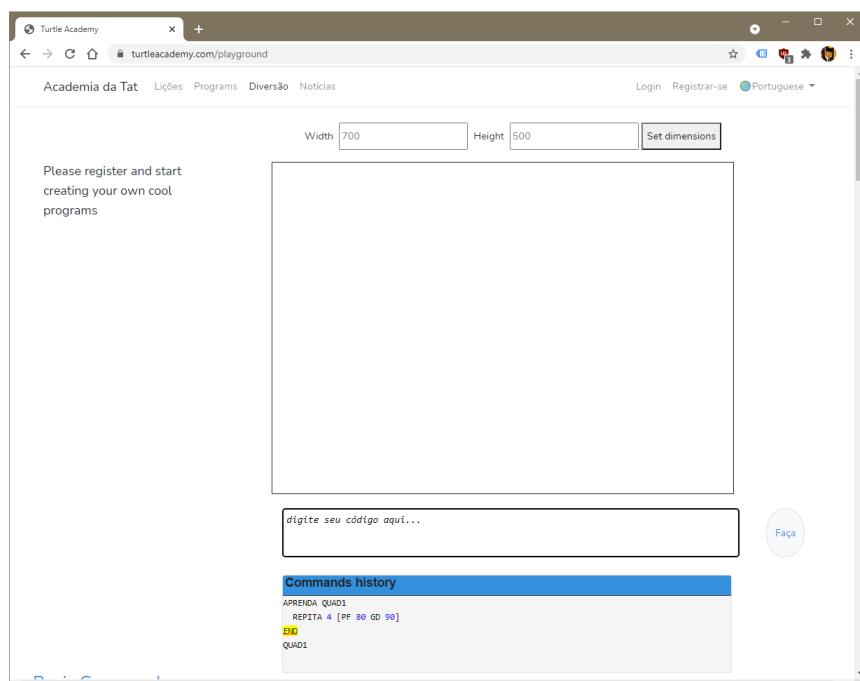


Figura 3.5 - Quadrado desenhado a partir da definição de um conhecimento artificial.

Assim como é possível repetir as primitivas internas também é possível repetir os procedimentos (que são as rotinas definidas a partir de ações derivadas). Por exemplo, imagine querer repetir o procedimento "**QUAD1**" por quatro vezes com ângulo de giro de "90" graus para a esquerda. Assim sendo, execute a instrução:

```
TAT
REPITA 4 [QUAD1 GE 90]
```

A figura 3.6 mostra o resultado desta operação.

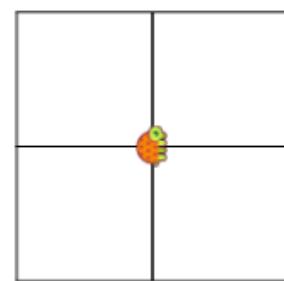


Figura 3.6 - Quadrado repetido.

Dependendo do valor de ângulo de giro é possível formar imagens geométricas muito diferentes como um conjunto de quadrados intercalados a partir da execução da instrução:

```
TAT
REPITA 8 [QUAD1 GE 45]
```

A figura 3.7 mostra o resultado desta operação.

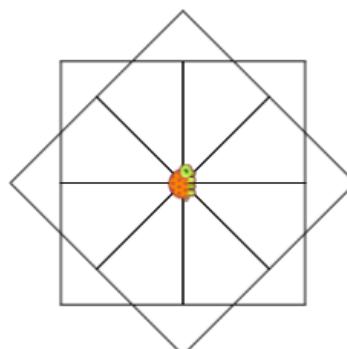


Figura 3.7 - Quadrados intercalados.

Muito bem, a partir de uma visão sobre a definição de procedimento será definido outro procedimento chamado "**PENTA1**" que desenha um pentágono. Assim, escreva o procedimento:

APRENDA PENTA1

REPITA 5 [PF 50 GD 360 / 5]

END

Agora execute a instrução:

TAT

PENTA1

A figura 3.8 mostra o resultado desta operação.

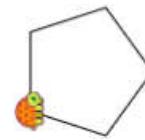


Figura 3.8 - Pentágono.

A partir da definição da imagem de um pentágono veja os dois exemplos de imagens geradas a partir do procedimento "PENTA1".

Execute primeiro a instrução:

TAT

REPITA 5 [PENTA1 GD 72]

A figura 3.9 mostra o resultado desta operação.

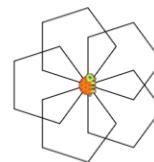


Figura 3.9 - Pentágono 1.

Depois execute a instrução:

TAT

REPITA 10 [PENTA1 GD 36]

A figura 3.10 mostra o resultado desta operação.

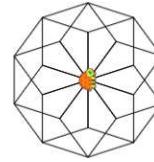


Figura 3.10 - Pentágono 2.

3.3 - Sub-rotinas

A definição de procedimentos trás para a linguagem Logo a possibilidade de se fazer a criação de comandos derivados. Todo comando derivado é baseado sobre o uso de alguma primitiva padrão da linguagem.

O fato que se definir procedimentos e dar-lhes a mesma capacidade que possuem os comandos traz muita mobilidade, pois um procedimento pode ser usado da mesma forma que um comando é usado no estabelecimento de instruções. Desta forma, torna-se naturalmente possível usar um procedimento dentro de outro procedimento, o que caracteriza a existência de sub-rotinas.

Para experimentar esta ideia considere realizar o desenho de uma flor com oito pétalas. Veja que uma flor, é grosso modo, um conjunto de pétalas. Uma pétala por sua vez pode ser a junção de duas meias circunferências.

Veja pelo que é descrito que o desenho de uma flor será realizado a partir de três procedimentos interligados. Assim sendo, considere os procedimentos "MEIOCIRC", "PETALA" e "FLOR". Escreva no ambiente um procedimento após o outro sem o uso de linhas em branco.

APRENDA MEIOCIRC

REPITA 90 [PF 1 GD 1]

END

```
APRENDA PETALA
```

```
MEIOCIRC
```

```
GD 90
```

```
MEIOCIRC
```

```
END
```

```
APRENDA FLOR
```

```
TAT
```

```
REPITA 8 [PETALA GD 45]
```

```
END
```

Veja que o efeito em cascata para se fazer o desenho da flor é o que se chama de sub-rotinas, ou seja, "**MEIOCIRC**" é uma sub-rotina usada por "**PETALA**" que, por sua vez, é uma sub-rotina usada por "**FLOR**".

Na sequência execute a instrução:

```
FLOR
```

A figura 3.11 mostra o resultado da execução do procedimento "**FLOR**".

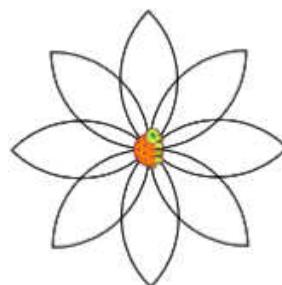


Figura 3.11 - Execução do procedimento "FLOR".

Imagine desenhar uma flor mais encorpada com um número maior de pétalas, por exemplo vinte pétalas. Para tanto, execute a instrução:

```
TAT
```

```
REPITA 20 [PETALA GD 36]
```

A Figura 3.12 mostra a imagem de uma flor com vinte pétalas.



Figura 3.12 - Execução do procedimento "PETALA" para criar flor.

A partir da definição de sub-rotinas é possível diminuir muito a carga de trabalho, pois é possível criar comandos derivados e especializados em desenhar parte de uma imagem e formar, a partir daí, imagens com as combinações dos comandos definidos em conjunto com as primitivas e mesmo as funções da linguagem Logo.

3.4 - Variável

Você viu anteriormente o uso de instruções baseadas em primitivas simples e primitivas com o uso de parâmetros.

No tópico anterior você aprendeu a escrever seus próprios comandos a partir da definição de procedimentos. Os procedimentos, então criados, se assemelharam aos comandos simples da linguagem Logo. É chegada a hora de aprender a definir os próprios parâmetros para certo procedimento. É aqui que entra a ideia de uso de *variáveis*.

Uma *variável* do ponto de vista mais amplo para a computação é uma região de memória que armazena determinado valor por certo espaço de tempo. O valor armazenado em memória poderá ou não ser usado em ações de processamento.

Para criar variáveis em memória deve-se fazer uso do comando **ATRIBUA** a partir da estrutura sintática:

ATRIBUA <"variável> <valor>

Onde, o indicativo "**variável**" refere-se ao nome de identificação da variável definido após o uso do símbolo de aspa inglesa ("") e "**valor**" a definição do valor associado ao nome da variável. Os caracteres maiúsculos e minúsculos interferem no nome de uma variável.

Para apresentar o conteúdo de uma variável é importante que seja usado antes do nome da variável o símbolo de dois pontos (:). Por exemplo, veja a definição e apresentação do conteúdo de uma variável chamada "**PAISES**" com o valor **Brasil, França e Argentina**.

ATRIBUA "PAISES [Brasil França Argentina]

ESCREVA :PAISES

A figura 3.13 mostra o resultado da ação anterior na área de ação do modo texto.

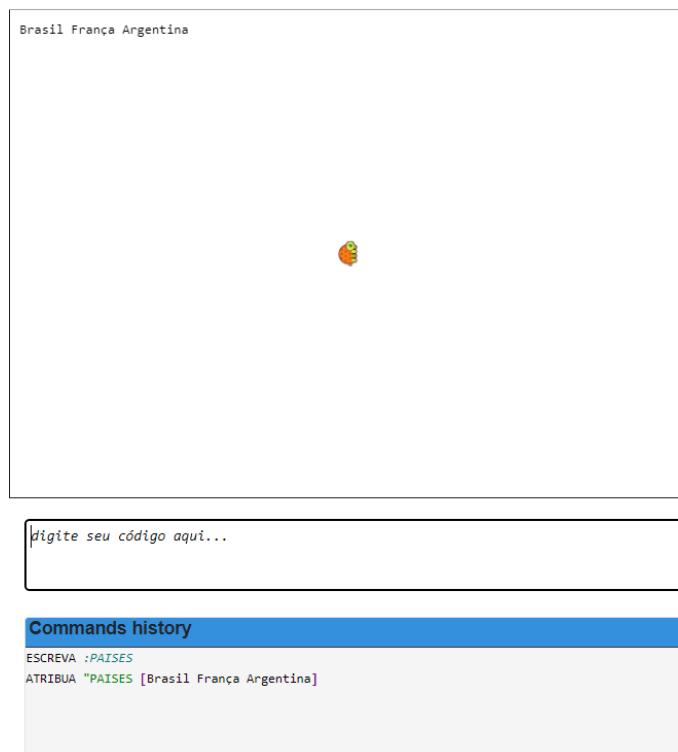


Figura 3.13 - Definição e apresentação de variável.

Veja outro exemplo de definição de variável e visualização do nome *Manzano*:

```
ATRIBUA "NOME "Manzano
ESCREVA :NOME
```

Perceba que para imprimir o conteúdo de uma variável usa-se o símbolo dois pontos. Cuidado em não fazer uso do símbolo aspa inglesa. Caso use o nome de uma variável com aspa inglesa a linguagem Logo não entenderá que se trata de uma variável e considerará como sendo apenas uma palavra a ser escrita usando o nome da variável.

Escreva a instrução:

```
ESCREVA :PAISES
```

E em seguida, escreva a instrução

```
ESCREVA "PAISES
```

Veja na figura 3.14 a diferença de resultado apresentado para cada uma das formas de acesso.

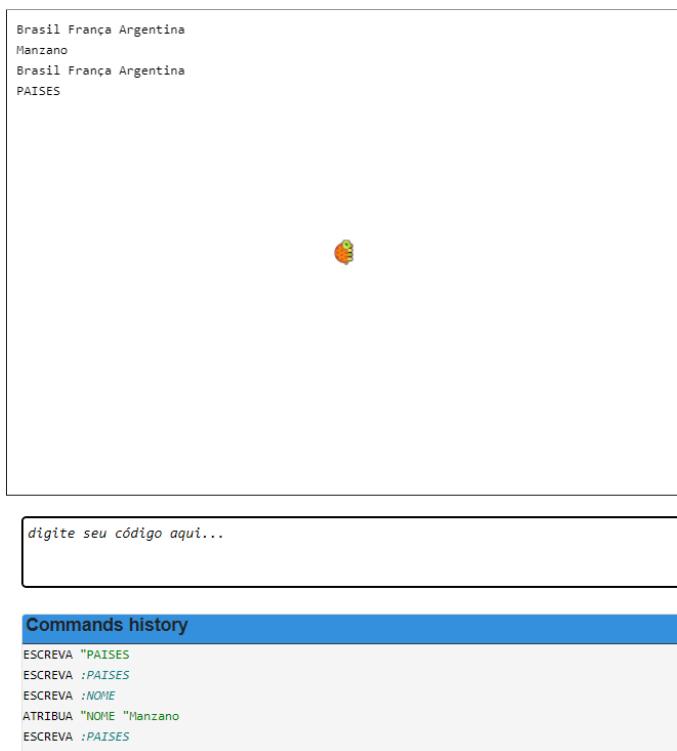


Figura 3.14 - Diferença no uso dos símbolos dois pontos e aspa inglesa.

No entanto, é possível fazer com que ocorra a apresentação do conteúdo de uma variável utilizando-se o símbolo aspa inglesa. Mas neste caso, é necessário fazer uso da primitiva **THING** (que pode ser traduzido como *COISA* ou melhor dizendo *OBJETO*) antes no nome da variável identificada com aspa inglesa. Observe a instrução seguinte:

```
ESCREVA THING "PAISES
```

O efeito no uso da primitiva **THING** antes no nome da variável com aspa inglesa, ou seja, a instrução **ESCREVA THING "PAISES** é exatamente o mesmo resultado obtido a partir da execução da instrução **ESCREVA :PAISES**. Veja a figura 3.15.

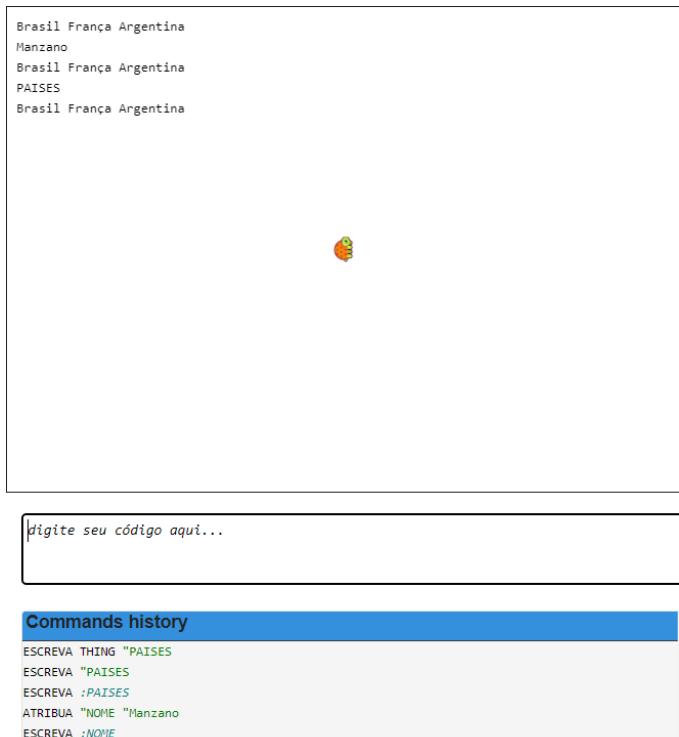


Figura 3.15 - Resultado da instrução "ESCREVA VALOR "PAISES".

A partir de uma visão geral da definição e uso de variáveis é possível criar um procedimento que, por exemplo, desenhe um quadrado em que o tamanho da figura será informado no uso e não dentro do procedimento. Assim sendo, informe o seguinte procedimento:

```
APRENDA QUAD2 :TAMANHO
REPITA 4 [PF :TAMANHO GD 90]
END
```

Em seguida execute as instruções:

```
TAT
QUAD2 40
QUAD2 60
QUAD2 80
QUAD2 90
```

Perceba que para cada chamada do procedimento "**QUAD2**" foi usado em conjunto um parâmetro diferente proporcionando efeitos diferenciados como apresenta a figura 3.16.



Figura 3.16 - Resultado a partir do uso de procedimento "QUAD2" com parâmetro.

A partir desses recursos é possível criar um procedimento mais "inteligente" que a partir da informação do tamanho de traço e quantidade de laços desenhe qualquer figura entre "3" e "360" graus. Para tanto, veja o procedimento "**POLIS**":

```

APRENDA POLIS :TAMANHO :LADO
REPITA :LADO [PF :TAMANHO GD 360 / :LADO]
END

```

A partir desta definição pode-se fazer uso, por exemplo, das seguintes instruções:

```

POLIS 80 3 - Desenha triângulo
POLIS 80 4 - Desenha quadrado
POLIS 80 5 - Desenha pentágono
POLIS 80 6 - Desenha hexágono
POLIS 1 360 - Desenha circunferência

```

A partir dessas orientações você já tem em mãos os recursos essenciais para a definição de parâmetros e de procedimentos. Assim sendo, já é possível criar comandos simples e comandos com parâmetros.

3.5 - Decisão

Logo é uma linguagem que possui internamente um nível de "inteligência" interessante. Entre as diversas possibilidades da linguagem oferecidas no ambiente "*Academia da Tartaruga*" há as primitivas **IFELSE** (traduzido como *SESENÃO*) e **IF** (traduzido como *SE*) que permitem a linguagem tomar decisões. Observe as seguintes estruturas sintáticas:

```

IF <condição> <[ação verdadeira]>
IFELSE <condição> <[ação verdadeira]> <[ação falsa]>

```

Onde, o indicativo "**condição**" (não importando **IF** ou **IFELSE**) refere-se a definição de uma relação lógica que devolve como resposta um valor lógico "*FALSO*" representado pelo valor numérico inteiro "**0**" (zero) ou um valor lógico "*VERDADEIRO*" representado pelo valor numérico inteiro "**1**" (um). O indicativo "**[ação verdadeira]**" corresponde a definição da ação a ser realizada dentro de uma lista caso a condição seja verdadeira, tanto para **IF** como para **IFELSE**. O indicativo "**[ação falsa]**" é executado quando a condição para **IFELSE** for falsa. A primitiva **IF** é usada na tomada de decisão simples e a primitiva **IFELSE** é usada na tomada de decisão composta.

Para realizar a definição da condição usada com as primitivas **IF** e **IFELSE** é necessário levar em consideração o uso de operadores específicos para o estabelecimento das relações entre variáveis com variáveis ou de variáveis com constantes. A tabela 3.2 descreve o conjunto de *operadores relacionais* usados no estabelecimento de condições.

OPERADOR	SIGNIFICADO
=	IGUAL A
>	MAIOR QUE
<	MENOR QUE
>=	MAIOR OU IGUAL A
<=	MENOR OU IGUAL A
<>	DIFERENTE DE

Tabela 3.2 - Operadores relacionais.

Para realizar um teste rápido sobre o uso de *operadores relacionais* execute as seguintes instruções observando a apresentação dos valores "**0**" para falso e "**1**" para verdadeiro:

```
ESCREVA 1 = 1 (mostra: 1)
ESCREVA 1 <> 1 (mostra: 0)
ESCREVA 1 < 2 (mostra: 1)
ESCREVA 1 > 2 (mostra: 0)
ESCREVA 1 >= 1 (mostra: 1)
ESCREVA 2 <= 2 (mostra: 1)
```

Como exemplo no uso de tomada de decisão composta considere o desenvolvimento de um procedimento chamado "**MAXIMO**" que retorne o maior valor numérico a partir de dois valores fornecidos como parâmetro. Informe o código seguinte:

```
APRENDA MAXIMO :A :B
  IFELSE :A > :B [ESCREVA :A] [ESCREVA :B]
END
```

Ao término desta definição feche o programa **Editor** com "**Arquivo/Guardar e Sair**" ou acione as teclas "**<Ctrl> + <D>**". Em seguida execute separadamente as instruções:

```
MAXIMO 9 11
MAXIMO 15 8
```

Veja na figura 3.17 a apresentação do resultado das operações.

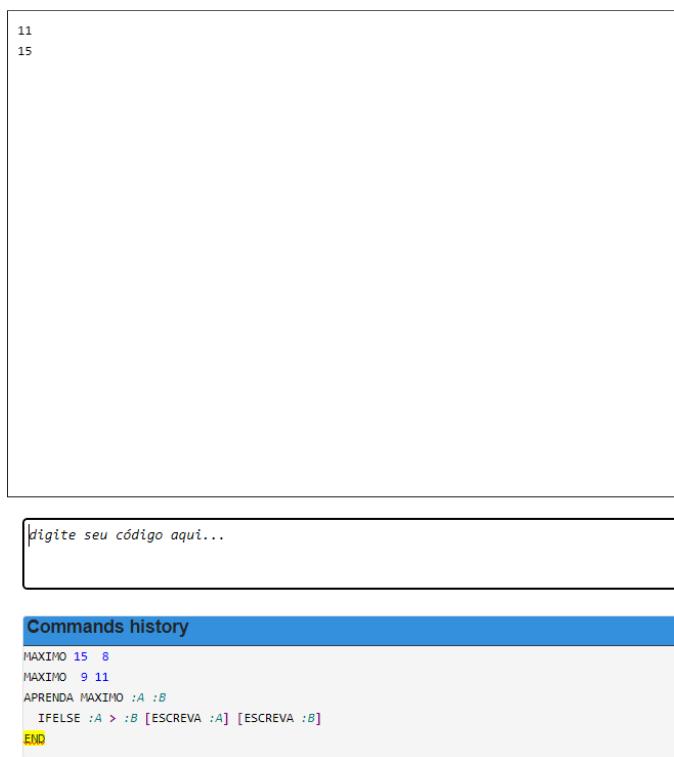


Figura 3.17 - Resultado de uma tomada decisão.

No sentido de demonstrar outro exemplo de tomada de decisão composta considere o procedimento "**PAR**" que apresenta a mensagem "**Ok**" se o valor numérico for par ou mostre a mensagem "**Erro**" caso o valor numérico não seja par. Assim sendo, codifique no ambiente o seguinte código:

```
APRENDA PAR :N
```

```
IFELSE (MODULO :N 2) = 0 [ESCREVA [Ok]] [ESCREVA [Erro]]
END
```

Em seguida execute separadamente as instruções:

```
PAR 2
PAR 3
```

Veja na figura 3.18 a apresentação do resultado das operações.

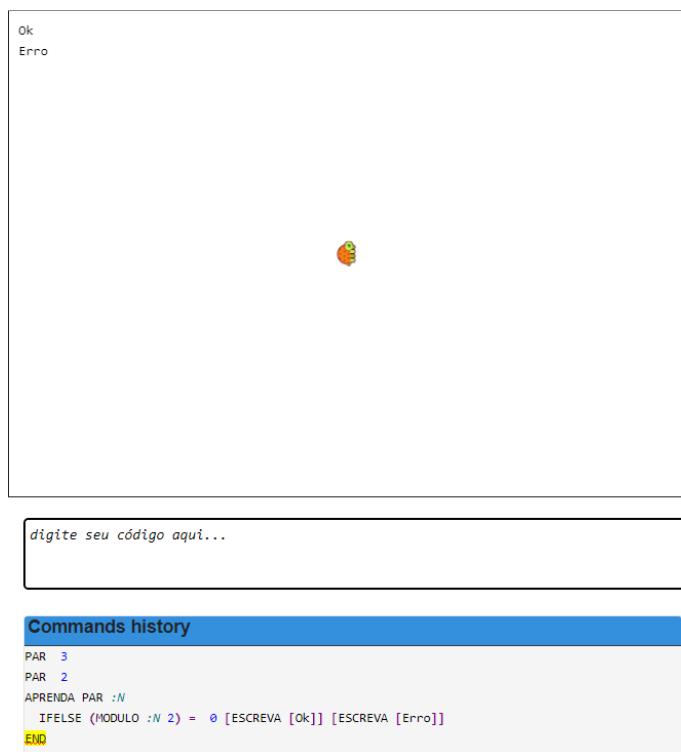


Figura 3.18 - Resultado do procedimento "PAR".

Observe a partir desses dois exemplos a definição da condição em vermelho. No procedimento "**MAXIMO**" tem-se a definição de uma condição de uma variável com outra variável, mas no procedimento "**PAR**" a condição é mais complexa, pois para saber se um valor é par é necessário validar o resto da divisão com a função **MODULO** e comparar este valor com a constante "**0**" (zero). Os trechos marcados em verde referem-se a ação se a condição for verdadeira e ocre se a condição for falsa.

Como exemplo de tomada de decisão simples considere o procedimento chamado "**IMPAR**" que apresenta a mensagem "Ok" se o valor numérico for par e não mostre nada caso contrário. Observe o código a seguir:

```
APRENDA IMPAR :N
IF (MODULO :N 2) <> 0 [ESCREVA [Ok]]
END
```

Em seguida execute separadamente as instruções:

```
IMPAR 3
IMPAR 2
```

Veja na figura 3.19 a apresentação do resultado das operações.

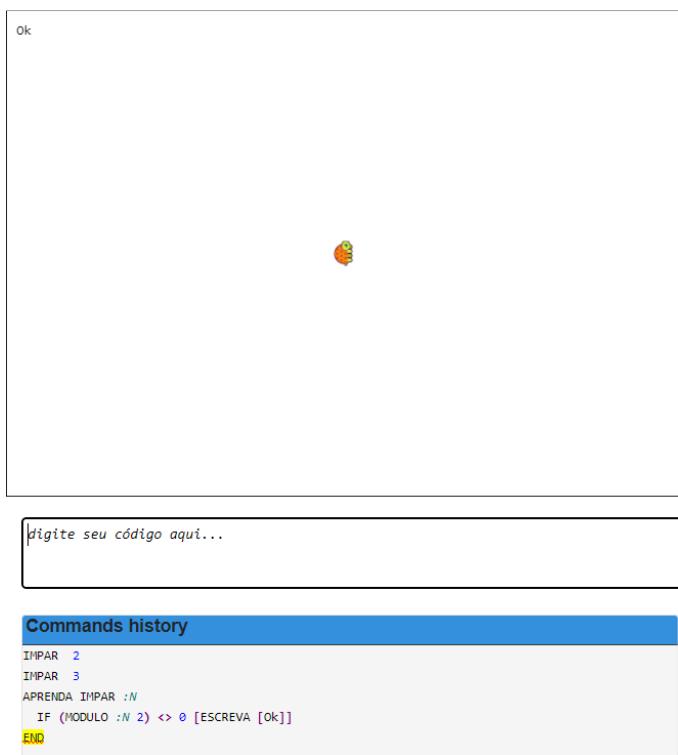


Figura 3.19 - Resultado do procedimento "IMPAR".

A da ação de tomada de decisão como exemplo um procedimento chamado "**FLORAL**" que aceite valores entre "1" e "7". Qualquer valor fora da faixa deve fazer com o procedimento mostrar a mensagem de erro "**Use valores entre 1 e 7**". Para tanto, escreva o código (não se preocupe com a primitiva **OR**, que traduzida significa *OU*, com sua explicação mais a frente):

```
APRENDA FLORAL :N
IFELSE OR :N < 1 :N > 7 [
  ESCREVA [Use valores entre 1 e 7]
][
  REPITA 8 [
    GD 45
    REPITA :N [
      REPITA 90 [
        PF 2
        GD 2
      ]
      GD 90
    ]
  ]
]
END
```

O procedimento "**FLORAL**" é uma adaptação de exemplo Logo encontrado num material de aula do Professor Carlos Eduardo Aguiar do Centro de Educação Superior a Distância do Estado do Rio de Janeiro: "<http://omnis.if.ufrj.br/~carlos/infoenci/notasdeaula/roteiros/aula10.pdf>".

Veja que no procedimento "**FLORAL**" a estrutura do código está sendo definida em outro estilo de escrita. Neste caso, baseando-a com o uso de endentações no sentido de indicar visivelmente quem está dentro de quem (observe as cores). Nada impede, por exemplo de escrever o procedimento "**FLORAL**" de outras maneiras, mas a forma apresenta é um estilo que deixa o emaranhado de instruções mais claras.

Ao término desta definição feche o programa **Editor** com "**Arquivo/Guardar e Sair**" ou acione as teclas "**<Ctrl> + <D>**". Em seguida execute separadamente as instruções:

TAT
FLORAL 0

TAT
FLORAL 1

TAT
FLORAL 2

TAT
FLORAL 3

TAT
FLORAL 4

TAT
FLORAL 5

TAT
FLORAL 6

TAT
FLORAL 7

TAT
FLORAL 8

Veja na figura 3.20 a apresentação dos vários resultados obtidos a partir dos valores numéricos entre **1** e **7** válidas para o procedimento "**FLORAL**".

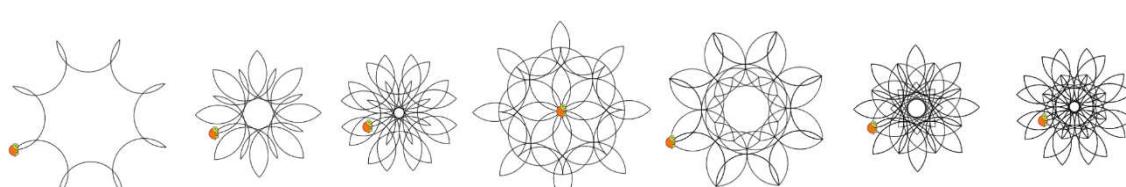


Figura 3.20 - Resultado do procedimento "FLORAL".

Além dos operadores relacionais são encontrados os operadores lógicos que auxiliam ações de tomada de decisão como é o caso do operador **OR** usado no procedimento "**FLORAL**". Além do operador **OR** existem os operadores lógicos **AND** (traduzido significa *E*) e **NOT** (traduzido significa *NÃO*). Atente para a tabela 3.3.

OPERADOR	SIGNIFICADO	RESULTADO
AND	CONJUNÇÃO	VERDADEIRO (1) quando todas as condições forem verdadeiras.
OR	DISJUNÇÃO	VERDADEIRO (1) quando pelo menos uma das condições for verdadeira.
NOT	NEGAÇÃO	VERDADEIRO (1) quando condição for falsa e FALSO (0) quando condição for verdadeira.

Tabela 3.3 - Operadores lógicos.

Os operadores lógicos **AND** e **OR** permitem vincular em uma tomada de decisão duas ou mais condições. Já o operador lógico **NOT** faz a inversão do resultado lógico da condição a sua frente. A ordem de prioridade entre os operadores lógicos é: **NOT**, **AND** e **OR**. Observe as tabelas verdadeiras a seguir para cada um dos operadores lógicos (**AND**, **OR** e **NOT**) e confronte o que é apresentado com o resumo descrito na tabela 3.3.

Operador de conjunção

A conjunção é a relação lógica entre duas ou mais condições que gera resultado lógico verdadeiro quando todas as proposições forem verdadeiras. A tabela 3.4 indica os resultados lógicos que são obtidos a partir do uso do operador lógico de conjunção "AND".

CONDIÇÃO 1	CONDIÇÃO 2	RESULTADO
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	FALSO

Tabela 3.4 - Tabela verdade para operador "AND".

Para demonstrar o uso do operador lógico de conjunção **AND** considere um procedimento chamado "**TESTE_E**" que receba um valor numérico e informe se este valor está ou não na faixa numérica entre "1" a "9". Caso o valor não esteja na faixa nada deve ser apresentado. Segue seu código:

```
APRENDA TESTE_E :N
IFELSE AND :N >= 1 :N <= 9 [
    ESCREVA [Valor está na faixa de 1 a 9.]
] [
    ESCREVA [Valor não está na faixa de 1 a 9.]
]
END
```

Em seguida execute separadamente as instruções a seguir:

```
TESTE_E 5
TESTE_E 0
TESTE_E 11
```

Procure fazer mais testes com outros valores.

Operador de disjunção

A disjunção é a relação lógica entre duas ou mais condições de tal modo que seu resultado lógico será verdadeiro quando pelo menos uma das proposições for verdadeira. A tabela 3.5 apresenta os resultados lógicos que são obtidos a partir do uso do operador lógico de conjunção "OR".

CONDIÇÃO 1	CONDIÇÃO 2	RESULTADO
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	FALSO

Tabela 3.5 - Tabela verdade para operador "OR".

Para demonstrar o uso do operador lógico de disjunção OR considere um procedimento chamado "**TESTE_OU**" que receba um valor textual "FRIO" ou "CHUVOSO" e informe respectivamente a mensagem "**Tempo ruim, proteja-se.**" e "**Tempo bom, aproveite.**". Escreva o código:

```
APRENDA TESTE_OU :TEMPO
IFELSE OR UPPERCASE :TEMPO = "FRIO UPPERCASE :TEMPO = "CHUVOSO [
    ESCREVA [Tempo ruim, proteja-se.]
][
    ESCREVA [Tempo bom, aproveite.]
]
END
```

Em seguida execute separadamente as instruções a seguir:

```
TESTE_OU "SOL
TESTE_OU "FRIO
TESTE_OU "QUENTE
TESTE_OU "CHUVOSO
```

Procure fazer mais testes com outros valores. Note o uso da função **UPPERCASE** (tradução para **MAIÚSCULO**) para garantir que não importando o formato do texto informado este será transformado em texto maiúsculo antes de ser verificado na condição.

OBS:

O operador lógico **OR** do ambiente "Academia da Tartaruga" possui um "bug" (erro de execução) que não verifica a segunda condição do operador. Desta forma, o procedimento "**TESTE_OU**" não funciona adequadamente para o teste de execução "**TESTE_OU "CHUVOSO"**" que deveria retornar a mensagem "*Tempo ruim, proteja-se*". A solução para este caso é o uso de três tomadas de decisão simples: uma verificando a condição "**:TEMPO = "FRIO"**", outra a condição "**:TEMPO = "CHUVOSO"**" e a terceira condição para tratar a mensagem "*Tempo bom, aproveite*".

Operador de negação

A negação é a rejeição ou a contradição do todo ou de parte de um todo. Pode ser a relação entre uma condição "**p**" e sua negação "**não-p**". Se "**p**" for verdadeira, "**não-p**" é falsa e se "**p**" for falsa, "**não-p**" é verdadeira. A tabela 3.6 mostra os resultados lógicos que são obtidos a partir do uso do operador lógico de negação "**NOT**".

CONDIÇÃO	RESULTADO
VERDADEIRO	FALSO
FALSO	VERDADEIRO

Tabela 3.6 - Tabela verdade para operador "NOT".

Para demonstrar o uso do operador lógico de negação **NOT** leve em consideração um procedimento "**TESTE_NAO**" que receba um valor numérico e mostra o quadrado deste valor caso este valor *não seja maior que "5"*. Escreva o código:

```
APRENDA TESTE_NAO :V
  IF NOT :V > 5 [
    ESCREVA POWER :V 2
  ]
END
```

Em seguida execute separadamente as instruções a seguir:

```
TESTE_NAO 3
TESTE_NAO 5
TESTE_NAO 7
```

Ao ser executado o procedimento serão apresentados apenas resultados para entradas menores ou iguais a 5 (que são a forma de respeitar a condição *valores não maiores que 5*).

3.6 - Recursão

A ação de recursão é um recurso que na linguagem Logo permite a um procedimento chamar a si mesmo certo número de vezes. Veja que, um procedimento recursivo não pode chamar a si mesmo infinitamente, pois se assim o fizer entrará em um processo infinito de chamadas sucessivas podendo interromper a ação operacional do computador como um todo no momento em que os recursos de memória se tornarem esgotados. É importante que o procedimento recursivo tenha a definição de uma condição de encerramento (*aterramento*) controlada com o comando **IF** e com o auxílio do comando **STOP** (traduzido como *PARE*).

A ação de aterramento deve ser definida antes da linha de código que efetua a recursão, pois ao contrário pode ocasionar o efeito colateral de execução infinita da recursividade inviabilizando seu uso (TOMIYAMA, 2016, p.2).

De modo prático um procedimento recursivo efetua ações de repetições semelhantemente as ações produzidas com o comando **REPITA**. A diferença é que o comando **REPITA** repete um trecho arbitrário de instruções e a recursão repete o próprio procedimento.

Para demonstrar um efeito de recursão considere um procedimento que produza a apresentação de imagens espiraladas. Vale ressaltar que uma espiral é o desenho de uma linha curva que

se desenrola num plano de modo regular a partir de um ponto, dele gradualmente afastando-se. Assim sendo, informe o seguinte código:

```
APRENDA_ESPIRAL :TAMANHO :ANGULO :NUMERO
IF :NUMERO = 0 [
    STOP
]
PF :TAMANHO
GD :ANGULO
ESPIRAL (:TAMANHO + 8) :ANGULO (:NUMERO - 1)
END
```

Ao término desta definição feche o programa **Editor** com "**Arquivo/Guardar e Sair**" ou acione as teclas "**<Ctrl> + <D>**". Em seguida execute separadamente as:

```
TAT_ESPIRAL 4 122 60
TAT_ESPIRAL 4 92 50
```

Veja na figura 3.21 a apresentação do resultado das operações.

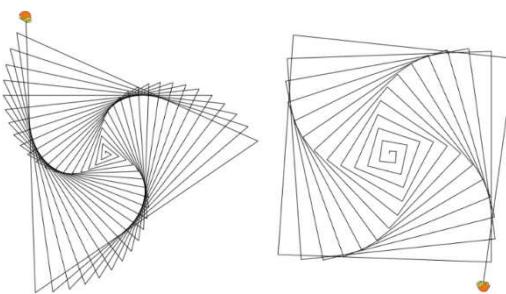


Figura 3.21 - Resultado do efeito recursivo sobre imagens espiraladas.

A partir do código do procedimento "**ESPIRAL**" note a definição em cor verde da decisão de aterramento que detecta, neste caso, se o valor do parâmetro (variável) **:NUMERO** é igual a "0" (zero) e sendo faz a parada da execução do procedimento com o comando **STOP**. Note que se este trecho não estiver definido a função **ESPIRAL** chamará a si mesma sucessivamente.

Enquanto o valor da variável **:NUMERO** é diferente de zero ocorre a execução do procedimento que a cada chamada a si mesmo aumenta o valor de **:TAMANHO** em "8" (oito), mantém constante a distorção do valor de **:ANGULO** e diminui em "1" (um) o valor de **:NUMERO**. A cada chamada uma parte da imagem é desenhada com as instruções **PF :TAMANHO** e **GD :ANGULO** definidas na cor ocre. Veja também que a soma e subtração de valores sobre o valor inicial dos parâmetros tem que ser definida entre parênteses.

O efeito espiralado para a imagem de um triângulo é conseguido com o valor de giro de ângulo "**122**" para o parâmetro **:ANGULO** e a imagem do quadrado é conseguido com o valor "**92**". Veja que esses valores são próximos ao valor real de giro para a apresentação das figuras geométricas planas.

O que aconteceria, por exemplo, se fossem usados os valores de ângulos "**120**" e "**90**" no procedimento "**ESPIRAL**"? Então, mãos à obra, execute separadamente as instruções:

```
TAT ESPIRAL 4 120 60  
TAT ESPIRAL 4 90 60
```

Veja os resultados na figura 3.22:

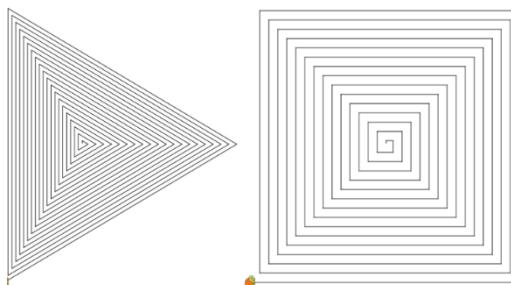


Figura 3.22 - Mais efeitos recursivos sobre imagens espiraladas.

No sentido de apresentar mais ações sobre recursões veja na próxima sequência de procedimentos os recursos para se desenhar uma treliça (unidade definida a partir de cinco triângulos) como base de sustentação para uma ponte (adaptado, MULLER, 1998 p. 345). Para exemplificar ação de *recursividade simples* considere o procedimento "FIB1" que apresenta o termo da sequência a partir da posição indicada. Na sequência, entre o seguinte código sem linhas em branco:

```
APRENDA TRIANGULAR  
PF 40 GD 135  
PF 40 / SQRT 2 GD 90  
PF 40 / SQRT 2 GD 135  
END
```

```
APRENDA TRELICA :X  
IF :X = 0 [STOP]  
REPITA 4 [TRIANGULAR PF 40 GD 90]  
GD 90 PF 40 GE 90  
TRELICA :X - 1  
END
```

Ao término desta definição feche o programa **Editor** com "**Arquivo/Guardar e Sair**" ou acione as teclas "**<Ctrl> + <D>**". Em seguida execute a instrução:

```
TAT TRELICA 7
```

Veja na figura 3.23 a apresentação do resultado do efeito de recursividade.

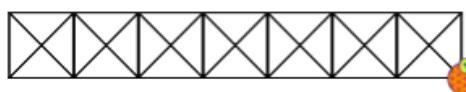


Figura 3.23 - Resultado do efeito de recursividade do procedimento "TRELICA".

Perceba que foram apresentados até este momento duas formas de se fazer a repetições de trechos de códigos. Uma com o comando **REPITA** e a outra com o efeito de recursividade. Se a recursividade for implementada com o cuidado da definição da condição de aterramento será um mecanismo útil de repetição.

3.7 - Exercícios de fixação

- Quais são as figuras geométricas planas desenhadas a partir das seguintes instruções? Diga qual é a figura sem executar a instrução no ambiente de programação.

REPITA 4 [PARAFRENTE 100 GIRADIREITA 90] _____

REPITA 5 [PF 100 GE 72] _____

REPITA 3 [PT 100 GD 120] _____

REPITA 36 [PF 20 GD 10] _____

- Criar procedimento chamado **RETANGULO1** (sem acento) que desenhe um retângulo com lados de tamanhos "60" e "100" para frente com giro de gruas para à direita. O procedimento deve desenhar a imagem sem o uso do recurso **REPITA**.
- Criar procedimento chamado **RETANGULO2** (sem acento) que desenhe um retângulo com lados de tamanhos "60" e "100" para trás com giro de gruas à esquerda. Usar a primitiva **REPITA**.
- Criar, sem uso da primitiva **REPITA**, procedimento chamado **PENTAGONO** (sem acento) que construa um pentágono com tamanho "40". Avance para frente com giro a direita.
- Criar procedimento chamado **DECAGONO** (sem acento) que construa uma figura com dez lados a partir do uso da primitiva **REPITA** com giro de graus à esquerda com avanço a gente de "30".
- Criar procedimento chamado **ICOSAGONO** (sem acento) que desenhe a figura de mesmo nome a partir do uso da primitiva **REPITA** com tamanho "35" movimentado para trás.

ANOTAÇÕES

CAPÍTULO 4 - Ações específicas

Anteriormente foram apresentados alguns recursos da linguagem que proporcionam experiências interessantes, mas ainda limitadas. A linguagem Logo é mais poderosa do que realizar a apresentação de imagens geométricas planas. Veja neste capítulo alguns outros recursos da linguagem.

4.1 - Entrada de dados

Anteriormente foram apresentados recursos para a armazenamento básico de valores na forma de variáveis quando do uso de parâmetros e a apresentação escrita de resultados, principalmente na área de ação do modo texto a partir do campo de entrada de comandos, dados e instruções.

No entanto, *Logo* permite a entrada de dados em tempo de execução. Para isso deve-se fazer uso da primitiva **READWORD** (traduzido como *LEIA PALAVRA*) que possui como estrutura sintática o estilo:

```
ATRIBUA "<variável> READWORD
```

Onde, o indicativo "**<variável>**" é a definição de uma variável para operação após a primitiva **ATRIBUA** associada ao uso de **READWORD** que abre uma caixa de diálogo aguardando a entrada de um dado. Para um teste rápido execute a instrução:

```
ATRIBUA "N READWORD
```

Veja que a primitiva **READWORD** deve seguir a instrução "**ATRIBUA "N"**" que quando executada apresenta uma caixa de diálogo para entrada de dados como indica a figura 4.1.

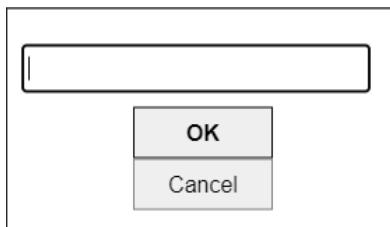


Figura 4.1 - Caixa de diálogo "Modo de Entrada".

Para fazer uma experiência no uso deste recurso considere o desenvolvimento de um procedimento chamado "**IDADE**" que solicite a entrada de uma idade, armazenando a idade na variável **:VALOR** e apresentando a mensagem "**Maior de idade**" se a idade for maior ou igual a "**18**" ou apresentando a mensagem "**Menor de idade**" caso a idade seja menor que "**18**". Assim sendo, entre o seguinte código:

```
APRENDA IDADE  
ATRIBUA "VALOR READWORD  
IF :VALOR >= 18 [ESCREVA [Maior de idade]]  
IF :VALOR < 18 [ESCREVA [Menor de idade]]  
END
```

Em seguida execute o procedimento "**IDADE**" algumas vezes e informe valores diferentes. Veja os resultados apresentados.

Anteriormente foi apresentado o comando **REPITA** que tem por finalidade repetir um trecho de código demarcado dentro de colchetes um determinado número de vezes. Há um comando que pode mostrar o momento da contagem em que o comando **REPITA** se encontra. Conheça a primitiva **CONTEVEZES**.

Para mostrar o conteúdo da primitiva **CONTEVEZES** além da primitiva **ESCREVA** é necessário usar em conjunto o comando **SENTENCE (SE)** que traduzido pode ser entendido como **SEN-TENÇA**, tendo por finalidade criar uma lista de caracteres concatenados a ser apresentada pelo comando **ESCREVA**. Assim sendo, considere apresentar os valores gerados pela execução do comando **REPITA** durante a ação de "5" iterações. A cada contagem de **REPITA** o valor na memória deve ser resgatado e apresentado. Para tanto, execute a instrução:

REPITA 5 [ESCREVA (SENTENCE "Conta: CONTEVEZES)]

Veja o resultado obtido junto a figura 4.2.

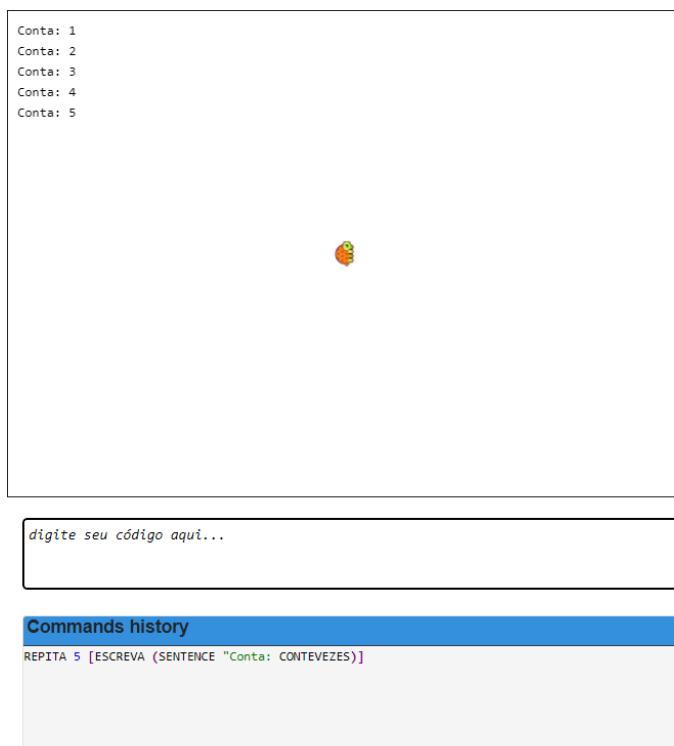


Figura 4.2 - Resultado da ação do comando "CONTEVEZES" junto a "FRASE" com "ESC".

Veja que a cada execução do comando **REPITA** o comando **CONTEVEZES** detectou um valor da contagem. A cada contagem o valor de **CONTEVEZES** foi concatenado a palavra "**Conta:**" por meio do comando **SENTENCE**. O resultado da tabuada a ser apresentado deve ser delimitado entre parênteses estabelecendo assim uma estrutura de lista.

Agora, considere um procedimento chamado "**TABUADA**" que apresente a tabuada de um número qualquer entre "2" e "9". Para auxiliar esta operação entra em jogo um novo comando chamado **CONTEVEZES** que tem por finalidade recuperar o momento de contagem do comando **REPITA**. Assim sendo, entre o seguinte código:

```

APRENDA TABUADA
ATRIBUA "NUMERO READWORD
IF AND :NUMERO >= 2 :NUMERO <= 9 [
  REPITA 10 [ESCREVA (
    SE :NUMERO "X CONTEVEZES "= :NUMERO * CONTEVEZES
  )]
]
END

```

Ao término, feche o programa **Editor** com "**Arquivo/Guardar e Sair**" ou acione as teclas "**<Ctrl> + <D>**". Em seguida execute separadamente a instrução:

TABUADA

Quando apresentada a caixa de diálogo para entrada, forneça um valor numérico entre "**2**" e "**9**". Por exemplo, entre o valor "**7**" e veja a tabuada apresentada como indicado na figura 4.3.

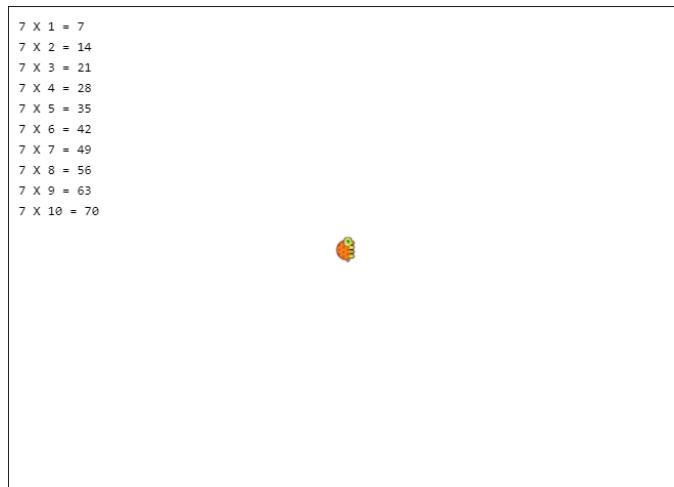


Figura 4.3 - Resultado de uma tabuada simples.

Perceba que Logo é uma linguagem que vai muito mais além do que simplesmente produzir lindas imagens e desenhos.

4.2 - Randomização

Um recurso que pode ser explorado em computadores é a capacidade destes conseguirem sortear valores, principalmente numéricos, de forma aleatória. Valores aleatórios também são chamados de *valores randômicos*. Daí o termo *randomização*.

A linguagem Logo possui para auxiliar operações de sorteio o comando **SORTEIENÚMERO** que é operado segundo a estrutura sintática:

SORTEIE <número>

Onde, o indicativo "**número**" refere-se ao valor máximo estabelecido para sorteio. Este valor poderá variar entre "**0**" (zero) e o valor estabelecido em "**número**" menos "**1**".

Veja algumas instruções de sorteio entre os valores "0" e "100":

```
ESCREVA SORTEIE 101  
ESCREVA SORTEIE 101  
ESCREVA SORTEIE 101  
ESCREVA SORTEIE 101  
ESCREVA SORTEIE 101
```

Veja que a cada execução o valor sorteado retornado é diferente um do outro como comprova a figura 4.4.

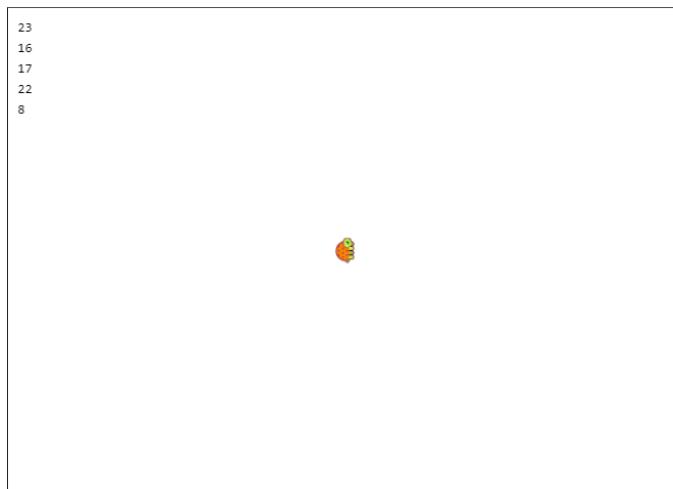


Figura 4.4 - Resultado do sorteio de valores entre "0" e "100".

A partir do recurso de randomização veja o procedimento "**MALUQUINHO**" que desenha figura geométrica desforme, pois avança para a direita aleatoriamente entre os valores "0" e "60" e gira para a esquerda aleatoriamente entre "0" e "360". Entre na sequência o seguinte código:

```
APRENDA MALUQUINHO  
REPITA 100 [  
  PF SORTEIE 61  
  GD SORTEIE 361  
]  
END
```

Em seguida execute a instrução:

```
TAT MALUQUINHO
```

Veja o resultado da operação na figura 4.5, ressaltando que em seu sistema a imagem tende a ser completamente diferente.

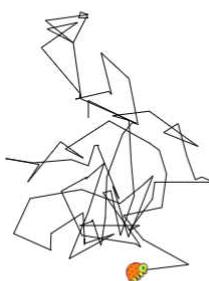


Figura 4.5 - Resultado de figura geométrica aleatória.

O comando **SORTEIE** sempre que é usado tende a gerar um valor aleatório diferente do anterior (pode ocorrer a repetição sequencial de dois valores, mas dificilmente três e quatro será virtualmente impossível), ou seja, se você carregar o ambiente "*Academia da Tartaruga*" na memória executar o comando **SORTEIE** e sair, carregar novamente o ambiente e executar novamente o comando **SORTEIE** terá valores sempre diferentes.

4.3 - Coordenadas

Todos as figuras geométricas planas apresentadas até este ponto foram desenhadas a partir do ponto central da área de ação do modo gráfico, ou seja, das coordenadas "0" para X e "0" para Y. O ponto central pode-se ser chamado de *CASA* por ser o ponto de início das operações geométricas em Logo.

Para verificar a posição em que a tartaruga se encontra usa-se o comando **POS** em conjunto com o comando **ESCREVA**. No entanto, encontram-se outros comandos de operação para auxiliar mudanças da tartaruga, tais como: **COORX** (mostra coordenada X), **COORY** (mostra coordenada Y), **MUDEX** (muda para coordenada X), **MUDEY** (muda para coordenada Y) e **HOME** (move tartaruga para o centro da tela - para casa).

Quando mudanças de posição são feitas dentro da área de ação do modo gráfico o rastro da tartaruga tende a ser traçado. Isso faz com que seja necessário antes de proceder mudança de posição usar o comando **USENADA**. Caso deseje traçar na mudança de coordenada use o comando **USELÁPIS**.

Para realizar alguns testes de posicionamento e verificação de posicionamento acione primeiro o comando **TAT** e em seguida execute as instruções:

```
USENADA  
ESCREVA POS  
MUDEX 50  
MUDEY -30  
ESCREVA POS
```

A figura 4.6 mostra os resultados apresentados. Veja os valores gerados após o uso da instrução **ESCREVA POS** na coordenada "0 0" com marcação em vermelho e na coordenada "50 -30" com marcação em azul. Em seguida use a instrução:

```
USELÁPIS
```

Para os comandos **MUDEX** e **MUDEY** os valores usados devem estar limitados a dimensão da área de trabalho, ou seja, valores definidos entre "0" e "500" (dentro do limite trabalhado neste livro). Veja que a partir da posição central (a *CASA*) é possível usar valores positivos para posicionar a tartaruga à direita ou acima da posição inicial ou usar valores negativos para posicionar a tartaruga à esquerda ou abaixo da posição inicial.

A Figura 4.7 representa esquematicamente os movimentos de salto com o par de comandos **MUDEX 30** com **MUDEY 20** e **MUDEX -30** com **MUDEY -10**. Na sequência, faça um novo carregamento da página da plataforma a partir do uso do comando de menu "**Diversão**".

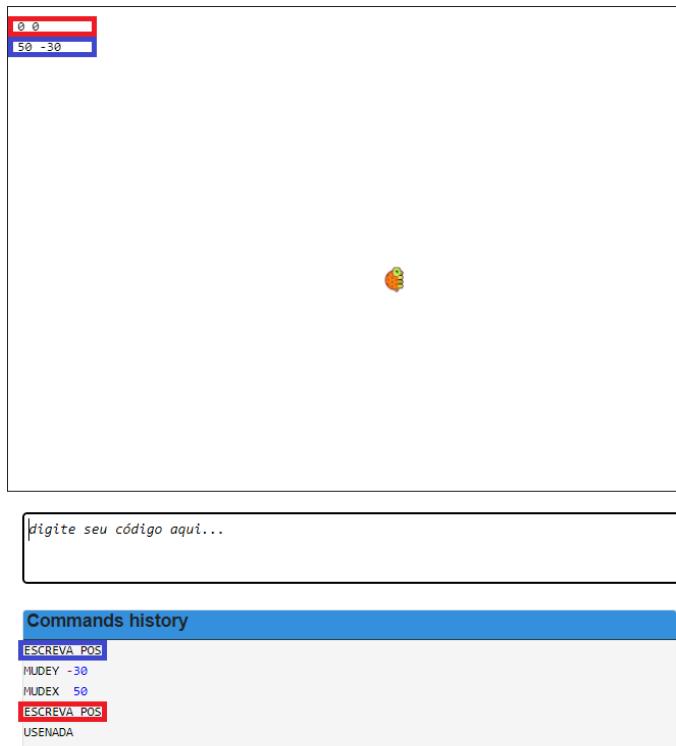


Figura 4.6 - Identificando posições cartesianas.

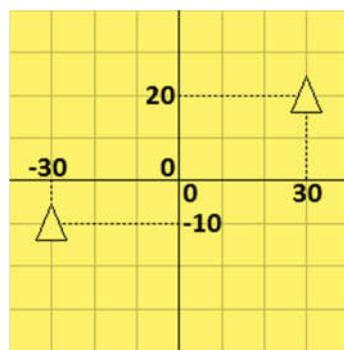


Figura 4.7 - Posições cartesianas da tartaruga.

O acesso as coordenadas X e Y podem ser realizadas a partir de uma única vez, por intermédio do comando **MUDEXY** e o retorno ao ponto central, a *CASA*, pode ser conseguido com o comando **CENTRO**. Teste as seguintes instruções:

```
TAT  
USENADA  
ESCREVA POS  
MUDEX -50  
MUDEY 25  
ESCREVA POS  
HOME  
ESCREVA POS  
USELÁPIS
```

Na sequência execute o comando **USELÁPIS**. A figura 4.8 mostra os valores das posições cartesianas após o uso dos comandos **MUDEX**, **MUDEY** e **CENTRO**.

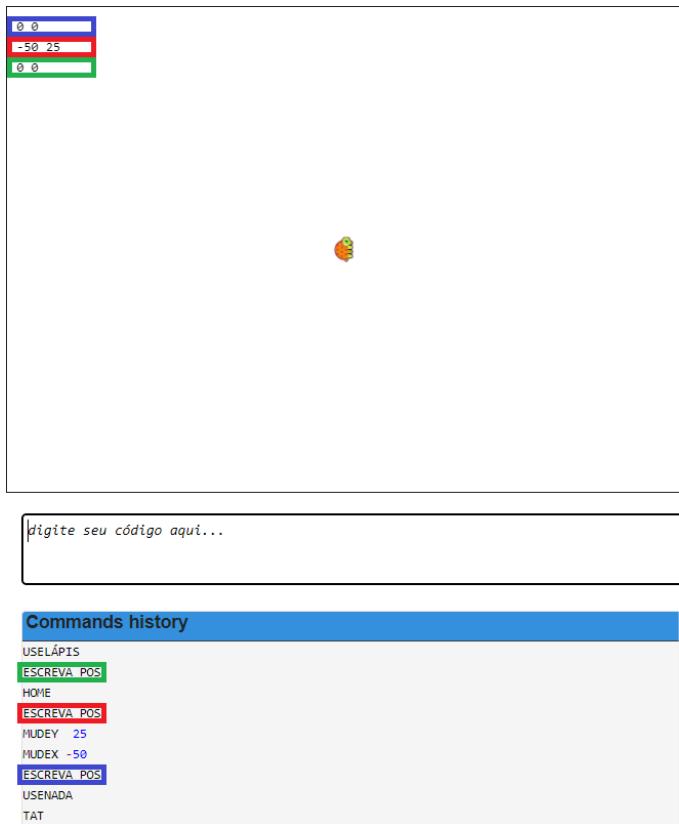


Figura 4.8 - Posições cartesianas após comandos "MUDEXY" e "CENTRO".

A partir desses "novos" recursos e de alguns recursos conhecidos é possível extrapolar e criar desenhos e imagens mais elaboradas, como por exemplo a figura da face quadrada indicada junto a figura 4.9, adaptada De Harvey (1997, p. 184).

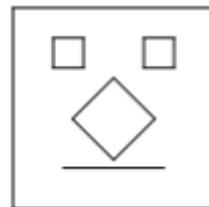


Figura 4.9 - Face quadrada.

Apesar de simples a imagem da figura 4.9 apresenta um grau de complexidade interessante, pois diversas ações são necessárias. Assim sendo, entre o seguinte conjunto de códigos:

```

APRENDA CABECA
  UL REPITA 4 [PF 100 GD 90]
  USENADA
END
APRENDA PREPARA.OLHO
  GE 90 PF 65 GD 90 PF 20
END
  
```

```

APRENDA OLHO
  REPITA 4 [PF 15 GD 90]
  END
APRENDA OLHO.ESQUERDO
  USELÁPIS
  OLHO
  UN PF 45
END
  
```

```
APRENDA OLHO.DIREITO  
USELÁPIS  
OLHO  
USENADA  
PT 15  
END
```

```
APRENDA BOCA  
USENADA  
PF 20 GD 90 PF 25  
USELÁPIS  
PF 50  
USENADA  
PT 75  
END
```

```
APRENDA NARIZ  
GD 90 PF 20 GE 45  
USELÁPIS  
REPITA 4 [PF 29 GD 90]  
USENADA  
HOME  
GE 135  
USELÁPIS  
END
```

```
APRENDA FACE  
CABECA  
BOCA  
PREPARA.OLHO  
OLHO.ESQUERDO  
OLHO.DIREITO  
NARIZ  
END
```

Em seguida execute separadamente as instruções após fazer uso do comando **TAT**:

```
OT  
FACE
```

Após executar o procedimento e visualizar a imagem da face quadrada execute o comando **MT** para reapresentar a tartaruga na área de ação do modo gráfico. Lembre-se que os comandos **OT** e **MT** são respectivamente as siglas dos comandos **OCULTETAT** e **MOSTRETAT**.

Às vezes é interessante fazer com que o ambiente Logo opere em velocidade menor do que trabalha, pois muitas vezes é interessante ver o desenho sendo vagarosamente formado. Para este tipo de necessidade há o comando **WAIT** (traduzido como *ESPERE*) que possui a estrutura sintática:

```
WAIT <tempo>
```

Onde, o indicativo "**tempo**" refere-se a um valor inteiro que determina valor em segundos. Por exemplo se quiser esperar *um segundo* use o valor "**60**", para *meio segundo* use "**30**" e assim sucessivamente. Veja então um exemplo mais sofisticado de criação de figura utilizando-se alguns dos comandos conhecidos. Atente para o código do procedimento **POLIGOM** operado respectivamente a partir dos parâmetros **:LADOS** e **:TAMANHO**:

```
APRENDA POLIGOM :LADOS :TAMANHO  
OT IF :LADOS = 30 [STOP]  
UL REPITA :LADOS [PF :TAMANHO GD 360 / :LADOS]  
UN HOME UL  
REPITA :LADOS [PF :TAMANHO GE 360 / :LADOS]  
WAIT 30  
POLIGOM (:LADOS + 1) :TAMANHO MT  
END
```

Em seguida execute separadamente as instruções após fazer uso do comando **TAT**:

POLIGOM 3 50

Veja o resultado da operação na figura 4.10. Atente para o efeito espelho conseguido com o posicionamento ao HOME após o desenho do lado direito para então desenhar o lado esquerdo. Atente para os movimentos dos efeitos dos comandos **USENADA** e **USELÁPIS** entre as imagens direita e esquerda.

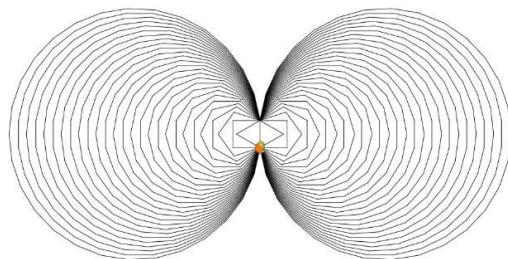


Figura 4.10 - Polígono com efeito de espelhamento.

4.4 - Cores

Os desenhos produzidos até este momento foram riscados em cor preta em um fundo branco. A linguagem Logo permite manipular o conjunto de cores do risco do desenho e do fundo da tela.

O uso de cores é obtido a partir da combinação do padrão de cores básicas, denominado **RGB** (**Red**, **Green** e **Blue**), onde cada cor a ser usada é a combinação do espectro de cores formado a partir das cores básicas definidas por meio de valores entre "0" e "255".

O sistema de cores **RGB** pode não ser intuitivo, mas é um sistema que possibilita a geração de **16.777.216** tons de cores diferentes, considerando-se os níveis de brilho e saturação dessas cores, mesmo que não seja possível ao olho humano distinguir tal espectro de cores. A tabela 4.1 mostra dezesseis tonalidades de cores básicas que podem ser usadas na "Academiada Tartaruga".

ÍNDICE	COR	CÓDIGO RGB
0	PRETO	[000 000 000]
1	AZUL	[000 000 255]
2	VERDE (LIMA)	[000 255 000]
3	CIANO (ÁGUA)	[000 255 255]
4	VERMELHO	[255 000 000]
5	MAGENTA (FÚCSIA)	[255 000 255]
6	AMARELO	[255 255 000]
7	BRANCO	[255 255 255]
8	MARROM	[155 096 059]
9	BRONZEADO	[197 136 018]
10	VRDE (OLIVA)	[100 162 064]
11	AZUL CELESTE	[120 187 187]
12	SALMÃO	[255 149 119]
13	ROXO (PÚRPURAMÉDIO)	[144 113 208]
14	LARANJA	[255 163 000]
15	CINCA CLARO	[183 183 183]

Tabela 4.1 - Índice de cores do ambiente Academia da Tartaruga.

Os valores das cores grafados entre colchetes na coluna "CÓDIGO RGB" da tabela 4.1 segue o formato posicional [R G B], em que cada componente é um valor numéricico entre "0" e "255".

A mudança de cores de fundo e do traço a ser riscado se faz com os comandos **PINTE** (que pode mudar a do fundo) e **MUDECL** (mude cor do lápis) com a indicação do padrão **RGB** da cor desejado ou pelo índice da cor indicado na tabela 4.1.

Para formatar com código **RGB** o fundo na cor preta e o lápis na cor amarelo use as instruções, após executar **TAT**:

```
PINTE MUDECL [000 000 000] PINTE  
MUDECL [255 255 000]
```

Para ver o resultado execute a instrução:

```
FACE
```

A figura 4.11 mostra o resultado com fundo preto e imagem em tom amarelo.

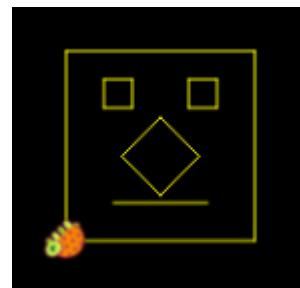


Figura 4.11 - Imagem no fundo preto em tom amarelo.

Para formatar com código **ÍNDICE** o fundo na cor azul e o lápis na cor azul celeste use as instruções, após executar **TAT**:

```
PINTE MUDECL 1 PINTE  
MUDECL 11
```

Para ver o resultado execute a instrução:

```
FACE
```

A figura 4.12 mostra o resultado com fundo azul e imagem em tom azul celeste.

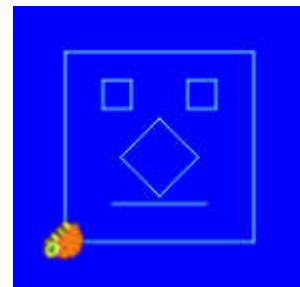


Figura 4.12 - Imagem no fundo azul em tom azul celeste.

OBS:

O uso de cores baseadas no **ÍNDICE** fica limitado aos valores de "0" a "15" indicados na tabela 4.1. Usar o padrão de cores a partir do formato [R G B] proporciona um leque maior de opções.

Para efetivar a mudança da cor de fundo é necessário fazer uso integrado as primitivas "**PINTE**" e "**MUDECL**" a partir da instrução "**PINTE MUDECL <valor> PINTE**", onde "**valor**" é a indicação do índice de cor ou a definição da lista **RGB**.

Para um teste do uso de cores considere o conjunto de procedimentos a seguir usados para desenhar um sol estilizado com o traço do lápis na cor laranja "[255 165 000]" em fundo na cor azul marinho "[000 000 128]" (adaptado de Abelson, 1981, p. 22). A Figura 4.13 apresenta a imagem gerada pela execução da chamada do procedimento "**SOL 10**".

Entre na sequência o conjunto de procedimentos seguintes:

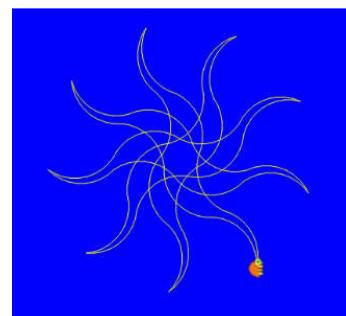


Figura 4.13 - Procedimento "SOL 10".

Atente para a marcação em vermelho definindo o trecho do estabelecimento de cor de fundo e em verde o trecho do estabelecimento da cor do lápis.

```

APRENDA ARCO1 :TAMANHO :GRAU
REPITA :GRAU / 10 [
  PARAFRENTE :TAMANHO
  GIRADIREITA 10
]
END

```

```

APRENDA RAIO :TAMANHO
REPITA 2 [
  ARCO2 :TAMANHO 90
  ARCO1 :TAMANHO 90
]
END

```

```

APRENDA ARCO2 :TAMANHO :GRAU
REPITA :GRAU / 10 [
  PARAFRENTE :TAMANHO
  GIRAESQUERDA 10
]
END

```

```

APRENDA SOL :TAMANHO
PINTE
MUDECL [0 0 128]
PINTE
MUDECL [255 165 0]
REPITA 9 [
  RAIO :TAMANHO
  GIRADIREITA 160
]
END

```

Em seguida execute as instruções **TAT** e "**SOL 10**".

Experimente executar os procedimentos "**ARCO1**", "**ARCO2**" e "**RAIO**" separadamente para ver o que cada um deles faz para compor o desenho do procedimento "**SOL**".

Para voltar a tela ao modo padrão execute a instrução:

TAT

Além das mudanças na definição das cores do fundo da tela e do traço é possível fazer o preenchimento de certa cor dentro de uma figura. O preenchimento com certa cor é realizado com o uso do comando **MUDECL** (mude cor do preenchimento). Além de estabelecer a cor com o comando **MUDECL** é necessário fazer uso do comando **PINTE** para que a cor escolhida preencha a área da figura.

O procedimento "**LARANJA**" faz a apresentação de um quadrado nesta cor.

A fim de efetuar uma experimentação do preenchimento interno de uma imagem com cores o procedimento seguinte desenhará um quadrado laranja. Desta forma, entre o seguinte código:

```

APRENDA LARANJA
REPITA 4 [PF 100 GD 90]
USENADA
SETXY 30 30
MUDECL 14
PINTE
SETXY 0 0
MUDECL [0 0 0]
USELÁPIS
END

```

Observe que no procedimento após a apresentação do quadrado se faz o recolhimento do lápis com o comando **USENADA**. Em seguida faz-se a movimentação da tartaruga para a coordenada

"30" e "30" com o uso da primitiva **SETXY**, estabelece-se a cor "14" (laranja) com o comando **MUDECL** e efetua-se a pintura da imagem com o comando **PINTE**. Após a pintura da imagem faz-se com que a tartaruga volte para sua posição "CASA" com a instrução "**SETXY 0 0**", lembrando que poderá para esta ação ser usado o comando **HOME** e configura-se o lápis com tom de cor preto. Na sequência faz-se a ativação do lápis com o comando **USELÁPIS**.

Em seguida execute as instruções **TAT** e **LARANJA** e veja a imagem apresentada semelhante à figura 4.14.

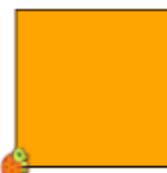


Figura 4.14 - Imagem com preenchimento.

4.5 - Fractais

Fractal é uma estrutura geométrica de forma complexa não regular que possui normalmente propriedades que se repetem em diversas escalas. A linguagem Logo se caracteriza em ser uma ferramenta adequada para a geração dessas formas geométricas.

Não é objetivo deste tópico, explorar este tema com profundidade, o objetivo é apenas e tão somente demonstrar a utilização da linguagem nesta tarefa operacional. Para tanto, considere a execução do procedimento "**PONTA**" o qual mostrará a imagem indicada na figura 4.15 a partir da chamada "**PONTA 30**".

```
APRENDA PONTA :LADO
  PF :LADO GE 60
  PF :LADO GD 120
  PF :LADO GE 60
  PF :LADO
END
```



Figura 4.15 - Imagem do procedimento "PONTA".

A partir do procedimento "**PONTA**" considere o procedimento "**FRACTAL1**" a seguir que executa o procedimento ponta seis vezes, como mostra a figura 4.16 após a chamada "**FRACTAL1 30**".

```
APRENDA FRACTAL1 :LADO
  REPITA 6 [
    GD 120
    PONTA :LADO
    GE 60
  ]
END
```

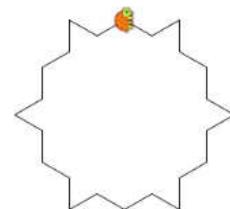


Figura 4.16 - Imagem do procedimento "FRACTAL1".

Note que com os procedimentos "**PONTA**" e "**FRACTAL1**" desenvolvidos foi efetuada uma imagem geométrica irregular (Figura 4.17) a partir de seis reexecuções da imagem gerada pelo procedimento "**PONTA**" (Figura 4.16).

A partir da imagem proposta pelo procedimento "**PONTA**" é possível formar outras imagens baseadas em fractais. Observe por exemplo o procedimento "**FRACTAL2**" em que se estabelecem mudanças na direção de deslocamento da tartaruga a partir dos comandos **GD** e **GE** dentro

do laço gerenciado GElo comando **REPITA**, como indicado na figura 4.17. Use a chamada de instrução "**FRACTAL2 30**".

```
APRENDA FRACTAL2 :LADO
  REPITA 6 [
    GE 120
    PONTA :LADO
    GD 60
  ]
END
```

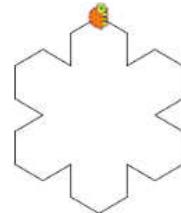


Figura 4.17 - Imagem do procedimento "FRACTAL2".

É possível a partir da execução do procedimento "**PONTA**" fazer a apresentação de formas fractais com formato mais complexo. Observe o procedimento "**FRACTAL3**" que mostra um trecho de imagem fractal bem tradicional, como indicado na figura 4.18. Use "**FRACTAL3 30**".

```
APRENDA FRACTAL3 :LADO
  GD 90
  REPITA 2 [
    PONTA :LADO
    GE 60
    PONTA :LADO
    GD 120
  ]
END
```

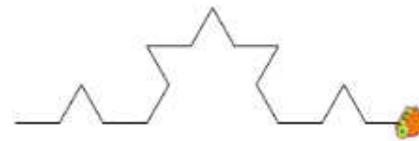


Figura 4.18 - Imagem do procedimento "FRACTAL3".

O efeito gerado GElo procedimento "**FRACTAL3**" faz uso do procedimento "**PONTA**" assim como também fizeram os procedimentos "**FRACTAL1**" e "**FRACTAL2**". A diferença está no posicionamento da tartaruga para a continuidade do desenho. Neste caso, faz-se o deslocamento de direção da tartaruga após desenhar "**PONTA**" GEla primeira vez "**60**" graus para a esquerda, faz novo desenho de "**PONTA**" e gira "**120**" graus para a direita.

A partir do procedimento "**FRACTAL3**" pode-se extrapolar a criação de fractais com base no procedimento "**PONTA**". Observe o procedimento "**FRACTAL4**" que gera a imagem indicada na figura 4.19 a partir da execução de "**FRACTAL4 30**" que mostra o fractal de *von Kock* (floco de neve).

```
APRENDA FRACTAL4 :LADO
  REPITA 4 [
    FRACTAL3 :LADO
    GE 120
    GD 30
  ]
END
```

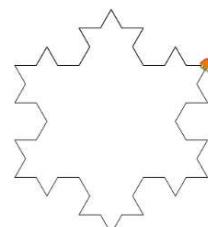


Figura 4.19 - Imagem do procedimento "FRACTAL4".

4.6 - Outras repetições

Além das repetições produzidas com o comando **REPITA** e as operações de recursão Logo possui outras primitivas para a realização de operações baseadas em repetições de trechos de códigos, como: **WHILE** (traduzido como *ENQUANTO*) e **FAÇAPARA**.

O comando **WHILE** (laço condicional pré-teste com fluxo verdadeiro) realiza repetições de trechos de código de forma condicional sem o uso de recursividade enquanto a condição de verificação permanece verdadeira. No momento em que a condição torna-se falsa o laço é automaticamente encerrado. Isso é útil em momentos que se necessite de laço interativo onde não se conhece antecipadamente o número de repetições.

Para verificar o efeito de funcionamento do comando **WHILE** considere o procedimento "**CONTAGEM1**" que apresenta os valores de contagem entre "1" e "5" de "1" em "1". Assim sendo, entre o seguinte código:

APRENDA CONTAGEM1

```
ATRIBUA "I 1
WHILE [:I <= 5] [
    ESCREVA :I
    ATRIBUA "I :I + 1
]
END
```

Ao término, feche o programa **Editor** com "**Arquivo/Guardar e Sair**" ou acione as teclas "**<Ctrl> + <D>**". Em seguida execute as instruções **TAT**, **LJC** e "**CONTAGEM1**" e veja a imagem apresentada semelhante à figura 4.20.

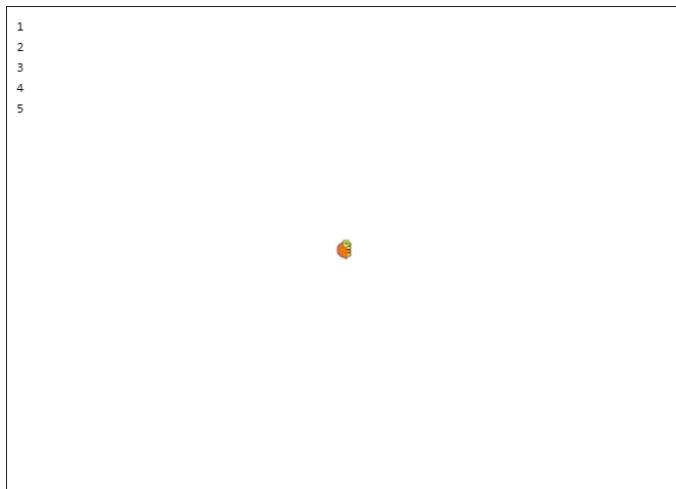


Figura 4.20 - Apresentação da contagem de "1" a "5" com "ENQUANTO".

Veja que para o comando **WHILE** funcionar a condição deve permanecer verdadeira por certo tempo, pois a repetição somente é encerrada quando a condição se torna falsa.

No procedimento "**CONTAGEM1**" a variável "I" é iniciada com o "1" e cada escrita é somada a ela mais "1 ("I :I + 1)" fazendo que a variável atinja um valor que fará a condição "[:I <= 5]" se tornar falsa encerrando as repetições. Atente para os pontos colorizados em vermelho e verde exemplificados anteriormente.

Veja o procedimento "**PENTAGRAMA**" que mostra uma estrela de cinco pontas gerada a partir das primitivas **WHILE** e **MUDEDIREÇÃO (MUDED)**. A primitiva **MUDED** pode ser usada como substituta da diretiva **GD**. Para tanto, entre o seguinte código:

APRENDA PENTAGRAMA

```

MUDADIREÇÃO 18
ATRIBUA "I 1
WHILE [:I <= 5] [
  PF 100
  GD 144
  ATRIBUA "I :I + 1
]
END

```

Em seguida execute as instruções **TAT** e "**PENTAGRAMA**" e veja a imagem apresentada semelhante à figura 4.21.



Figura 4.21 - Apresentação de imagem gerada com comando "ENQUANTO".

Os trechos marcados em vermelho mostram a estrutura a ser usada na definição de repetições iterativas que venham a substituir o uso do comando **REPITA**. Usas as atribuições de inicialização ("I 1") e incremento ("I :I + 1") são obrigatórias.

Para se fazer o desenho da estrela de cinco pontas é necessário usar um ângulo de "144" graus como apresentado. Talvez a pergunta em sua mente, seja: como saberei esse valor de grau? A resposta é basicamente simples. Divida o valor de graus máximos que é "360" por "5" e obter-se-á o valor "72" que é o ângulo interno de uma figura geométrica, multiplique o valor "72" por "2" e obter-se-á o valor "144" que é o valor do ângulo externo considerado pela linguagem Logo para a geração das imagens de figuras geométricas.

A primitiva **FAÇAPARA** é semelhante a **REPITA** por realizar ações iterativas. Para ver o funcionamento de **FAÇAPARA** considere o procedimento "**CONTAGEM3**" que mostra os valores de contagem entre "1" e "5" de "1" em "1". Assim sendo, entre o seguinte código:

APRENDA CONTAGEM2

```

FAÇAPARA [ I 1 5 1] [
  ESCREVA :I
]
END

```

Note que a primitiva **FAÇAPARA** usa a definição de uma variável e a definição de três valores: o primeiro valor determina o início da contagem, o segundo valor determina o fim da contagem e o terceiro valor determina o incremento da contagem, que pode ser omitido quando o incremento é de "1" em "1".

Em seguida execute as instruções **TAT** e "**CONTAGEM2**".

Já que foi feita uma estrela de cinco pontas (pentagrama) com o comando **WHILE** será feita uma estrela de quarenta pontas com o comando **FAÇAPARA**. Para tanto, definida o procedimento "**ESTRELA40**". Assim sendo, entre o seguinte código:

APRENDA ESTRELA40

```
FAÇAPARA [I 1 40] [  
    PT 200  
    GE 170  
]  
END
```

Em seguida execute as instruções **TAT** e "**ESTRELA40**" e veja a imagem apresentada semelhante à figura 4.22.

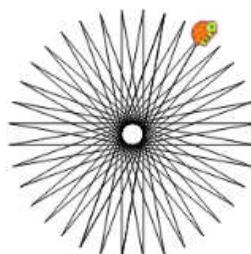


Figura 4.22 - Apresentação de imagem gerada com comando "PARA".

Os comandos: **REPITA**, **WHILE** e **FAÇAPARA** são recursos complementares e não necessariamente substitutos um dos outros. Esses comandos podem ser trabalhados juntos dentro de um mesmo projeto.

Os comandos podem ser usados sequencialmente ou encadeados. Sequencialmente quando um é definido após o outro e encadeado quando um é definido dentro do outro. Veja o procedimento seguinte chamado "**RODAFLOR**" que faz uso dos comandos **REPITA**, **WHILE** e **FAÇAPARA** no mesmo projeto na forma sequencial e encadeada. Assim sendo, entre o seguinte código:

APRENDA RODAFLOR

```
FAÇAPARA [I 1 6] [  
    ATRIBUA "I 1  
    WHILE [:I <= 120] [  
        PF 1 GD 1  
        ATRIBUA "I :I + 1  
    ]  
    GD 60  
    REPITA 180 [  
        PF 1 GD 1  
    ]  
    GD 60  
]  
END
```

Em seguida execute as instruções **TAT** e "**RODAFLOR**" e veja a imagem apresentada semelhante à figura 4.23.

O comando **FAÇAPARA** pode ser um substituto ao comando **REPITA** exatamente por operar na mesma categoria de repetições, uma vez que é usado para



Figura 4.23 - imagem gerada com repetições sequenciais e encadeadas.

repetições iterativas. No entanto, possui duas características particulares que o diferencia do comando **REPITA**.

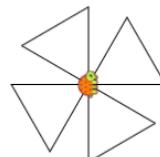
O comando **REPITA** usa para identificar o valor da contagem o apoio do comando **CONTEVEZES**. Com o comando **FAÇAPARA** é necessário fazer a definição de uma variável para a contagem e definir seus valores de início, fim e incremento quando diferente de "1". No entanto, o comando **FAÇAPARA** apresenta maior mobilidade que **REPITA** pois permite definir valores de contagem variados enquanto que **REPITA** sempre faz a contagem de "1" em "1" começando em "1" até o valor limite a ele estabelecido.

OBS:

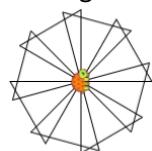
O ambiente "*FMSLogo*" executa os procedimentos desenvolvidos de maneira rápida. Apesar dessa rapidez ser importante no processamento dos dados elaborados por computadores, ela pode atrapalhar um pouco a percepção de como os desenhos são efetivamente feitos. Desta forma, pode-se pedir para que o ambiente opere em velocidade menor com o uso da primitiva **ESPERE** após riscar um traço. Por exemplo: "**REPITA 4 [PF 120 ESPERE 60 GD 90]**". O valor "60" corresponde a "1" segundo.

4.7 - Exercícios de fixação

1. Crie um procedimento chamado **TRIANGEX**, que apresente um triângulo equilátero com lado de tamanho "70" para trás a partir de giros a esquerda.
2. Criar procedimento chamado **FLORTRIG** desenhado a partir do procedimento **TRIANGEX** com giro a direita de modo que tenha a seguinte aparência.



3. Criar procedimento chamado **VENTITRIG** desenhado a partir do procedimento **TRIANGEX** com giro a esquerda de modo que tenha a seguinte aparência.

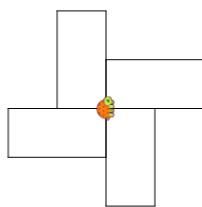


4. Sem executar no computador descrimine qual imagem é apresentada.

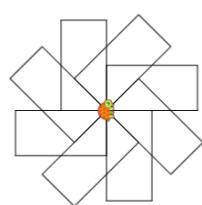
REPITA 12 [REPITA 3 [PF 50 GE 120] GE 30]

-
-
5. Criar procedimento chamado **RETANGULO3** com tamanhos "100" (altura) e "50" (largura). Movimente-se para frente com giro a direita.

6. Criar procedimento chamado **CATAVENTO1** com o formato da figura seguinte a partir do procedimento **RETANGULO3**.



7. Criar procedimento chamado **TRIANGPR** que desenhe um triângulo equilátero com tamanho definido por parâmetro com comando **FAÇAPARA** contando de "0" a "2" no sentido a frente com giro a direita.
8. Criar procedimento chamado **CATAVENTO2** com o formato da figura seguinte a partir do procedimento **RETANGULO3**.



9. Descubra sem o uso do computador qual é a imagem:

```
APRENDA QUADRO :TAMANHO
  REPITA 4 [
    PF :TAMANHO
    GD 90
  ]
  GD 45
  PF :TAMANHO * 7 / 5
  PT :TAMANHO * 7 / 5
  GE 45
  PF :TAMANHO
  GD 135
  PF :TAMANHO * 7 / 5
  PT :TAMANHO * 7 / 5
END
```

CAPÍTULO 5 - Ações complementares

A linguagem Logo vai além da possibilidade de apenas fazer belas figuras. Logo é uma linguagem como outra qualquer e possibilita muitos recursos para o desenvolvimento de programação lógica, funcional e estruturada. Este capítulo distancia-se do mundo gráfico da tartaruga e mostra a linguagem em uma outra dimensão.

5.1 - Funções matemáticas

Anteriormente foram comentadas e apresentadas algumas funções matemáticas da linguagem, como: **ABS**, **INT**, **MODULO**, **POWER**, **PRODUCT**, **QUOTIENT**, **SQRT**, **SUM** e **SORTEIE**, sendo essas algumas das funções existentes. A linguagem Logo possui as funções matemáticas: **ARCCOS**, **ARCCOSRAD**, **ARCSEN**, **ARCSENRAD**, **ARCTAN**, **ARCTANRAD**, **ARREDONDE**, **COS**, **COSRAD**, **EXP**, **LN**, **LOG10**, **PI**, **SEN**, **SENRAD**, **TAN** e **TANRAD**, indicadas na tabela 5.1.

FUNÇÃO	OPERAÇÃO
COS	Calcula o cosseno de um ângulo medido em graus.
EXP	Calcula o exponencial de um número
LN	Calcula o logaritmo natural (base "e") de número.
LOG10	Calcula o logaritmo na base 10 de número.
SIN	Calcula o seno do ângulo medido em graus.
TAN	Calcula a tangente do ângulo medido em graus.

Tabela 5.1 - Funções matemáticas e suas ações de ângulos.

Veja a seguir alguns exemplos de apresentação de resultados gerados a partir do uso das funções matemáticas apresentadas.

ESCREVA COS 45
ESCREVA EXP 1
ESCREVA LN (EXP 2)
ESCREVA LOG10 100
ESCREVA SIN 90
ESCREVA TAN 90

Veja os resultados gerados a partir das funções matemáticas anteriores na figura 5.1.

0.7071067811865476
2.718281828459045
2
2
1
16331239353195370

Figura 5.1 - Apresentação dos resultados no uso de funções matemáticas.

É sabido que um computador efetua três ações essenciais: entrada de dados, processamento de dados e saída de dados na forma de informação. As operações de entrada na linguagem Logo podem ser realizadas com o uso de parâmetros junto aos procedimentos ou a partir da primitiva **READWORD**; já as operações de saída são produzidas pelas primitivas **ESCREVA** ou **ROTULE**. Em relação ao processamento este evento pode ocorrer em duas dimensões que se misturam: uma dimensão matemática e outra dimensão lógica.

As funções matemáticas e os operadores aritméticos são responsáveis por atenderem a necessidade de execução de cálculos aritméticos na resolução de problemas matemáticos. As funções matemáticas podem ser utilizadas no desenvolvimento de figuras, especialmente as funções trigonométricas. Veja um exemplo de imagem com o uso da função **SIN**:

```
REPITA 360 [MUDEXY (SIN(2 * CONTEVEZES)) * 150 (SIN(3 * CONTEVEZES)) * 100]
```

A figura 5.2 mostra o resultado da execução da instrução com uso da função **SIN**.

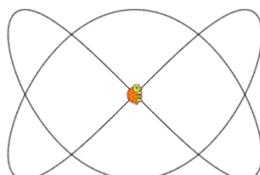


Figura 5.2 - Imagem obtida a partir do uso da função "SEN".

5.2 - Funções lógicas

Além das funções matemáticas, temos um conjunto de funções lógicas que auxiliam diversas ações de processamento. Funções do tipo lógicas são recursos que retornam como resposta de sua operação os valores **1** (verdadeiro) ou **0** (FALSO) sendo ótimas no estabelecimento de condições para as tomadas de decisão. Veja algumas funções lógicas encontradas na linguagem Logo: **BEFORE?, GREATER?, MEMBER?, NUMBER?, LESS? e EQUAL?**, indicadas na tabela 5.2.

FUNÇÃO	OPERAÇÃO
BEFORE?	Retorna 1 se parâmetro 1 vem à frente de parâmetro 2.
GREATER?	Retorna 1 se parâmetro 1 é maior que parâmetro 2.
MEMBER?	Retorna 1 se parâmetro 1 for membro de parâmetro 2.
NUMBER?	Retorna 1 se parâmetro for valor numérico.
LESS?	Retorna 1 se parâmetro 1 é menor que parâmetro 2.
EQUAL?	Retorna 1 se parâmetros forem iguais.

Tabela 5.2 - Funções lógicas (algumas).

Veja a seguir alguns exemplos de apresentação de resultados gerados a partir do uso das funções lógicas apresentadas.

```
ESCREVA BEFORE? "ABC "ABD
ESCREVA BEFORE? "XYZ "ABC
ESCREVA GREATER? 5 4
ESCREVA GREATER? 4 5
ESCREVA MEMBER? 1 [1 2 3 4 5]
ESCREVA MEMBER? 7 [1 2 3 4 5]
ESCREVA MEMBER? 7 "678
ESCREVA MEMBER? 7 [678]
ESCREVA LESS? 4 5
ESCREVA LESS? 5 4
ESCREVA EQUAL? 2 2
ESCREVA EQUAL? 1 2
ESCREVA EQUAL? [1 2 3] [1 2 3]
ESCREVA EQUAL? [1 2 3] [3 2 1]
```

Veja os resultados gerados a partir das funções comentadas na figura 5.3.

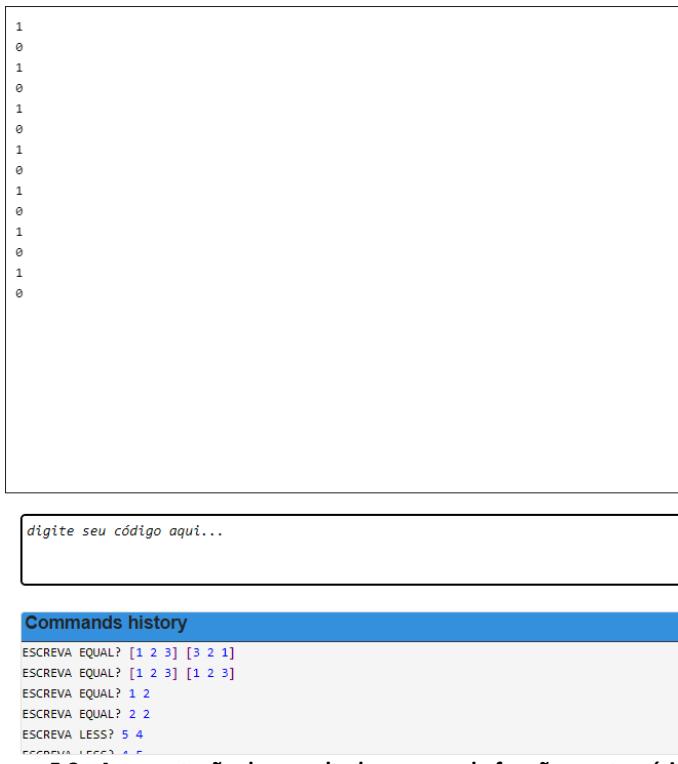


Figura 5.3 - Apresentação dos resultados no uso de funções matemáticas.

As funções lógicas são úteis no estabelecimento de controles em tomadas de decisão e execução de laços condicionais exatamente por retornarem como resposta a sua ação um valor **VERD** ou **FALSO**. Essas funções podem ser usadas para auxiliar diversas ações da linguagem seja no estabelecimento de operações matemáticas ou na elaboração de figuras.

As operações de processamento lógico são efetivadas a partir do uso dos operadores relacionais, operadores lógicos e funções especializadas.

5.3 - Algumas funções complementares

O conjunto de funções na linguagem Logo é extenso e torna-se inadequado exemplificar todas elas em um livro, uma vez que o ambiente "FMSLogo" possui boa documentação em português de todos os recursos disponíveis na linguagem. Veja algumas funções complementares que não foram anteriormente qualificadas, sendo: **NAME?** (função lógica) **WORD**, **FIRST**, **LAST** e **THING**. Observe a tabela 5.3.

FUNÇÃO	OPERAÇÃO
NAME?	Retorna 1 se conteúdo for o nome de uma variável.
WORD	Concatena palavras formando outra palavra.
FIRST	Retorna o primeiro elemento de uma série de valores.
LAST	Retorna o último elemento de uma série de valores.
THING	Retorna o conteúdo associado a uma variável.

Tabela 5.3 - Funções complementares (algumas).

As funções **WORD**, **FIRST** e **LAST** podem ser demonstradas em uma linha de instrução como foram demonstradas as funções matemáticas e lógicas. Veja a seguir alguns exemplos e atente ao uso das chaves com a função **FIRST** e colchetes com a função **LAST** quando usadas na manipulação de conjuntos de elementos.

```
ESCREVA WORD "Lo "go  
ESCREVA WORD "FMS "Logo  
ESCREVA FIRST [1 2 3 4 5]  
ESCREVA FIRST "Logo  
ESCREVA FIRST [[L][o][g][o]]  
ESCREVA LAST [1 2 3 4 5]  
ESCREVA LAST "Computador  
ESCREVA LAST [[L][o][g][o]]  
ESCREVA LAST [A B C D E]
```

Veja os resultados gerados a partir das funções comentadas na figura 5.4.

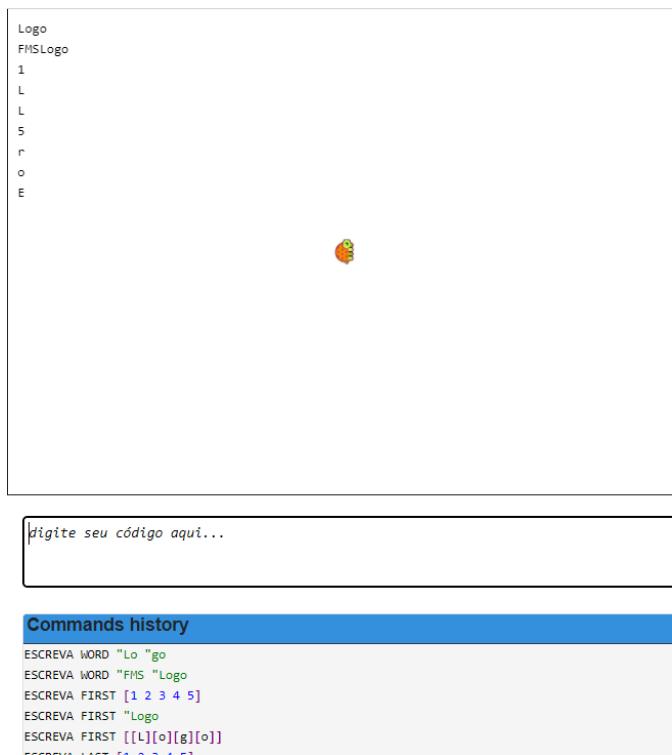


Figura 5.4 - Apresentação dos resultados no uso de funções auxiliares.

As funções **NAME?**, **KEY?** e **THING** para serem demonstradas necessitam de mais passos que as demais funções. Por esta razão estão sendo demonstradas em separado.

A função **THING** opera sua ação sobre variáveis. Desta forma, é necessário que haja na memória definição de variáveis que possam ser utilizadas pela função. Assim sendo, considere o seguinte trecho de instruções (ATARI, 1983, p. 76) que definem as variáveis de memória:

```
ATRIBUA "VENCEDOR "COMPUTADOR  
ATRIBUA "COMPUTADOR [100 PONTOS]
```

Observe um detalhe que pode passar desapercebido. Veja que na primeira instrução são definidas duas variáveis: **VENCEDOR** e **COMPUTADOR**. Veja que a variável **COMPUTADOR** ao ser definida após a variável **VENCEDOR** tornou-se conteúdo da variável **VENCEDOR**.

A partir dessa ocorrência, perceba, que o conteúdo da variável **COMPUTADOR** está vinculado a variável **VENCEDOR**. O acesso ao conteúdo vinculado entre as variáveis é realizado com o uso da função **VALOR**. Veja as próximas instruções:

```
ESCREVA :VENCEDOR
ESCREVA THING :VENCEDOR
ESCREVA THING "VENCEDOR
```

Veja os resultados gerados com o uso da função **VALOR** indicados na figura 5.5. Atente a diferença de uso dos símbolos [:] dois pontos e [""] aspa inglesa no acesso ao conteúdo da variável indicada.

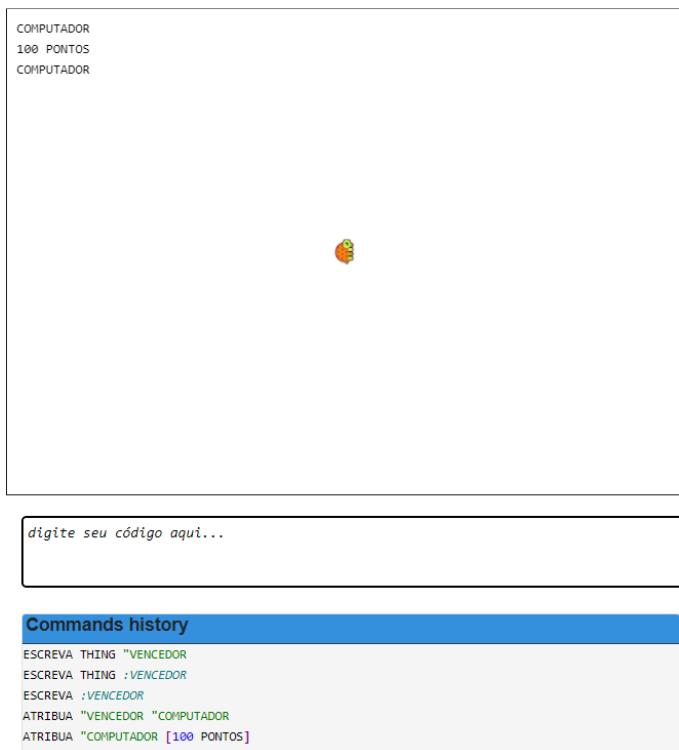


Figura 5.5 - Apresentação de resultados com função "VALOR".

Observe na sequência o procedimento "**INC**" que acrescenta o valor "+ 1" a uma variável indicada e faz uso da função **NAME?**. Assim, entre o seguinte código:

```
APRENDA INC :X
IF NOT NAME? :X [STOP]
IF NUMBER? THING :X [ATRIBUA :X 1 + THING :X]
END
```

Execute na sequência as instruções a seguir e veja o resultado indicado na figura 5.6.

```
ATRIBUA "TOTAL 1
ESCREVA :TOTAL
INC "TOTAL
ESCREVA :TOTAL
INC "TOTAL
ESCREVA :TOTAL
```

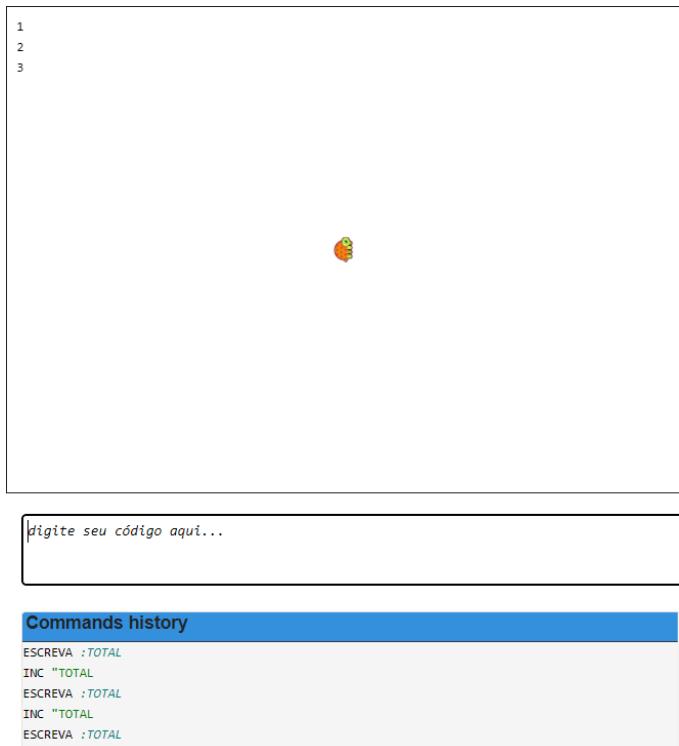


Figura 5.6 - Apresentação de resultados com função "VALOR" em procedimento.

Observe a evolução dos valores de "1" a "3" nos passos indicados. Note que o procedimento "**INC**" recebe um parâmetro (seu conteúdo) representado pela variável **X**.

Veja que a primeira instrução do procedimento "**INC**" é verificar se o conteúdo passado para o argumento **X** é de fato uma variável. Isso é detectado pela função **NOME?** com auxílio do operador lógico de negação **NOT**, ou seja, não sendo o nome fornecido uma variável **STOP**. A função **NAME?** retorna o valor **1** se a condição for satisfeita ou **0** caso contrário.

A segunda instrução do procedimento "**INC**" valida se o **THING** da variável **X** informado é de fato um número (**ÉNÚMERO?**), sendo efetue a atribuição (**ATRIBUA**) de "**+ 1**" sobre o conteúdo do **THING** armazenado na memória a partir da variável informada em **X**.

5.4 - Programação sem figuras

Logo é uma linguagem que possui muito mais do que aparentemente mostra. É possível desenvolver várias categorias de aplicação com Logo, mesmo não sendo usada no universo comercial. As aplicações Logo são usadas mais intensamente no universo educacional e podem ser amplamente usadas industrialmente junto a controle de robôs em produção crítica que colocam a vida humana em risco.

A primeira versão da linguagem foi escrita em computadores que não são usados domesticamente, ou seja, computadores hoje chamados de grande porte. Sua interface de acesso para crianças pré-alfabetizadas era intermediada por um console de Terminal (equipamento com monitor de vídeo e teclado) e para crianças não alfabetizadas a partir de um console com botões conectados ao computador ligado por um cabo (cordão umbilical) a um robô mecânico, chamada tartaruga que percorria sobre uma grande folha de papel traçando imagens geométricas planas (figura 5.7).

Sem sombra de dúvidas pensar em Logo nas décadas de 1960 e 1970 era muito divertido, pois era possível ver um robô sobre uma folha de papel produzir desenhos incríveis (figura 5.8).

Depois veio a fase do controle remoto de periféricos onde o robô já não possuía mais um cordão umbilical e sim controlado por ondas de rádio, mais emocionante ainda.

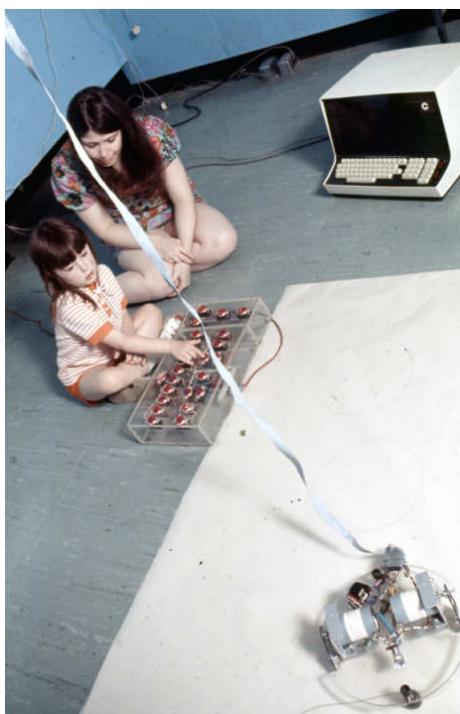


Figura 5.7 - Criança desenhando com Logo (SOLOMON, 2020, p. 39).



Figura 5.8 - Tartaruga remota (CATLIN, 2016).

Possivelmente por questões de custos o controle da tartaruga diretamente na tela de um computador se tornou bastante popular afastando fisicamente a linguagem da tartaruga.

A linguagem Logo vem sendo, há muito tempo, erroneamente vinculada a aprendizagem apenas de crianças. Isso pode até ter sido verdade, como as figuras anteriores mostram, há muito tempo, em meados de 1960, por ter sido inicialmente apresentada a esse grupo de pessoas. No entanto, a partir de 1975 com a introdução dos microcomputadores sua popularidade decolou e Logo se tornou bastante popular atraindo a atenção de outros públicos. Por controlar as ações de um robô, Logo se mostrou uma excelente linguagem para uso em ações de automação industrial e mecatrônica, indo além do que simplesmente fazer desenhos. Em certos momentos deste livro você deve ter notado exemplos de procedimentos que não elaboraram desenhos na área de ação do modo gráfico, como o procedimento "**TABUADA**" do capítulo "4" que mostra o resultado de sua operação na área de ação do modo texto. Esse é um tipo de ação que muitas vezes passa despercebido com o uso da linguagem Logo, pois é comum ver seu foco voltado a aplicações geométricas, até mesmo na quantidade de material produzido para apresentar a linguagem.

Para exemplificar a linguagem fora do eixo geométrico considere a seguir alguns exemplos aplicativos simples que efetuam algumas ações computacionais comuns a qualquer linguagem de programação.

Considere para o primeiro exemplo um procedimento chamado "**AREACIRC**" que receba a entrada de um valor real para a medida do raio de uma circunferência e apresente o resultado da área desta circunferência sem desenhá-la com três casa decimais. Para tanto, entre o seguinte código:

APRENDA AREACIRC

```
ATRIBUA "RAIO READWORD  
ATRIBUA "AREA 3.14159265359 * (POWER :RAIO 2)  
ESCREVA (SE "Área "= :AREA)  
END
```

Execute o procedimento "**AREACIRC**" e informe para um primeiro teste o valor "**5**" e veja a apresentação do resultado da área com "**78.53981633975**".

Observe cada detalhe de cada instrução do procedimento "**AREACIRC**". Note que são usados diversos elementos integrados na ideia de um todo com cada componente realizando uma ação que complementa outro componente.

O segundo exemplo é fundamentado a partir de um procedimento chamado "**DECISAO**" que recebe a entrada separada de dois valores numéricos representados pelas variáveis "**A**" e "**B**", soma-os armazenando seu resultado na variável "**X**" e mostra o resultado da soma na variável "**X**" caso este seja maior que "**10**". Para somas obtidas menores que "**10**" não deve ser realizada nenhuma operação. Desta forma, selecione a opção "**Editar...**" menu "**Arquivo**", acione botão "**OK**", acione as teclas de atalho "**<Ctrl> + <A>**", tecle "****" e entre o seguinte código:

APRENDA DECISAO

```
ATRIBUA "A READWORD  
ATRIBUA "B READWORD  
ATRIBUA "X :A + :B  
IF :X > 10 [  
    ESCREVA (SE "Resultado "= :X)  
]  
END
```

Execute o procedimento "**DECISAO**" e informe separadamente os valores "**5**" e "**6**" e veja a apresentação do resultado "**11**", depois execute novamente o procedimento e informe os valores "**5**" e "**5**" e observe que nada é apresentado.

5.5 - Manipulação básica de dados

Em alguns outros momentos deste estudo foram apresentados certos detalhes sobre o uso e apresentação de textos e o tratamento de dados. Cabe junto a este tópico rever alguns detalhes e apresentar outros que complementem este conhecimento. Observe a tabela 5.5, adaptado de Downey & Gay (2003, p. 31-32, 55, 57-58, 60 e 63).

FUNÇÃO	OPERAÇÃO
ASCII	Retorna código ASCII do caractere informado.
CHAR	Retorna o caractere correspondente ao código ASCII.
COUNT	Conta caracteres de uma palavra ou palavras de uma frase.
REVERSE	Inverte uma sequência de caracteres.
ITEM	Mostra um elemento de uma coleção a partir da posição.
LOWERCASE	Mostra caracteres em formato minúsculo.
SEMPRIMEIRO	Mostra todos os elementos, exceto o primeiro.
SEMÚLTIMO	Mostra todos os elementos, exceto o último.

Tabela 5.5 - Funções complementares (mais algumas).

Veja alguns detalhes indicados por instruções em azul e seus efeitos como resultados gerados em vermelho.

ESCREVA FIRST "ABCDE

A

ESCREVA FIRST 54321

5

ESCREVA LAST "ABCDE

E

ESCREVA LAST 54321

1

ESCREVA ASCII "A

65

ESCREVA CHAR 65

A

ESCREVA SEMPRIMEIRO 1.2345

.2345

ESCREVA REVERSE "ABCDE

EDCBA

ESCREVA LOWERCASE "ABCDE

abcde

ESCREVA SEMÚLTIMO "ABCDE

ABCD

ESCREVA ITEM 2 "ABCDE

B

ESCREVA ITEM 4 54321

2

ESCREVA WORD "Linguagem "Logo

LinguagemLogo

ESCREVA SENTENCE "Linguagem "Logo

Linguagem Logo

ESCREVA SE "Linguagem "Logo

Linguagem Logo

(ESCREVA "Linguagem "Logo)

Linguagem Logo

ESCREVA (SENTENCE "Estudo "de "Linguagem "Logo)

Estudo de Linguagem Logo

ESCREVA (SENTENCE 1 2 3 4 5)

1 2 3 4 5

ESCREVA (PRODUCT 2 3 4)

24

ESCREVA (SUM 1 2 3 4 5)

15

ESCREVA COUNT [Estudo de Linguagem Logo]

4

ESCREVA COUNT "ABCDE

5

ESCREVA COUNT (SENTENCE "Estudo "de "Linguagem "Logo)

4

ESCREVA COUNT (WORD "Estudo "de "Linguagem "Logo)

21

ESCREVA COUNT WORD PRIMEIRO "ABCD SEMPRIMEIRO 1234567890

10

5.6 - Escopo e visibilidade de variáveis

O espaço de memória pode ser controlado com a definição do escopo de comportamento das variáveis, que podem ser globais e locais, dependendo apenas da primitiva de definição de variável em uso. As variáveis globais são definidas com a primitiva **ATRIBUA** já usada neste livro e as variáveis locais são definidas com a primitiva **LOCAL**. Quando uma variável global é definida dentro de um procedimento ela se torna visível fora do procedimento e para todos os subprocedimentos relacionados o que é diferente para uma variável definida como local, pois neste caso, esta variável é visível dentro do procedimento em que foi definida e também aos subprocedimento relacionados, mas não será visível fora do procedimento.

Para realizar este teste acione a opção de menu "**Diversão**" e atente para o procedimento chamado "**VARGLOBA1**" com a definição de uma variável global e o acesso a esta variável de forma interna e externa ao procedimento. Para tanto, entre o seguinte código:

APRENDA VARGLOBA1

ATRIBUA "CONTADORG 1

ESCREVA :CONTADORG

ATRIBUA "CONTADORG :CONTADORG + 1

ESCREVA :CONTADORG

END

Em seguida execute a instrução de chamada do procedimento:

VARGLOBA1

Observe os resultados apresentados e na sequência na linha de comando do campo de entrada de comandos, dados e instruções informe a seguinte instrução:

ESCREVA SUM :CONTADORG 1

Veja na figura 5.12 a apresentação dos resultados de uso de variável global. Note que a instrução "ESCREVA SUM :CONTADORG 1" soma mais "1" ao valor da variável "CONTADORG" tornando seu resultado "3" por ser "CONTADORG" uma variável global.

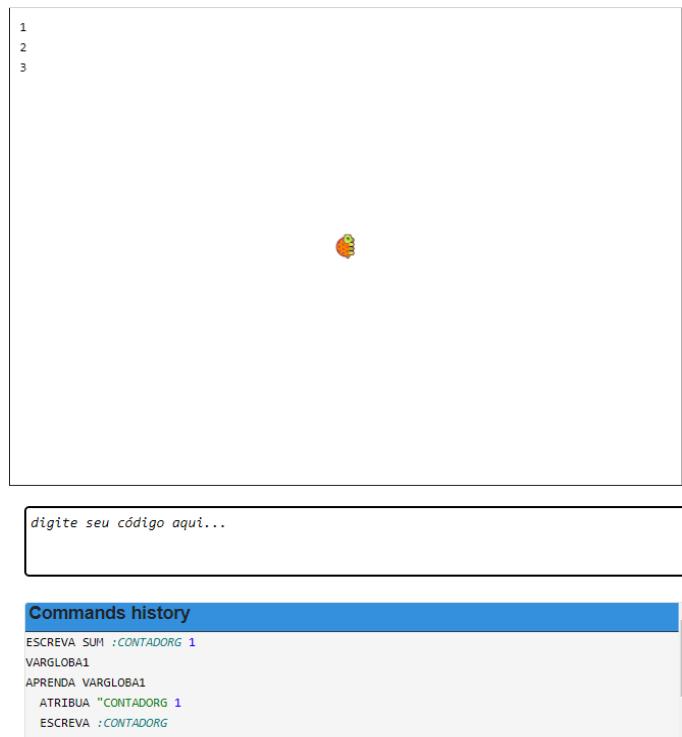


Figura 5.12 - Resultado no uso de variável global.

Para realizar este teste acione a opção de menu "**Diversão**" e atente para o procedimento chamado "**VARLOCAL1**" com a definição de uma variável local com acesso apenas interno ao procedimento. Desta forma, entre o seguinte código:

```
APRENDA VARLOCAL1
  LOCAL "CONTADORL"
  ATRIBUA "CONTADORL 1"
  ESCREVA :CONTADORL
  ATRIBUA "CONTADORL :CONTADORL + 1"
  ESCREVA :CONTADORL
END
```

Em seguida execute a instrução de chamada do procedimento:

VARLOCAL1

Observe os resultados apresentados e na sequência na linha de comando do campo de entrada de comandos, dados e instruções informe a seguinte instrução:

ESCREVA SUM :CONTADORL 1

Veja na figura 5.13 a apresentação dos resultados de uso de variável global. Note que a instrução "ESCREVA SUM :CONTADORL 1" tenta somar mais "1" ao valor da variável "CONTADORL" que retorna como resposta a informação "Don't know about variable CONTADORL" indicando que a variável não existe mais na memória, por ser a variável "CONTADORL" uma variável local.

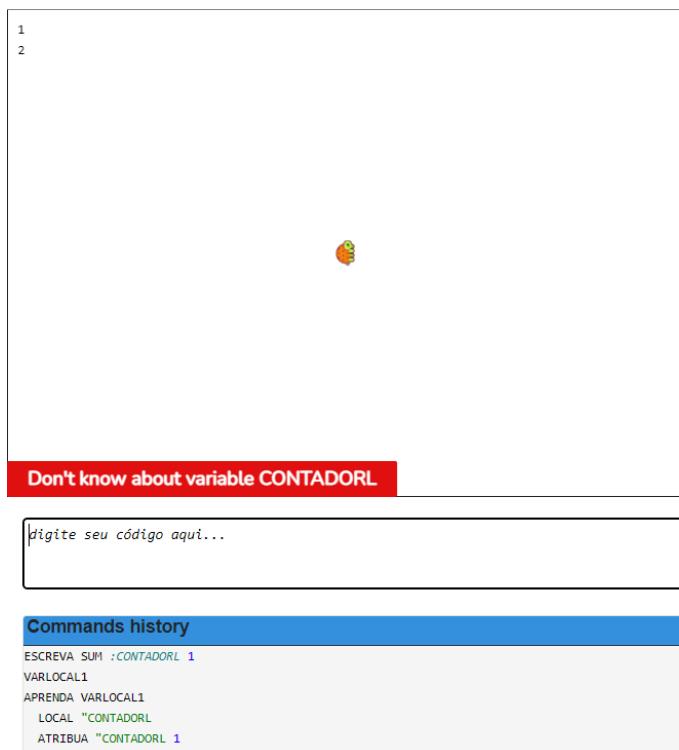


Figura 5.13 - Resultado no uso de variável local.

Observe que a primitiva **LOCAL** permite que seja definida uma variável com estado de visibilidade local, mas após está definição se faz uso da primitiva **ATRIBUA** normalmente. Então, atente ao fato de que se uma variável definida com **ATRIBUA** sem o uso prévio de "**LOCAL**" é global e com o uso prévio de "**LOCAL**" é local.

Veja em seguida o comportamento de variáveis globais e locais a partir do uso de subprocedimento (sub-rotinas).

Observe o procedimento chamado "**VARGLOBA2**" com a definição global de variável e uso de sub-rotina para a ação de contagem. Para tanto, entre o seguinte código após executar a ação da opção do menu "**Diversão**":

```
APRENDA VARGLOBA2
  ATRIBUA "CONTADORGX 1
  ESCREVA :CONTADORGX
  SUBVARGLOBA2
END

APRENDA SUBVARGLOBA2
  ATRIBUA "CONTADORGX :CONTADORGX + 1
  ESCREVA :CONTADORGX
END
```

Em seguida execute a instrução de chamada do procedimento:

VARGLOBA2

Observe os resultados apresentados e na sequência na linha de comando do campo de entrada de comandos, dados e instruções informe a seguinte instrução:

ESCREVA SUM :CONTADORGX 1

Veja a apresentação dos resultados "1", "2"e "3" a partir da execução do procedimento principal com chamada de subprocedimento (sub-rotina) utilizando-se variável global, indicado na figura 5.14.

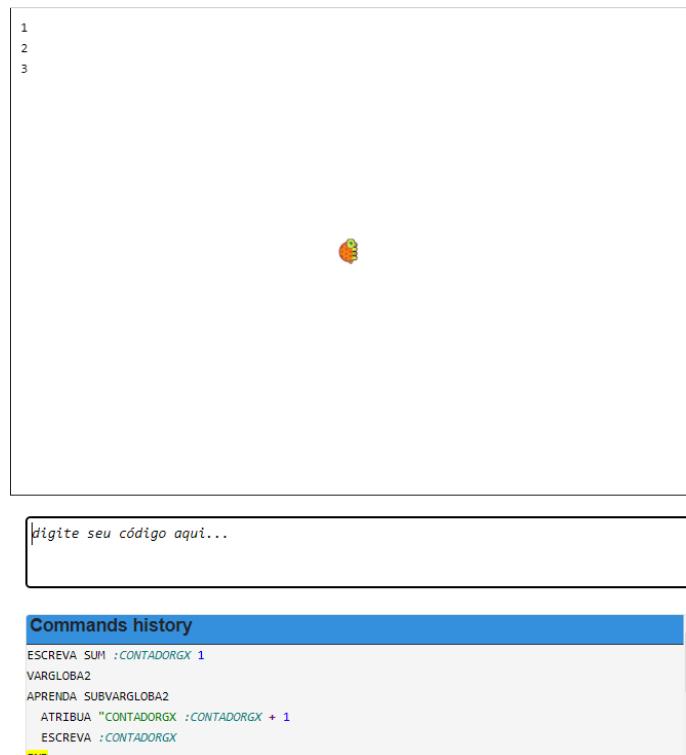


Figura 5.14 - Resultado no uso de variável global com procedimento e sub-rotina.

Na sequência veja o procedimento chamado "**VARLOCAL2**" com a definição local de variável e uso de sub-rotina para a ação de contagem. Assim sendo, entre o seguinte código após executar a ação da opção do menu "**Diversão**":

```
APRENDA VARLOCAL2
  LOCAL "CONTADORLX
  ATRIBUA "CONTADORLX 1
  ESCREVA :CONTADORLX
  SUBVARLOCAL2
END
```

```
APRENDA SUBVARLOCAL2
  ATRIBUA "CONTADORLX :CONTADORLX + 1
  ESCREVA:CONTADORLX
END
```

Em seguida execute a instrução de chamada do procedimento:

VARLOCAL2

Veja a apresentação do valor "1" e da mensagem "**Don't know how to ESCREVA:CONTADORLX**" a partir da execução do procedimento principal com chamada de sub procedimento (sub-rotina) utilizando-se variável local, indicado na figura 5.15. Note que a sub-rotina retorna erro por não conseguir "ver" a variável na memória (por ser uma variável local).

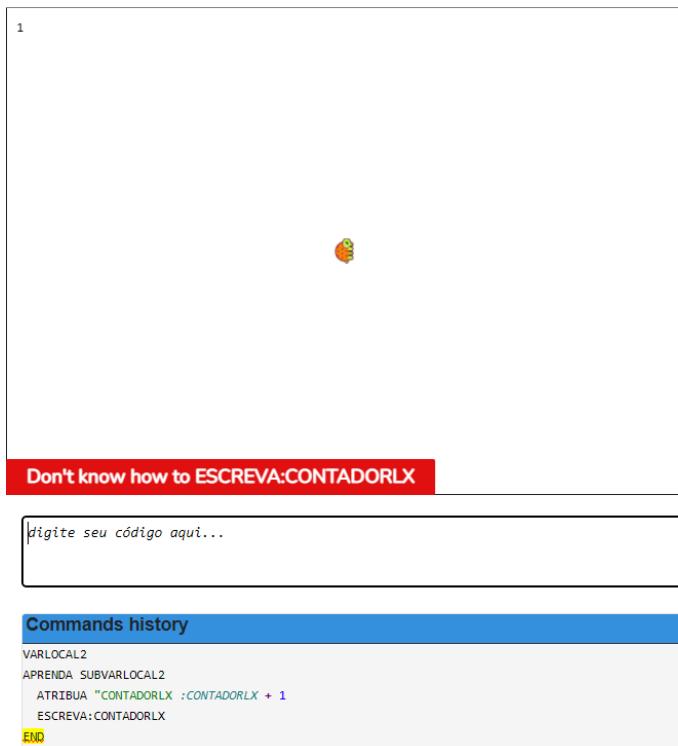


Figura 5.15 - Resultado no uso de variável local com procedimento e sub-rotina.

5.7 - Exercícios de fixação

1. Criar procedimento chamado **CAP0501** que efetue a leitura de um valor numérico inteiro e apresente o resultado do valor lido elevado ao quadrado sem efetuar o armazenamento do resultado em memória. A variável que receberá a entrada do dado deve ser definida como local.
2. Criar procedimento chamado **CAP0502** que efetue a leitura de um valor numérico inteiro e apresente o resultado do valor lido elevado ao cubo com armazenamento do resultado calculado em memória. As variáveis devem ser definidas como local.
3. Criar procedimento chamado **CAP0503** que efetue a leitura de uma temperatura em graus Celsius e apresente essa temperatura em graus Fahrenheit, sua conversão. A fórmula de conversão é " $F \leftarrow C * 9 / 5 + 32$ ", sendo "F" a temperatura em Fahrenheit e "C" a temperatura em Celsius. Armazene em memória o resultado calculado. Use variáveis locais. Formate a saída numérica com duas casas decimais.
4. Criar procedimento chamado **CAP0504** que efetue a leitura de uma temperatura em graus Fahrenheit apresente essa temperatura em graus Celsius, sua conversão. A fórmula de conversão é " $C \leftarrow (F - 32) * 5 / 9$ ", sendo "F" a temperatura em Fahrenheit e "C" a temperatura em Celsius. Armazene em memória o resultado calculado. Use variáveis locais. Formate a saída numérica com duas casas decimais.

5. Criar procedimento chamado **CAP0505** que efetue a leitura de dois valores numéricos inteiros (representados pelas variáveis locais "A" e "B") e mostre o resultado armazenado em memória do quadrado da diferença do primeiro valor (variável "A") em relação ao segundo valor (variável "B") junto a variável local "R".
6. Criar procedimento chamado **CAP0506** que efetue a leitura de um número inteiro qualquer em uma variável local e multiplique este número por "2" armazenando o resultado em memória. Apresentar o resultado da multiplicação somente se o resultado for maior que "30".
7. Criar procedimento chamado **CAP0507** que efetue a leitura de dois valores numéricos reais representados pelas variáveis locais "A" e "B" e apresente o resultado da diferença do maior valor pelo menor valor sem armazenar o resultado em memória.
8. Criar procedimento chamado **CAP0508** que efetue a leitura de dois valores numéricos reais representados pelas variáveis locais "A" e "B" e apresente o resultado da diferença do maior valor pelo menor valor com armazenamento do cálculo em memória.
9. Criar procedimento chamado **CAP0509** que efetue a leitura de três valores numéricos inteiros desconhecidos representados pelas variáveis locais "A", "B" e "C". O procedimento deve somar esses valores, armazenar o resultado em memória e apresentar este resultado somente se for "**maior ou igual**" a "100".
10. Criar procedimento chamado **CAP0510** que apresente a soma dos cem primeiros números naturais "(1 + 2 + 3 + ... + 98 + 99 + 100)" utilizando-se a primitiva "**REPITA**".
11. Criar procedimento chamado **CAP0511** que apresente a soma dos cem primeiros números naturais "(1 + 2 + 3 + ... + 98 + 99 + 100)" utilizando-se a primitiva "**WHILE**".
12. Criar procedimento chamado **CAP0512** que apresente a soma dos cem primeiros números naturais "(1 + 2 + 3 + ... + 98 + 99 + 100)" utilizando-se a primitiva "**FAÇAPARA**".
13. Criar procedimento chamado **CAP0513** que apresente o somatório dos *valores pares* existentes na faixa de "1" até "500". Use a primitiva "**REPITA**".
14. Criar procedimento chamado **CAP0514** que apresente o somatório dos *valores pares* existentes na faixa de "1" até "500". Use a primitiva "**FAÇAPARA**".
15. Criar procedimento chamado **CAP0515** que mostre todos os valores numéricos divisíveis por "4" *menores* que "20" iniciando a contagem em "1". Use a primitiva "**REPITA**".
16. Criar procedimento chamado **CAP0516** que mostre todos os valores numéricos divisíveis por "4" *menores* que "20" iniciando a contagem em "1". Use a primitiva "**FAÇAPARA**".
17. Criar procedimento chamado **CAP0517** que apresente os valores numéricos inteiros compreendidos na faixa de "0" e "4". Use a primitiva "**REPITA**".
18. Criar procedimento chamado **CAP0518** que apresente os valores numéricos inteiros compreendidos na faixa de "0" e "4". Use a primitiva "**WHILE**".

19. Criar procedimento chamado **CAP0519** que apresente os valores numéricos inteiros compreendidos na faixa de "0" e "4". Use a primitiva "**FAÇAPARA**".
20. Criar procedimento chamado **CAP0520** que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use a primitiva "**REPITA**".
21. Criar procedimento chamado **CAP0521** que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use a primitiva "**WHILE**".
22. Criar procedimento chamado **CAP0522** que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use a primitiva "**FAÇAPARA**".

Apêndice A - Exemplos geométricos

Neste livro foram apresentados exemplos de criação de figuras geométricas. Nesta parte, são indicados outros exemplos de figuras geradas a partir do uso de instruções diretas ou de procedimentos isolados ou usados como apoio para instruções diretas. O objetivo deste material é ampliar seu conhecimento e fornecer subsídios para aumentar sua criatividade. Algumas das imagens apresentadas são reproduções de códigos de sítios ou manuais antigos da linguagem Logo: Petti (2021), Muller (1998), Harvey (1997), Corrales Mora (1996), Sparer (1984), Winter (1984), Abelson (1984), ATARI (1983), Kheriaty & Gerhold (1982), Bass (2002), Erfan's (2021) e Joys (2021).

INSTRUÇÕES DIRETAS COM E SEM AUXILIO DE PROCEDIMENTO

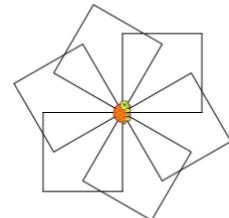
Parte das figuras apresentadas usarão o procedimento coringa chamado **FIGURAS** (figuras geométricas simples) que poderá desenhar imagens, de triângulos a circunferências.

```
APRENDA FIGURAS :LADOS :TAMANHO  
REPITA :LADOS [PF :TAMANHO GD 360 / :LADOS]  
END
```

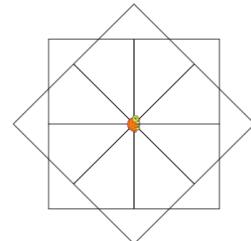
Desta forma, caso deseje a imagem de um quadrado execute "**FIGURAS 4 80**", caso deseje a imagem de um triângulo execute "**FIGURAS 3 80**", desejando uma circunferência execute "**FIGURAS 360 1**" e assim por diante.

A seguir são apresentadas as instruções e as imagens geradas por essas instruções. Atente para cada detalhe pois isto poderá ajudar na resolução dos exercícios. Para a realização desses testes configure o ambiente com os valores "**854**" e "**610**" respectivamente para os campos "**Width**" e "**Height**".

```
TAT  
REPITA 6 [GD 60 FIGURAS 4 80]
```

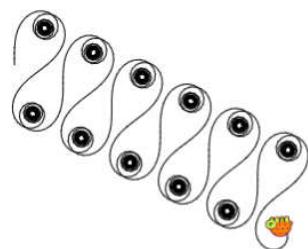


```
TAT  
REPITA 8 [FIGURAS 4 120 GD 45]
```



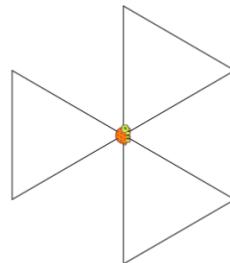
TAT

FAÇAPARA [I 0 2000] [PF 5 GD (90 * SIN :I)]



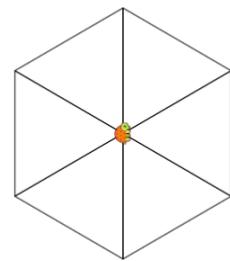
TAT

REPITA 3 [FIGURAS 3 150 GD 120]



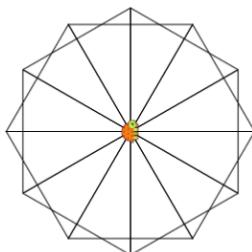
TAT

REPITA 6 [FIGURAS 3 140 GD 60]



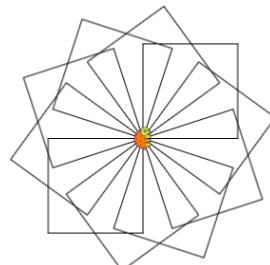
TAT

REPITA 360 / 30 [FIGURAS 3 130 GD 30]

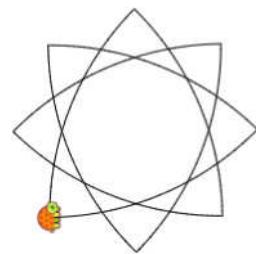


TAT

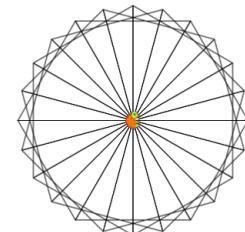
REPITA 10 [REPITA 4 [PF 100 GD 90] GD 36]



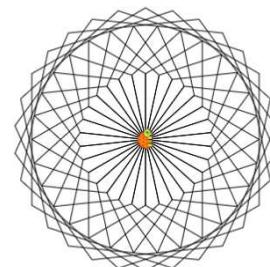
TAT
REPITA 8 [REPITA 45 [PF 4 GD 1] GD 90]



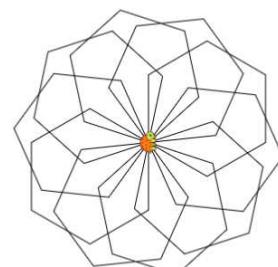
TAT
REPITA 24 [REPITA 3 [PF 150 GD 120] GD 15]



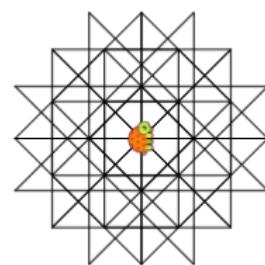
TAT
REPITA 30 [REPITA 6 [PF 75 GD 60] GD 12]



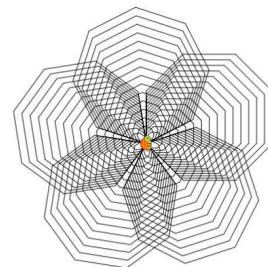
TAT
REPITA 10 [REPITA 6 [PF 75 GD 60] GD 36]



TAT
REPITA 8 [REPITA 8 [GE 135 PF 90] GE 45]



```
TAT  
FAÇAPARA [I 10 80 5] [  
    REPITA 5 [  
        REPITA 8 [  
            PF :I GD 45  
        ]  
        GD 72  
    ]  
]
```



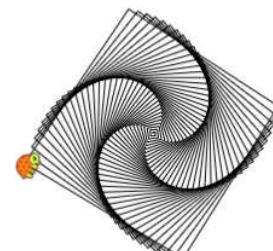
```
TAT  
REPITA 100 [PF CONTEVEZES GD 40]
```



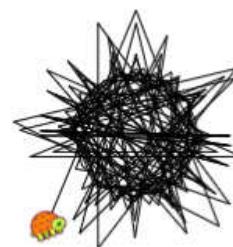
```
TAT  
REPITA 100 [PF CONTEVEZES GD 80]
```



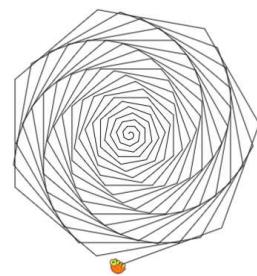
```
TAT  
REPITA 150 [PF CONTEVEZES GD 89]
```



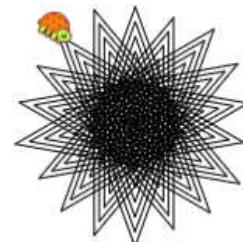
```
TAT  
REPITA 150 [PF CONTEVEZES GD CONTEVEZES * 1.5]
```



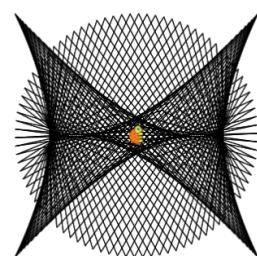
TAT
REPITA 150 [PF CONTEVEZES GD 50]



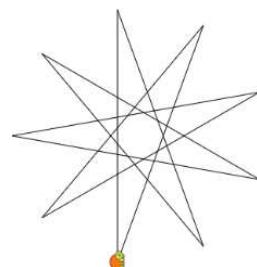
TAT
REPITA 150 [PF CONTEVEZES GD 500]



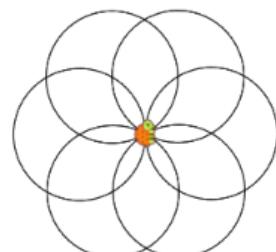
TAT
REPITA 360 [
 MUDEXY (SIN(89 * CONTEVEZES)) * 150
 (SIN(179 * CONTEVEZES)) * 150
]



TAT
REPITA 9 [PF 300 GD 160]

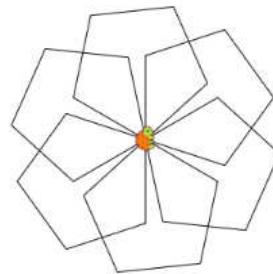


TAT
REPITA 6 [GD 60 FIGURAS 360 1]



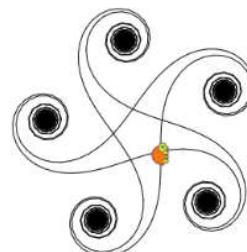
TAT

REPITA 6 [GD 60 FIGURAS 5 80]



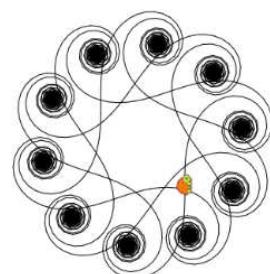
TAT

REPITA 1800 [PF 10 GD CONTEVEZES + .1]



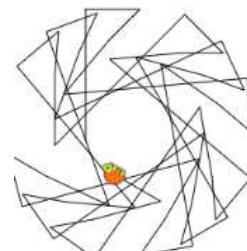
TAT

REPITA 3600 [PF 10 GD CONTEVEZES + .2]



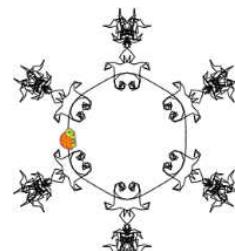
TAT

REPITA 12 [
PF 120 GD 90 PF 50 GD 135 PF 40
GE 185 PT 65 GD 45 PF 50
]

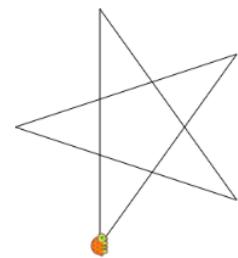


TAT

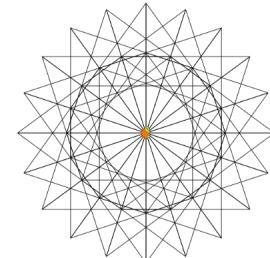
FAÇAPARA [I 0 1002] [
PF 8 MUDED (360 * (POWER :I 3) / 1002)
]



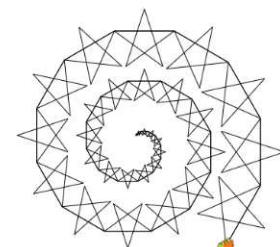
TAT
REPITA 5 [PF 250 GD 144]



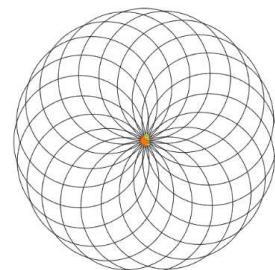
TAT
REPITA 20 [REPITA 5 [PF 250 GD 144] GD 18]



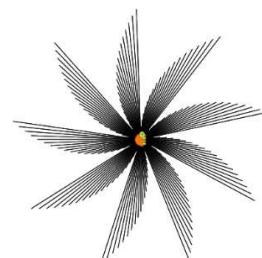
TAT
FAÇAPARA [I 0 95 3] [
REPITA 5 [
PF :I GD 144
]
PF :I GD 30
]



TAT
REPITA 20 [REPITA 180 [PF 4 GD 2] GD 18]



TAT
REPITA 9 [
FAÇAPARA [I 10 200 10] [
PF :I PT :I GD 2
]
]



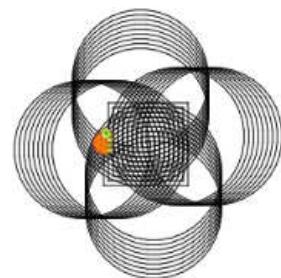
TAT

REPITA 7 [PF 100 GD 360 * 3 / 7]



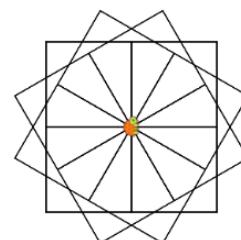
TAT

```
REPITA 36 [
  REPITA 36 [
    PF 10 GD 10
  ]
  PF CONTEVEZES GD 90 PF CONTEVEZES
]
```



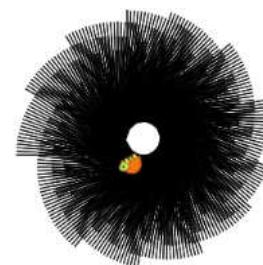
TAT

REPITA 12 [REPITA 4 [FIGURAS 4 100] GD 30]



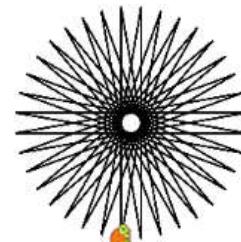
TAT

REPITA 12 [REPITA 55 [PF 100 PT 100 GD 2] PF 45]



TAT

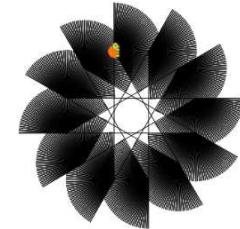
REPITA 54 [REPITA 8 [PF 200 GD 170]]



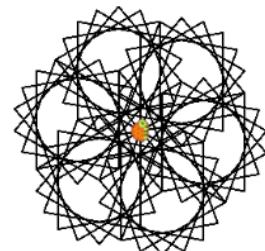
TAT
REPITA 18 [REPITA 5 [GD 40 PF 100 GD 120] GD 20]



TAT
REPITA 12 [
 REPITA 75 [
 PF 70 PT 70 GD 2
]
 PF 187.5
]



TAT
REPITA 12 [REPITA 360 [PF 100 GD 100] GD 60]



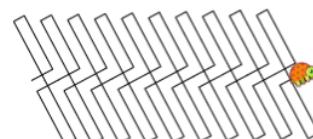
INSTRUÇÕES A PARTIR DA DEFINIÇÃO DE PROCEDIMENTOS

Os scripts a seguir são baseados e adaptado a partir da bibliografia usada além de material instrucional nos sítios <https://fmslogo.sourceforge.io/workshop/> e <https://helloacm.com/logo-turtle-tutorial-how-to-draw-fractal-stars/>.

```
APRENDA SEGMENTO
  PF 20 GE 90
  PF 50 GE 90
  PF 10 GE 90
  PF 55
  PF 55 GD 90
  PF 10 GD 90
  PF 50 GD 90
  PF 20
END

APRENDA PADRAO
  GD 62
  REPITA 10 [ SEGMENTO ]
END
```

EXECUTE: TAT PADRAO

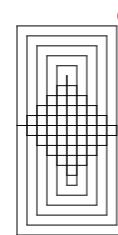


```

APRENDA CAMINHO
REPITA 22 [
  GD 90
  PF 110 - CONTEVEZES * 10
  GD 90
  PF CONTEVEZES * 10
]
END

```

EXECUTE: TAT CAMINHO

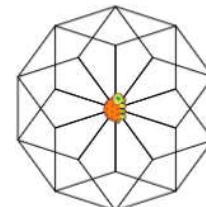


```

APRENDA HEXAFLOR :PETALAS
REPITA :PETALAS [
  FIGURAS 5 50
  GD 360 / :PETALAS
]
END

```

EXECUTE: TAT HEXAFLOR 10



```

APRENDA CICLO :INDICE :ULTIMO
MUDEXY :INDICE          0
MUDEXY :ULTIMO          :INDICE
MUDEXY (:ULTIMO - :INDICE) :ULTIMO
MUDEXY 0                (:ULTIMO - :INDICE)
DICE)
MUDEXY :INDICE          0
END

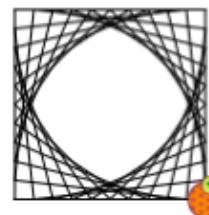
```

```

APRENDA QUADART
REPITA 10 [CICLO CONTEVEZES * 10 100]
END

```

EXECUTE: TAT QUADART

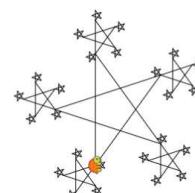


```

APRENDA ESTRELANDO :TAMANHO :LIMITE
IF :TAMANHO < :LIMITE [STOP]
REPITA 5 [PF :TAMANHO ESTRELANDO
:TAMANHO * .3 :LIMITE GD 144]
END

```

EXECUTE: TAT ESTRELANDO 150 10



```

APRENDA BANDEIRA
PF 50
FIGURAS 4 50
END

```

EXECUTE: TAT BANDEIRA

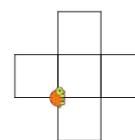


```

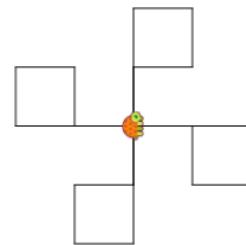
APRENDA CRUZ
REPITA 4 [BANDEIRA GD 90]
END

```

EXECUTE: TAT CRUZ



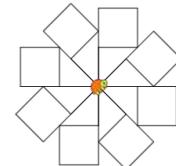
```
APRENDA VOLTABANDA  
BANDEIRA  
PT 50  
END
```



```
APRENDA BANDEIRAS  
REPITA 4 [VOLTABANDA GD 90]  
END
```

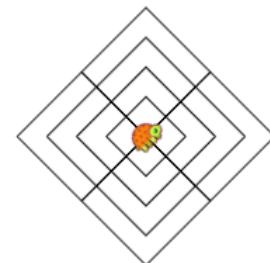
EXECUTE: TAT BANDEIRAS

```
APRENDA MUITASBANDEIRAS  
BANDEIRAS  
GD 45  
BANDEIRAS  
END
```



EXECUTE: TAT MUITASBANDEIRAS

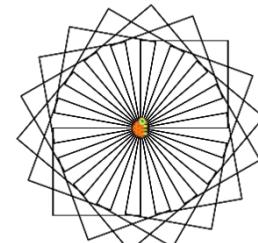
```
APRENDA QUADRADOS  
FIGURAS 4 20  
FIGURAS 4 35  
FIGURAS 4 50  
FIGURAS 4 65  
END
```



```
APRENDA DIAMANTES  
GD 45  
REPITA 4 [QUADRADOS GD 90]  
END
```

EXECUTE: TAT DIAMANTES

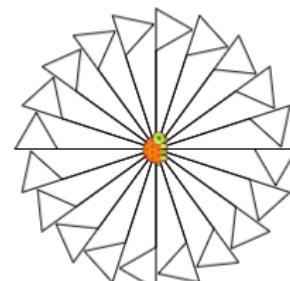
```
APRENDA QUADRADO  
REPITA 4 [FIGURAS 4 100]  
END
```



```
APRENDA FLORQUADRADA  
REPITA 18 [QUADRADO GD 20]  
END
```

EXECUTE: TAT FLORQUADRADA

```
APRENDA BANDTRI  
PF 100 GD 120  
PF 30 GD 120  
PF 30 GD 120  
PT 70  
END
```

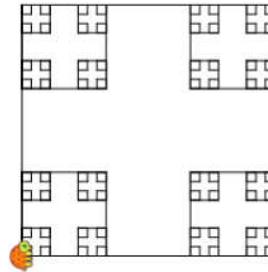


```
APRENDA FLORBANDTRI  
REPITA 20 [BANDTRI WAIT 30 GD 18]  
END
```

EXECUTE: TAT FLORBANDTRI

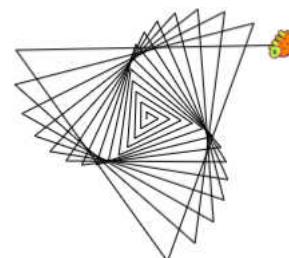
```
APRENDA QUADRICULADO :VALOR
IF (:VALOR < 3) [
  STOP
]
REPITA 4 [
  QUADRICULADO :VALOR / 3
  PF :VALOR
  GD 90
]
END
```

EXECUTE: TAT QUADRICULADO 200



```
APRENDA TUNELTRI :TAMANHO :ANGULO
IF (:TAMANHO > 200) [
  STOP
]
PF :TAMANHO
GD :ANGULO
TUNELTRI :TAMANHO + 5 :ANGULO + 0.12
END
```

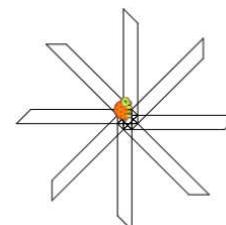
EXECUTE: TAT TUNELTRI 5 120



```
APRENDA PA
REPITA 2 [PF 100 GD 135 PF 20 GD 45]
END
```

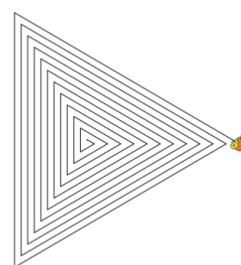
```
APRENDA HELICE
REPITA 8 [PA GD 135 PF 20]
END
```

EXECUTE: TAT HELICE



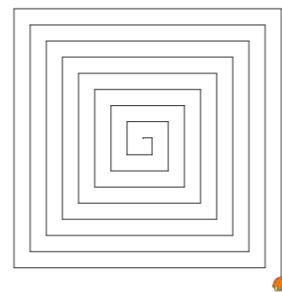
```
APRENDA ESPIRALTRI :LADO
IF (:LADO > 350) [
  STOP
]
PF :LADO WAIT 20
GD 120 WAIT 20
ESPIRALTRI :LADO + 10 WAIT 30
END
```

EXECUTE: TAT ESPIRALTRI 1



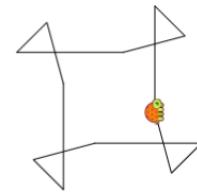
```
APRENDA ESPIRALQUAD :LADO
IF (:LADO > 350) [
  STOP
]
PF :LADO WAIT 20
GD 90 WAIT 20
ESPIRALQUAD :LADO + 10 WAIT 30
GD 90 WAIT 50
END
```

EXECUTE: TAT ESPIRALQUAD 1



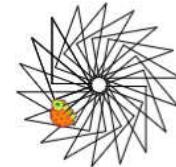
```
APRENDA FORMA  
PF 100 GD 135  
PF 40 GD 120  
PF 60 GD 15  
END
```

EXECUTE: TAT REPITA 4 [FORMA]



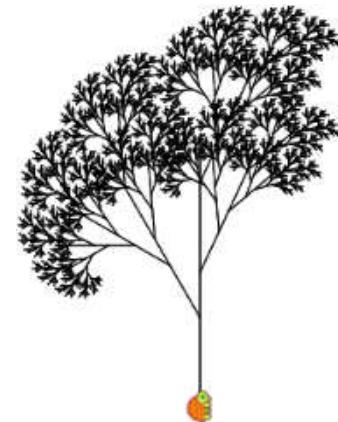
```
APRENDA FORMATRI  
PF 50 GD 150  
PF 60 GD 100  
PF 30 GD 90  
END
```

EXECUTE: TAT REPITA 20 [FORMATRI]



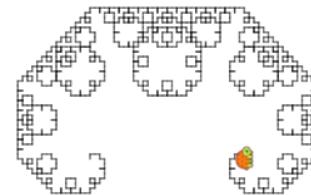
```
APRENDA ARVORE :TAMANHO  
IF :TAMANHO < 5 [PF :TAMANHO PT :TAMANHO STOP]  
PF :TAMANHO / 3  
GE 30  
ARVORE :TAMANHO * 2 / 3  
GD 30  
PF :TAMANHO / 6  
GD 25  
ARVORE :TAMANHO / 2  
GE 25  
PF :TAMANHO / 3  
GD 25  
ARVORE :TAMANHO / 2  
GE 25  
PF :TAMANHO / 6  
PT :TAMANHO  
END
```

EXECUTE: TAT ARVORE 200



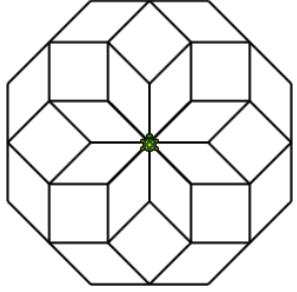
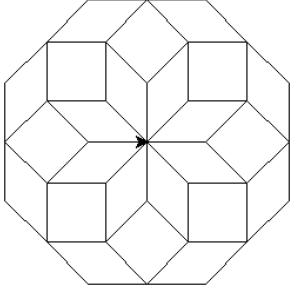
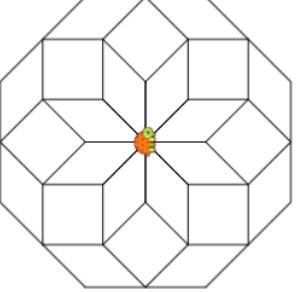
```
APRENDA CURVAC :TAMANHO :NIVEL  
IF :NIVEL = 0 [PF :TAMANHO STOP]  
CURVAC :TAMANHO :NIVEL - 1 GD 90  
CURVAC :TAMANHO :NIVEL - 1 GE 90  
END
```

EXECUTE: TAT CURVAC 5 10



Apêndice B - Espiral hexagonal (imagem da capa)

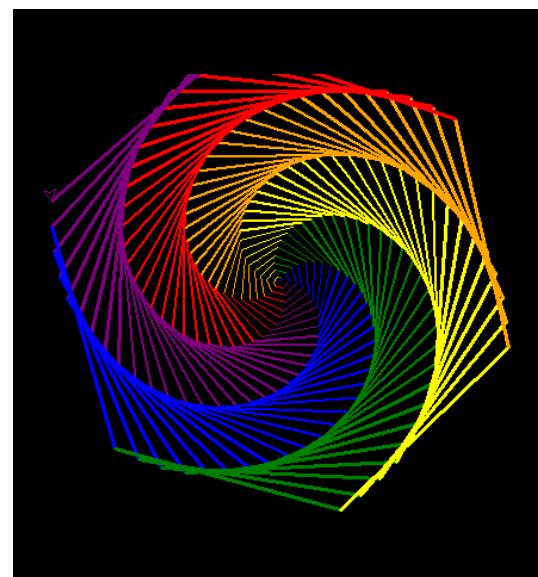
Muitos dos recursos de uso da linguagem Logo podem ser suportados em outras linguagens de programação. Por exemplo, a linguagem "Small Basic" desenvolvida pela empresa Microsoft (<https://smallbasic-publicwebsite.azurewebsites.net>) e "Python" produzida pela comunidade Python (<https://www.python.org>), além do processador de textos "Writer" do "LibreOffice". Observe os códigos escritos em "Small Basic", "Python" e "Logo" e a imagem gerada:

SMALL BASIC	PYTHON	LOGO (FMSLogo)
<pre>For I = 1 To 8 For J = 1 To 8 Turtle.Move(50) Turtle.Turn(45) EndFor Turtle.Turn(45) EndFor</pre>	<pre>import turtle for I in range(8): for J in range(8): turtle.fd(50) turtle.rt(45) turtle.rt(45) input()</pre>	<pre>FAÇAPARA [I 1 8] [FAÇAPARA [J 1 8] [PF 50 GD 45] GD 45]</pre>
		

Onde, o indicativo "freq" refere-se a uma frequência de som medida em hertz (vibração do som por segundo) e "duração" o tempo de execução do som em milissegundos, ambos indicados como valores numéricos inteiros. Quanto maior for a frequência mais agudo o som será.

A imagem da capa deste livro é originalmente apresentada no sítio "ProgrammerSough" a partir do endereço (<https://www.programmersought.com/article/92794745509/>), tendo sido produzida por meio da linguagem "Python" de acordo com o código seguinte adaptado:

```
#Drawing colorful spirals
import turtle
def draw_spin():
    colores = [
        'red', 'purple', 'blue',
        'green', 'yellow', 'orange'
    ]
    turtle.bgcolor('black')
    for x in range(200):
        turtle.pencolor(colores[x % 6])
        turtle.width(x / 100 + 1)
        turtle.forward(x)
        turtle.left(59)
draw_spin()
input()
```

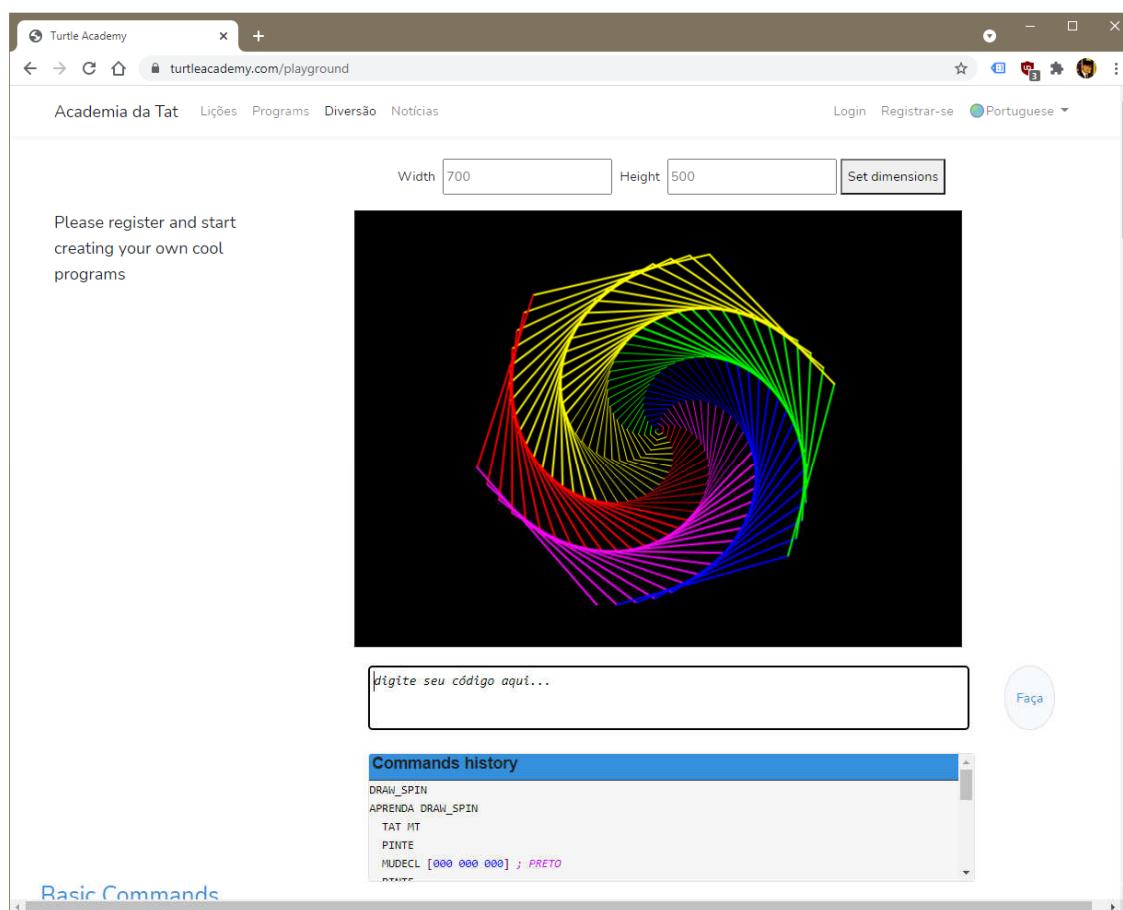


A partir do código "Python" foi realizada a transliteração e escrita de um código compatível em "Logo" que gerou a imagem usada na capa deste trabalho com o código:

APRENDA DRAW_SPIN

```
TAT MT  
PINTE  
MUDECL [000 000 000] ; PRETO  
PINTE  
FAÇAPARA [X 0 199] [  
    IF (MODULO :X 6) = 0 [MUDECL [255 000 000]] ; 0 - VERMELHO  
    IF (MODULO :X 6) = 1 [MUDECL [128 000 128]] ; 1 - ROXO  
    IF (MODULO :X 6) = 2 [MUDECL [000 000 255]] ; 2 - AZUL  
    IF (MODULO :X 6) = 3 [MUDECL [000 128 000]] ; 3 - VERD  
    IF (MODULO :X 6) = 4 [MUDECL [255 255 000]] ; 4 - AMARELO  
    IF (MODULO :X 6) = 5 [MUDECL [255 165 000]] ; 5 - LARANJA  
    MUDELARGURA (:X / 100 + 1)  
    PF :X  
    GE 59  
]  
OT  
MUDELARGURA 0  
END
```

O código do procedimento "**DRAW_SPIN**" produzido em "**Logo**" ao ser executado mostra imagem semelhante ao código original produzido em "**Python**".



Veja o que realiza cada instrução do procedimento "**DRAW_SPIN**" a partir das primitivas *Logo* e sua relação com os comandos *Python*:

- "APRENDA" e "END" são usadas para definir o escopo de escrita do programa no procedimento a partir do nome indicado a frente de "APRENDA", sendo isso compatível ao comando "**def**";
- "TAT" e "MT" efetuam respectivamente a limpeza da tela e a apresentação do ícone da tartaruga não tendo nenhuma relação direta com o código *Python* indicado;
- "PINTE MUDECL [000 000 000] PINTE" efetua a mudança da cor do fundo da área de trabalho para a cor preto de acordo com o código "RGB [000 000 000]" similar a instrução "turtle.bgcolor('black')";
- "FAÇAPARA" efetua a execução de duzentas passagens do grupo de instruções subordinadas contadas de "0" até "199" antes de encontrar a primitiva "OT" estando de acordo com a instrução "**for x in range(200):**" que efetivamente faz contagem de "0" até "199";
- Dentro do escopo de ação da primitiva "FAÇAPARA" encontra-se definida uma sequência de instruções "IF" que detectam os valores do resto da divisão do conteúdo da variável "X" por "6" (que é a quantidade de cores em uso) que geram valores de resto entre "0" e "5" para a detecção e uso da cor definida para cada linha traçada do hexágono, estando este conjunto de instruções consoante a instrução "turtle.pencolor(colores[x % 6])". Logo não opera com o uso de variáveis compostas (matrizes) como *Python* "colores[x % 6]" por esta razão o uso das primitivas "SE" é necessário;
- "MUDELARGURA (X / 100 + 1)" tem por finalidade a partir da primitiva "MUDELARGURA" mudar a largura do traço do desenho em andamento estando está instrução em consonância com a instrução "turtle.width(x/100 + 1)";
- "PF :X" faz a apresentação do traço com valores crescentes na variável "X" de "0" até "199" sendo compatível com a instrução "turtle.forward(x)";
- "GE 59" faz o giro em graus do traço para a formação de uma figura hexagonal, pois "59" é o valor numérico mais próximo de "60" que são os graus de formação de um hexágono estando de acordo com a instrução "turtle.left(59)";
- As demais instruções "OT" e " MUDELARGURA 0" efetuam respectivamente o ocultamento da tartaruga e o retorno do traço ao seu modo padrão, não tendo nenhuma relação com as demais instruções do código em *Python* que não vem ao caso.

ANOTAÇÕES

Apêndice C - Gabarito

CAPÍTULO 3

- Quais são as figuras geométricas planas desenhadas a partir das seguintes instruções? Diga qual é a figura sem executar a instrução no ambiente de programação.

REPITA 4 [PARAFRENTE 100 GIRADIREITA 90] Quadrado

REPITA 5 [PF 100 GE 72] Pentágono

REPITA 3 [PT 100 GD 120] Triângulo

REPITA 36 [PF 20 GD 10] Circunferência

- Criar procedimento chamado **RETANGULO1** (sem acento) que desenhe um retângulo com lados de tamanhos **60** e **100** para frente com giro de gruas para à direita. O procedimento deve desenhar a imagem sem o uso do recurso **REPITA**.

APRENDA RETANGULO1

```
PF 60  
GD 90  
PF 100  
GD 90  
PF 60  
GD 90  
PF 100  
GD 90
```

END

- Criar procedimento chamado **RETANGULO2** (sem acento) que desenhe um retângulo com lados de tamanhos **60** e **100** para trás com giro de gruas à esquerda. Usar a primitiva **REPITA**.

APRENDA RETANGULO2

```
REPITA 2 [  
    PT 60  
    GE 90  
    PT 100  
    GE 90  
]  
END
```

4. Criar, sem uso da primitiva **REPITA**, procedimento chamado **PENTAGONO** (sem acento) que construa um GEntágono com tamanho **40**. Avance para frente com giro a direita.

APRENDA PENTAGONO

```
PF 40  
GD 72  
END
```

5. Criar procedimento chamado **DECAGONO** (sem acento) que construa uma figura com dez lados a partir do uso da primitiva **REPITA** com giro de graus à esquerda com avanço a gente de **30**.

APRENDA DECAGONO

```
REPITA 10 [PF 30 GE 36]  
END
```

6. Criar procedimento chamado **ICOSAGONO** (sem acento) que desenhe a figura de mesmo nome a partir do uso da primitiva **REPITA** com tamanho **35** movimentando para trás.

APRENDA ICOSAGONO
REPITA 20 [PT 35 GD 18]
END

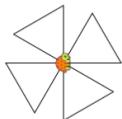
ou APRENDA ICOSAGONO
REPITA 20 [PT 35 GE 18]
END

CAPÍTULO 4

- Crie um procedimento chamado **TRIANGEX**, que apresente um triângulo equilátero com lado de tamanho **70** para trás a partir de giros a esquerda.

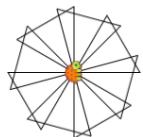
APRENDA TRIANGEX
REPITA 3 [
PT 70 GE 120
]
END

- Criar procedimento chamado **FLORTRIG** desenhado a partir do procedimento **TRIANGEX** com giro a direita de modo que tenha a seguinte aparência.



APRENDA FLORTRIG
REPITA 4 [
TRIANGEX GE 90
]
END

- Criar procedimento chamado **VENTITRIG** desenhado a partir do procedimento **TRIANGEX** com giro a esquerda de modo que tenha a seguinte aparência.



APRENDA VENTITRIG
REPITA 8 [
TRIANGEX GE 45
]
END

- Sem executar no computador descrimine qual imagem é apresentada.

REPITA 12 [REPITA 3 [PF 50 GE 120] GE 30]

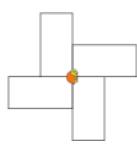
A imagem apresentada baseia-se em um quadrado repetido três vezes que após ser

Rotacionado a trinta graus é repetido mais doze vezes.

- Criar procedimento chamado **RETANGULO3** com tamanhos **100** (altura) e **50** (largura). Movimente-se para frente com giro a direita.

APRENDA RETANGULO3
REPITA 2 [
PF 100
GE 90
PF 50
GE 90
]
END

6. Criar procedimento chamado **CATAVENTO1** com o formato da figura seguinte a partir do procedimento **RETANGULO3**.

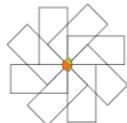


```
APRENDA CATAVENTO1
REPITA 4 [
    RETANGULO3 GE 90
]
END
```

7. Criar procedimento chamado **TRIANGPR** que desenhe um triângulo equilátero com tamanho definido por parâmetro com comando **FAÇAPARA** contando de **0** a **2** no sentido a frente com giro a direita.

```
APRENDA TRIANGPR :TAMANHO
FAÇAPARA [0 0 2] [
    PF :TAMANHO GE 120
]
END
```

8. Criar procedimento chamado **CATAVENTO2** com o formato da figura seguinte a partir do procedimento **RETANGULO3**.



```
APRENDA CATAVENTO2
REPITA 8 [
    RETANGULO3 GE 45
]
END
```

9. Descubra sem o uso do computador qual é a imagem:

```
APRENDA QUADRO :TAMANHO
REPITA 4 [
    PF :TAMANHO
    GD 90
]
GD 45
PF :TAMANHO * 7 / 5
PT :TAMANHO * 7 / 5
GE 45
PF :TAMANHO
GD 135
PF :TAMANHO * 7 / 5
PT :TAMANHO * 7 / 5
END
```

Mostra um quadrado com divisões perpendiculares dando um efeito de quadradinhos.

criado a partir de quatro triângulos.

CAPÍTULO 5

Para a realização dos testes das correções apresentadas sugere-se antes de cada operação executar a opção de menu "**Diversão**" do menu da página da plataforma "*Academia da Tartaruga*".

1. Criar procedimento chamado **CAP0501** que efetue a leitura de um valor numérico inteiro e apresente o resultado do valor lido elevado ao quadrado sem efetuar o armazenamento do resultado em memória. A variável que receberá a entrada do dado deve ser definida como local.

```
APRENDA CAP0501
LOCAL "N
ATRIBUA "N READWORD
ESCREVA (SE "Resultado "= POWER INT :N 2)
END
```

2. Criar procedimento chamado **CAP0502** que efetue a leitura de um valor numérico inteiro e apresente o resultado do valor lido elevado ao cubo com armazenamento do resultado calculado em memória. As variáveis devem ser definidas como local.

```
APRENDA CAP0502
LOCAL "N
LOCAL "R
ATRIBUA "N READWORD
ATRIBUA "R POWER INT :N 3
ESCREVA (SE "Resultado "= :R)
END
```

3. Criar procedimento chamado **CAP0503** que efetue a leitura de uma temperatura em graus Celsius e apresente essa temperatura em graus Fahrenheit, sua conversão. A fórmula de conversão é " $F \leftarrow C * 9 / 5 + 32$ ", sendo "**F**" a temperatura em Fahrenheit e "**C**" a temperatura em Celsius. Armazene em memória o resultado calculado. Use variáveis locais. Formate a saída numérica com duas casas decimais.

```
APRENDA CAP0503
LOCAL "C
LOCAL "F
ATRIBUA "C READWORD
ATRIBUA "F :C * 9 / 5 + 32
ESCREVA (SE [Temperatura em Fahrenheit] "= :F)
END
```

4. Criar procedimento chamado **CAP0504** que efetue a leitura de uma temperatura em graus Fahrenheit apresente essa temperatura em graus Celsius, sua conversão. A fórmula de conversão é "**C ← ((F – 32) * 5) / 9**", sendo "F" a temperatura em Fahrenheit e "C" a temperatura em Celsius. Armazene em memória o resultado calculado. Use variáveis locais. Formate a saída numérica com duas casas decimais.

```
APRENDA CAP0504
LOCAL "F
LOCAL "C
ATRIBUA "F READWORD
ATRIBUA "C ((:F - 32) * 5) / 9
ESCREVA (SE [Temperatura em Celsius] "= :C)
END
```

5. Criar procedimento chamado **CAP0505** que efetue a leitura de dois valores numéricos inteiros (representados pelas variáveis locais "A" e "B") e mostre o resultado armazenado em memória do quadrado da diferença do primeiro valor (variável "A") em relação ao segundo valor (variável "B") junto a variável local "R".

```
APRENDA CAP0505
LOCAL "A
LOCAL "B
LOCAL "R
ATRIBUA "A READWORD
ATRIBUA "B READWORD
ATRIBUA "R POWER (INT :A - INT :B) 2
ESCREVA (SE [Quadrado da diferença] "= :R)
END
```

6. Criar procedimento chamado **CAP0506** que efetue a leitura de um número inteiro qualquer em uma variável local e multiplique este número por "2" armazenando o resultado em memória. Apresentar o resultado da multiplicação somente se o resultado for maior que "30".

```
APRENDA CAP0506
LOCAL "N
LOCAL "R
ATRIBUA "N READWORD
ATRIBUA "R (INT :N) * 2
IF :R > 30 [
    ESCREVA (SE "Resultado" "= :R)
]
END
```

7. Criar procedimento chamado **CAP0507** que efetue a leitura de dois valores numéricos reais representados pelas variáveis locais "A" e "B" e apresente o resultado da diferença do maior valor pelo menor valor sem armazenar o resultado em memória.

```
APRENDA CAP0507
LOCAL "A
LOCAL "B
ATRIBUA "A READWORD
ATRIBUA "B READWORD
IFELSE :A > :B [
    ESCREVA (SE "Resultado "= :A - :B)
][
    ESCREVA (SE "Resultado "= :B - :A)
]
END
```

8. Criar procedimento chamado **CAP0508** que efetue a leitura de dois valores numéricos reais representados pelas variáveis locais "A" e "B" e apresente o resultado da diferença do maior valor pelo menor valor com armazenamento do cálculo em memória.

```
APRENDA CAP0508
LOCAL "A
LOCAL "B
LOCAL "R
ATRIBUA "A READWORD
ATRIBUA "B READWORD
IFELSE :A > :B [
    ATRIBUA "R :A - :B
][
    ATRIBUA "R :B - :A
]
ESCREVA (SE "Resultado "= :R)
END
```

9. Criar procedimento chamado **CAP0509** que efetue a leitura de três valores numéricos inteiros desconhecidos representados pelas variáveis locais "A", "B" e "C". O procedimento deve somar esses valores, armazenar o resultado em memória e apresentar este resultado somente se for "**maior ou igual**" a "**100**".

```
APRENDA CAP0509
LOCAL "A
LOCAL "B
LOCAL "C
LOCAL "R
ATRIBUA "A READWORD
ATRIBUA "B READWORD
ATRIBUA "C READWORD
ATRIBUA "R INT :A + INT :B + INT :C
IF :R >= 100 [
    ESCREVA (SE "Resultado "= :R)
]
END
```

10. Criar procedimento chamado **CAP0510** que apresente a soma dos cem primeiros números naturais "**(1 + 2 + 3 + ... + 98 + 99 + 100)**" utilizando-se a primitiva "**REPITA**".

```
APRENDA CAP0510
  LOCAL "S
  ATRIBUA "S 0
  REPITA 100 [
    ATRIBUA "S :S + CONTEVEZES
  ]
  ESCREVA (SE "Somatório: " = :S)
END
```

11. Criar procedimento chamado **CAP0511** que apresente a soma dos cem primeiros números naturais "**(1 + 2 + 3 + ... + 98 + 99 + 100)**" utilizando-se a primitiva "**WHILE**".

```
APRENDA CAP0511
  LOCAL "S
  ATRIBUA "S 0
  LOCAL "I
  ATRIBUA "I 1
  WHILE [:I <= 100] [
    ATRIBUA "S :S + :I
    ATRIBUA "I :I + 1
  ]
  ESCREVA (SE "Somatório: " = :S)
END
```

12. Criar procedimento chamado **CAP0512** que apresente a soma dos cem primeiros números naturais "**(1 + 2 + 3 + ... + 98 + 99 + 100)**" utilizando-se a primitiva "**FAÇAPARA**".

```
APRENDA CAP0512
  LOCAL "S
  ATRIBUA "S 0
  FAÇAPARA [I 1 100] [
    ATRIBUA "S :S + :I
  ]
  ESCREVA (SE "Somatório: " = :S)
END
```

13. Criar procedimento chamado **CAP0513** que apresente o somatório dos *valores pares* existentes na faixa de "**1**" até "**500**". Use a primitiva "**REPITA**".

```
APRENDA CAP0513
  LOCAL "S
  ATRIBUA "S 0
  REPITA 500 [
    IF (MODULO CONTEVEZES 2) = 0 [
      ATRIBUA "S :S + CONTEVEZES
    ]
  ]
  ESCREVA (SE "Somatório: " = :S)
END
```

14. Criar procedimento chamado **CAP0514** que apresente o somatório dos *valores pares* existentes na faixa de "1" até "500". Use a primitiva "**FAÇAPARA**".

```
APRENDA CAP0514
LOCAL "S
ATRIBUA "S 0
FAÇAPARA [I 1 500] [
    IF (MODULO :I 2) = 0 [
        ATRIBUA "S :S + :I
    ]
]
ESCREVA (SE "Somatório: " = :S)
END
```

15. Criar procedimento chamado **CAP0515** que mostre todos os valores numéricos divisíveis por "4" *menores* que "20" iniciando a contagem em "1". Use a primitiva "**REPITA**".

```
APRENDA CAP0515
REPITA 19 [
    IF (MODULO CONTEVEZES 4) = 0 [
        ESCREVA CONTEVEZES
    ]
]
END
```

16. Criar procedimento chamado **CAP0516** que mostre todos os valores numéricos divisíveis por "4" *menores* que "20" iniciando a contagem em "1". Use a primitiva "**FAÇAPARA**".

```
APRENDA CAP0516
FAÇAPARA [I 1 19] [
    IF (MODULO :I 4) = 0 [
        ESCREVA :I
    ]
]
END
```

17. Criar procedimento chamado **CAP0517** que apresente os valores numéricos inteiros compreendidos na faixa de "0" e "4". Use a primitiva "**REPITA**".

Contagem de "0" a "4" com REPITA não é possível, pois a primitiva inicia a sua

ação sempre em "1", a menos que efetue o seguinte ajuste (não muito bom):

```
APRENDA CAP0517
  LOCAL "I"
  ATRIBUA "I 0"
  REPITA 5 [
    ESCREVA :I
    ATRIBUA "I :I + 1"
  ]
END
```

Veja que a solução não é muito boa pois necessitou de uma variável auxiliar "I"

dando um estilo de "ENQUANTO" ou "FAÇA.ATÉ". Então é melhor usar as primitivas

"ENQUANTO" ou "FAÇA.ATÉ" e resolver o problema de forma mais adequada.

18. Criar procedimento chamado **CAP0518** que apresente os valores numéricos inteiros compreendidos na faixa de "0" e "4". Use a primitiva "**WHILE**".

```
APRENDA CAP0518
  LOCAL "I"
  ATRIBUA "I 0"
  WHILE [:I <= 4] [
    ESCREVA :I
    ATRIBUA "I :I + 1"
  ]
END
```

19. Criar procedimento chamado **CAP0519** que apresente os valores numéricos inteiros compreendidos na faixa de "0" e "4". Use a primitiva "**FAÇAPARA**".

```
APRENDA CAP0519
  FAÇAPARA [I 0 4] [
    ESCREVA :I
  ]
END
```

20. Criar procedimento chamado **CAP0520** que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use a primitiva "REPITA".

Situação semelhante ao exercício 17. Só que aqui não vale nem a pena tentar fazer

a solução. O "REPITA" além de iniciar em "1" executa o salto de contagem sempre de

"1" em "1".

21. Criar procedimento chamado **CAP0521** que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use a primitiva "WHILE".

```
APRENDA CAP0521
LOCAL "I"
ATRIBUA "I" 0
WHILE [:I <= 15] [
    ESCREVA :I
    ATRIBUA "I" :I + 3
]
END
```

22. Criar procedimento chamado **CAP0522** que apresente os valores numéricos inteiros compreendidos na faixa de "0" até "15" de "3" em "3". Use a primitiva "FAÇAPARA".

```
APRENDA CAP0522
FAÇAPARA [I 0 15 3] [
    ESCREVA :I
]
END
```

ANOTAÇÕES

Bibliografia

- ABELSON, H. **TI Logo**. New York: McGraw-Hill, 1984.
- _____. **TI Logo Education**. Lubbock: Texas Instruments & McGraw-Hill, 1981.
- ATARI. **ATARI Logo: Reference Manual**. Quebec: Logo Computer System, Inc., 1983.
- BASS, J. H. **Logo Programming**. Logo Spoken Here. 2002. Disponível em: <<http://pages.intnet.mu/jhbpage/main.htm>>. Acesso 28 jun. 2021.
- CATLIN, D. **My Personal Tribute to Seymour Papert**. GO Magazine. August, 2016. Disponível em: <<http://go.roamer-educational-robot.com/2016/08/12/my-personal-tribute-to-seymour-papert/>>. Acesso 26 jun. 2021.
- CORRALES MORA, M. **Lenguage Logo I: Descubriendo un mundo nuevo**. San José: EUNED, 1996.
- DOWNEY, A. B. & GAY, G. **How to think like a Computer Scientist: Logo version**. GNU Edition, 2003. Disponível em: <<http://openbookproject.net/thinkcs/archive/logo/english/thinklgo.pdf>>. Acesso 29 jun. 2021.
- ERFAN'S. **Welcome to Erfan's Zone of Logo**. Nestead. 2021. Disponível em: <<http://erfan96.50webs.com/>>. Acesso 28 jun. 2021.
- HARVEY, B. **Computer science Logo style: Symbolic computing**. 2nd. Massachusetts: MIT Press. 1998, v. 1.
- JOYS, D. **Hs-logo**. Logo Turtle Graphics Interpreter. 2021. Disponível em: <<https://deepakjois.github.io/hs-logo/>>. Acesso 28 jun. 2021.
- KHERIATY, I. & GERHOLD, G. **Radio Shack color Logo**. Massachusetts: Micropi, 1982.
- LOGO FOUNDATION. **What Is Logo?**. New York: Logo Foundation, 2021. Disponível em: <https://el.media.mit.edu/logo-foundation/>. Acesso em: 16 jun. 2021.
- MULLER, J. **The Great Logo Adventure: Discovering Logo on and Off the Computer**. Madison, AL, United States: Doone Pubns, 1998.
- PALEOTRONIC. **Past and Future Turtles - The Evolution of the Logo Programming Language: Part I**. Australia: Paleotronic Magazine, 2021. Disponível em: <https://paleotronic.com/2021/05/22/past-and-future-turtles-the-evolution-of-the-logo-programming-language-part-1/>. Acesso em: 16 jun. 2021.
- PETTI, W. A. **Math Cats**. Disponível em: <<http://www.mathcats.com/>>. Acesso em 23 jun. 2021.
- SOLOMON, C. J. (Et al). **History of Logo**. Proc. ACM Program. Lang. 4, HOPL, Article 79. June, 2020. Disponível em: <<https://dl.acm.org/doi/pdf/10.1145/3386329>>. Acesso em 28 jun. 2021.
- SPARER, E. **Sinclair Logo 1 Turtle Graphics**. Cambridge: Sinclair Research Ltd., 1984.
- TOMIYAMA, M. N. **Recursão**. Minas Gerais: Universidade Federal de Uberlândia - Faculdade de Computação, 2016. Disponível em: <<http://www.facom.ufu.br/~madriana/PF/recursao.pdf>>. Acesso em 22 jun. 2021.
- WINTER, M. J. **The Commodore 64 Logo workbook**. Chatsworth: DATAMOST, 1984.

Referências bibliográficas

APPLE. **Apple Logo II: Reference manual.** California: Apple Computer, Inc. and Quebec: Logo Computer System. Inc. 1984.

ATARI. **ATARI Logo: Introduction programming through turtle graphics.** Quebec: Logo Computer System. Inc. 1983.

GOLDBERG, K. P. **Learning Commodore 64 Logo together.** Washington: Microsoft Press, 1984.

MANZANO, J. A. N. G. **Linguagem Logo: Programação de computadores - princípios de inteligência artificial.** São Paulo: All Print. 2012.

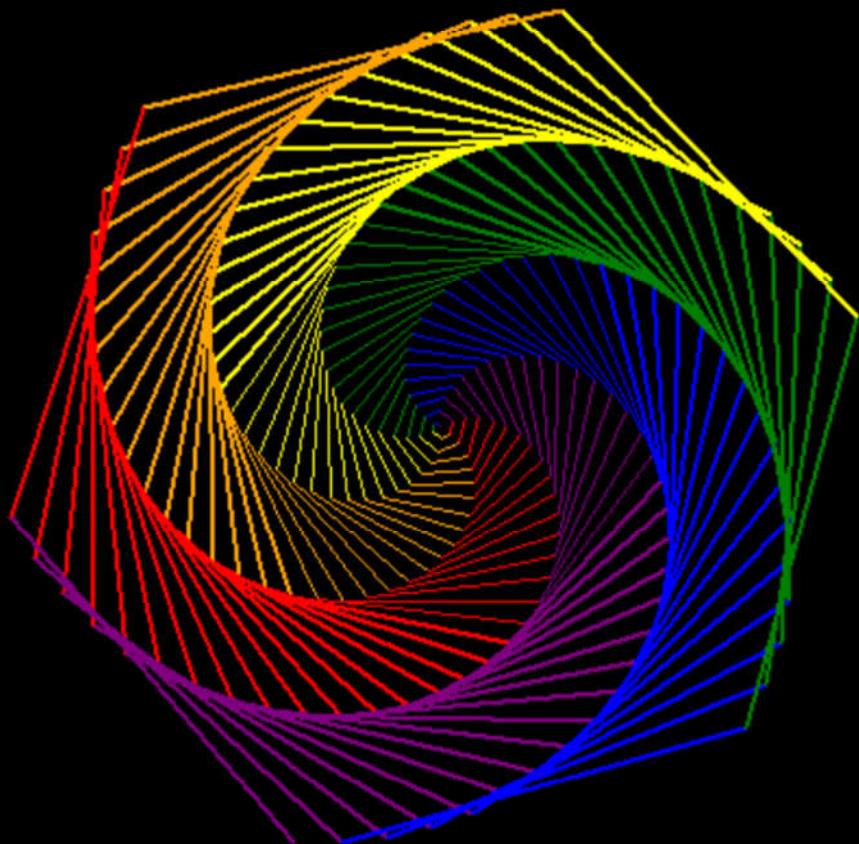
SASSENRATH, C. **Amiga LOGO: Tutorial and reference.** Commodore-Amiga, Inc. and Carl Sassenrath, 1989.

SOLOMON, C. J. **Apple Logo: Introduction programming through turtle graphics.** Quebec: Logo Computer System. Inc. 1982.

ANOTAÇÕES

ACADEMIA DA TARTARUGA

Programação em Logo



ESTE LIVRO MOSTRA A LINGUAGEM "LOGO" DE MANEIRA DINÂMICA E PRÁTICA A PARTIR DE EXEMPLOS DE APLICAÇÃO FOCADOS NOS PRINCÍPIOS DE LÓGICA DE PROGRAMAÇÃO.

A LINGUAGEM "LOGO" FOI CRIADA NA DÉCADA DE 1960 POR UMA EQUIPE MULTIDISCIPLINAR DE ESPECIALISTAS EM EDUCAÇÃO E INTELIGÊNCIA ARTIFICIAL DIRIGIDA PELO PROFESSOR SEYMOUR PAPERT.

NESTE TRABALHO A LINGUAGEM É APRESENTADA A PARTIR DO PONTO DE VISTA DAS AÇÕES DA GEOMETRIA DA TARTARUGA E DE AÇÕES BASEADAS NO DESENVOLVIMENTO TRADICIONAL DE PROGRAMAS COMO FAZEM OUTRAS LINGUAGENS DE PROGRAMAÇÃO.

