Lógica de Programação para não programadores

Desenvolvido por Fernando Anselmo
Versão 1.0

Introdução

Lógica de programação é a maneira como as ideias são encadeadas para permitir a escrita de um programa de computador, um algoritmo. Um algoritmo é uma sequência de passos para se executar uma função. Saber lógica de programação é saber o melhor jeito de escrever um código, para o computador interpretar corretamente. É saber se comunicar com a maquina a partir de uma linguagem seja lá qual for.

Scratch é um programa com o objetivo de ensinar lógica de programação em blocos, foi criada em 2007 pelo Media Lab do MIT. É possível criar histórias animadas, jogos e outros programas interativos. Seus objetivos principais são: criar uma linguagem mais suscetiva à manipulação, ser mais social e mais significante. Assim a forma como os blocos podem ser manipulados lhe confere uma possibilidade rápida de aprendizagem através da prática de manipulação e teste dos projetos.

Sonic Pi é um ambiente de codificação para interpretação de performances musicais ao vivo, foi criado para apoiar tanto aulas de lógica de informática quanto aulas de música nas escolas. Foi desenvolvido por *Sam Aaron* no Laboratório de Informática da Universidade de Cambridge em colaboração com Raspberry Pi Foundation. Utiliza um mecanismo de síntese Supercollider e modelo cronometragem precisa, sendo possível ser utilizado para codificação ao vivo e outras formas de performance musical algorítmica e de produção.

VisuALG

VisuAlg é um programa que edita, interpreta e executa algoritmos com uma linguagem próxima do português estruturado como um programa normal de computador. É um programa de livre uso e distribuição, empregado no ensino de programação em várias escolas e universidades no Brasil e no exterior, sendo um bom recurso para quem está iniciando no aprendizado de algoritmos, não só para praticar a sua criação mas também para melhor entender sua execução por meio do visualizador de variáveis que funciona como um depurador.

A lógica de programação é necessária não apenas para pessoas que desejam trabalhar com desenvolvimento de sistemas e programas. Saber como ordenar a informação e as ações em um fluxo lógico é essencial para aprender qualquer matéria.

Os blocos de programação

Neste capítulo, vamos usar o **Scratch** para entender como funciona os conceitos de conjuntos de informações, sensores, variáveis e blocos de decisão e repetição.
Scratch é uma linguagem programação que foi especialmente concebido para fazer animações e jogos com facilidade. Um ambiente de programação simples e intuitivo. Recomendado para ser usada por principiantes, jovens ou adultos, que queiram iniciar-se no mundo da programação de computadores, ganhando gosto e asas para voos mais altos noutras linguagens mais poderosas e profissionais.
A tela inicial do Scratch é composta pelas seguintes áreas:

- Blocos de Comandos, que apresenta e possibilita a escolha dos grupos de comandos desta linguagem de programação. Estão dispostos em dez caixas: Movimento, Aparência, Som, Caneta, Variáveis, Eventos, Controle, Sensores, Operadores e Mais Blocos. Cada caixa possui uma cor específica para facilitar a visualização.
- Edição, que possibilita a criação do projeto, ou a programação de eventos (ou "scripts").
- Bastidores, é uma lista miniaturas dos atores que estão sendo utilizados no projeto.
- Apresentação, que viabiliza a execução do projeto criado.
- Palco, para definição dos atores (ou "sprites[1]") e cenários que integram um dado projeto.

Movimento e Interação com Atores

Quando o Scratch é aberto, no **Palco** já aparece o gato (este é conhecido como **Ator**). É possível inserir ou criar um novo ator. Da mesma forma, é possível ter vários atores no palco.
Para determinar o que cada ator executará, devemos criar uma sequencia de comandos, arrastando blocos gráficos e juntando-os em pilhas chamadas de **Script**. Basta um duplo clique em qualquer bloco para fazer rodar um script.

Prática

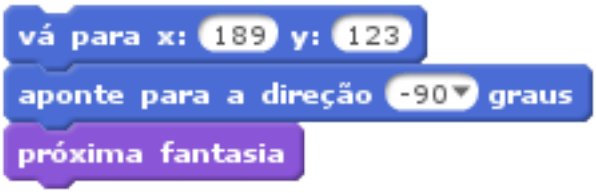
Vamos criar um script para mostrar como é fácil produzir um resultado visível. Selecionar a caixa "Eventos" e arrastar o bloco "Quando clicar em 'bandeira verde'" para a área de edição. Selecionar a caixa "Controle" e arrastar o bloco "repita 10 vezes" para a área de edição grudando-o no bloco anterior. Selecionar a caixa "Movimento" e arrastar o bloco "Girar 'direita' 15 graus" para a área de edição colocando-o dentro do bloco anterior. O resultado final será este:

1 Em computação gráfica, um **sprite** (do latim spiritus, que significa "duende", "fada") é um objeto gráfico que se move numa tela sem deixar traços de sua passagem (como se fosse um "espírito") e foi inventado como um método rápido de animação de várias imagens agrupadas numa tela para jogos de computador.

Pode-se adicionar um comando no meio do bloco de comandos, mas é preciso ter cautela pois, depois de inserido, já não sai ao puxá-lo. Para o retirar, é preciso puxar o que está debaixo e dividir o bloco em duas partes; depois é que se puxa por ele (porque fica a ser o último da primeira parte do bloco).
Para mover uma pilha basta arrastar o bloco de cima. Arrastando um bloco do meio todos os de baixo são arrastados juntos. Para copiar uma pilha de blocos, de um ator para outro, basta arrastá-la para a miniatura do outro ator nos bastidores.

Exercício 1

1.1. Fazer um ator se movimentar por todo o cenário.
1.2. Fazer o movimento parecer mais real trocando o traje do ator.
1.3. Fazer o ator se movimentar até um ponto específico do cenário.
1.4. Atente-se no bloco abaixo, o que este bloco produz?



Dica: Para descobrir o que faz cada bloco basta clicar sobre ele com o botão direito do mouse e selecionar Ajuda no menu que aparece.

A aparência de um ator pode ser modificada pela sua apresentação em um traje diferente. Qualquer imagem pode ser usada como um traje; pode-se desenhar essa imagem no editor de Pintura, importar de uma lista ou baixá-la de um site na internet. O Scratch reconhece os formatos de imagem: JPG, BMP, PNG, GIF (animado também). A ordem dos trajes pode ser modificada arrastando-se as miniaturas. Clicar com o botão direito do mouse em uma miniatura de traje o transforma em um novo ator.
Para realizar uma animação o efeito é o mesmo princípio de uma imagem GIF, onde aparecem diferentes posições de um personagem e a troca das imagens das posições produz a ideia de animação. Escolha o objeto que será animado e clique em Trajes. É possível criar as diferentes posições do objeto desenhando o novo a partir do inicial (fazer uma cópia do original e editar) ou importar as posições.
Ao realizar movimentos, é importante que o ator ao tocar na borda do palco volte. Um exemplo disso pode ser uma bola que role, bate na borda e volte.
Para fazer um ator falar. Para isto, basta usar o bloco de comando **Diga**. Nele podemos determinar o que será dito e o tempo que essa mensagem aparecerá. Além disso podemos utilizar sons, podem ser gravados novos sons

ou importados de arquivos de som. Scratch pode ler arquivos MP3, WAV não zipados, AIF e AU (desde que não sejam de 24 bits).

Trabalho com Variáveis

Variável é uma representação simbólica de elementos de um certo conjunto. Cada variável corresponde a uma posição de memória, cujo conteúdo pode se alterado ao longo do tempo durante a execução de um programa. Embora uma variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante.
Variável é um pedacinho de memória onde podemos guardar um valor qualquer, para ser usado nas operações. Já existem algumas variáveis do Scratch, como por exemplo as posições X e Y dos atores.
É possível criar quaisquer tipo de variáveis que necessitamos, abrindo o bloco das "Variáveis" e clicando em "Criar uma variável". Basta dar um nome para ela e podemos utilizá-la para compormos nossos scripts.

Exercício 2

2.1. Receber um valor do usuário. Armazenar este valor na variável "resposta". Fazer o ator dizer o valor que foi armazenado.
2.2. Fazer que o ator peça um número, e mostrar o dobro do número.
2.3. Ler 2 números e mostrar a soma deles na tela. Modificar para fazer outras operações básicas: multiplicar, dividir e subtrair.

Sensores

O Scratch também possui controles para o início da execução dos scripts. Um exemplo é a **Bandeira Verde** que fica sobre a tela de visualização das programações. Pode ser usada para iniciar o funcionamento de um script. Para isso é necessário que seja colocado no script o bloco de controle que indica.
Blocos de programação são conjuntos agrupados de instruções e declarações que têm um escopo com uma ação definida. Existem diversos tipos de blocos que são organizados em delimitadores, de movimentação e condicionais.

- Os delimitadores são blocos exclusivamente usados para delimitar o início e o término de um programa em blocos.
- Os de movimentação são os que possuem algum comando de ação para o ator.
- E, os condicionais são os que fazem a verificação de uma certa condição para determinar se outros blocos a eles submetidos serão avaliados e executados ou se serão desconsiderados.

Para iniciar um script, além de usar a bandeira verde, é possível determinar uma tecla do teclado que funcione como disparadora do script. Desta forma, quando a tecla for pressionada, o script inicia sua execução.
Para determinar que o início da execução será determinado por uma tecla, é necessário colocá-la no início de um script o controle.

Prática

Atente-se no bloco abaixo, que deve ser inserido para que este seja executado quando clicarmos em do ator?

DICA: Os comandos amarelos são blocos "Controle" (os roxos blocos de "Aparência" e os azuis blocos de "Movimento").
Na caixa "Sensores" existem ferramentas para avaliar quando um botão do mouse ou uma tecla do teclado são pressionados. Mas esta ocasião também é boa para usar mais uns comandos novos.

Blocos de Decisão e Repetição

Para muitas programações, jogos e histórias é importante usar testes. Podemos fazer uma bola bater em um objeto e quando ela bater, voltar. Mas como ela vai saber que bateu? Como determinar o que acontece quando ela bate.

Prática

Atente-se ao bloco da prática anterior, que deve ser inserido para que o gato mie sempre que uma determinada tecla seja pressionada.
Como vimos, os comandos de um bloco são cumpridos pela ordem em que se encontram. Isso é chamado "Fluxo Linear", porque uma ação começa na ponta de cima e termina na de baixo. Mas, por vezes, há necessidade de repetir um conjunto de instruções ou de optar entre dois conjuntos de blocos.

Exercício 3

3.1. Fazer com que um ator fale dez números inteiros na sequencia a partir do 1.
3.2. Pedir um número e mover em passos o ator para a esquerda e direita.
3.3. Pedir um número e se este for par fazer o ator mover 10 passos para frente, caso seja ímpar fazer o ator mover 10 passos para trás.
3.4. Por 3 vezes fazer um ator ator tocar um instrumento qualquer a seguinte sequencia de notas por 0.2 batidas: 60, 61, 62 e 61.

Criar música no computador

Neste capítulo, vamos usar o **Sonic PI** para entender como é a diferença entre Laços e Serviços, Sintetizadores e Samples, o que é a programação em sub-blocos e os números aleatórios.
Sonic Pi é um ambiente de programação de código aberto, projetado para criar novos sons com código em um ambiente de codificação ao vivo; foi desenvolvido pelo Dr. Sam Aaron da Universidade de Cambridge.
A interface do Sonic Pi possui três áreas principais. A maior delas é para escrever seu código, é chamado de **Painel de Programação**. No painel de saída são mostradas as informações de execução do seu programa e como está sendo executado. Ao clicar no botão de ajuda na parte superior da janela, o terceiro painel no fundo da janela é visualizado, em forma de documentação de auxílio. Esta última área contém várias informações sobre diferentes códigos que podemos utilizar, bem como modificar para produzir diferentes sons de sintetizadores, samples, e muito mais.
Na área de programação utilize o comando:

```
play 60
```

Clique no ícone de reprodução na parte superior da tela. O que acontece? Agora o que acontece se você digitar **pley 60** e clicar no ícone de reprodução?

Este é um exemplo de um erro em seu código. Nas atividades posteriores, se o painel de erro for exibido você tem um erro que é necessário corrigir.
Agora tentemos a seguinte sequencia:

```
play 60
play 67
play 69
```

Clique no ícone de reprodução na parte superior da tela. O que acontece? O computador está tocando cada nota em sequência (uma após a outra), mas está acontecendo tão rápido que soam como se estivessem tocando todas ao mesmo tempo.
Outra forma de tocarmos as notas e através de variáveis prontas que representam seus números, por exemplo:

```
play :C4
```

Isto pode ser representado pela seguinte tabela:

Número da Oitava	Nota											
	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0	12	13	14	15	16	17	18	19	20	21	22	23
1	24	25	26	27	28	29	30	31	32	33	34	35
2	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59
4	60	61	62	63	64	65	66	67	68	69	70	71
5	72	73	74	75	76	77	78	79	80	81	82	83
6	84	85	86	87	88	89	90	91	92	93	94	95
7	96	97	98	99	100	101	102	103	104	105	106	107
8	108	109	110	111	112	113	114	115	116	117	118	119
9	120	121	122	123	124	125	126	127				

Precisamos realizar uma pausa entre cada nota. Podemos fazer isso ao digitar a seguinte depois de cada comando **play**:

```
sleep 1
```

O valor inserido após a palavra **sleep** representa um tempo medido em segundos. O que usaríamos para representar meio segundo?

Exercício 4

4.1. A música tema do jogo Super Mário é conseguida com a seguinte sequencia de notas: :E4, :E4, :E4, :C4, :E4, :G4, :G3, :C4, :G3, :E3, :A3, :B3, :As3, :A3, :G3, :E4, :G4, :A4, :F4, :G4, :E4, :C4, :D4 e :B3. Execute-as com o Sonic Pi.

4.2. Produza uma música conhecida.

Diferença entre Laços e Serviços

No exemplo abaixo, podemos perceber que algumas linhas de código estão recuadas. Isto torna mais fácil de ler o código e verificar se existe algum erro.

```
2.times do
  play 60
  sleep 0.5
  play 62
  sleep 0.5
  play 64
  sleep 0.5
  play 60
  sleep 0.5
end
```

Laços notas por um determinado número de vezes é extremamente útil para evitarmos excesso de linhas de código. Mas é se desejamos fazer um laço que dure para sempre?

Podemos utilizar um bloco como este:

```
loop do
  play 60
  sleep 0.5
end
```

Este bloco só terminará quando interrompemos o programa. Mas para que serve isso? Qual a utilidade de usarmos laços infinitos?

Antes de respondermos a essa questão, precisamos ver um outro modelo de bloco infinito chamado **live_loop**. Ao tocarmos o bloco anterior e pressionarmos várias vezes o botão Executar (Run) o que acontece?
Isso acontece porque o programa é chamado novamente criando uma nova execução dos serviços, então vamos trocar para:

```
live_loop :melodia do
  play 60
  sleep 0.5
end
```

Agora pressione o botão Run e observe que não existe mais a sobreposição. Observe que este bloco possui um nome (neste caso :melodia), assim o Sonic Pi reconhece este e a cada vez que pressionamos o botão Run este é sobreposto e não mais criado um novo bloco.

Prática

Blocos **live_loop** são muito interessantes pois podemos alterá-los em tempo de execução. Ou seja, pressione o botão Run modifique o valor da nota e pressione novamente o botão Run.

Sintetizadores e Samples

Outro detalhe interessante do Sonic Pi é a quantidade de sintetizadores que podemos trabalhar, por exemplo tente o seguinte código:

```
use_synth :fm
2.times do
  play 60
  sleep 0.5
  play 67
  sleep 0.5
end
```

Qual foi o som produzido?

É possível não apenas criar música com Sonic Pi usando notas simples, também pode-se criar música com samples (amostras). Samples são sons ou músicas pré-gravadas que podemos adicionar para nossa composição musical. Esta é uma maneira muito simples de fazer sua música soar incrível.

```
2.times do
  sample :loop_amen
  sleep 1.753
end
```

Há vários tipos de samples já incluídos com Sonic Pi. Para encontrar os nomes deles, clique na ajuda seguido de Samples no lado esquerdo da janela de ajuda. Clique em qualquer um dos nomes de exemplo para obter mais informações sobre como usá-lo.

Prática

Música possuem faixas de apoio que se repetem. Até agora, brincamos com uma única melodia. Vamos tentar usar duas músicas ao mesmo tempo!
Clicar em uma nova aba buffer. Digite o seguinte trecho de código:

```
live_loop :laco1 do
  sample :loop_amen
  sleep 1.753
end
```

Este primeiro laço atuará como a base da música. Aperte o botão Run e agora digite a seguinte faixa de apoio:

```
live_loop :laco2 do
  play 65
  sleep 0.753
  play 64
  sleep 1
end
```

Agora aperta o botão Run novamente e devemos ouvir ambas as faixas tocando ao mesmo tempo.

Programando sub-blocos

Uma forma interessante em se trabalhar com programação é a criação de pequenos trechos de blocos para que possam ser executados organizadamente por outro bloco principal. Sozinhos esses blocos são apenas partes de um todo, como por exemplo, sua mão ou sua perna.
Para criar um sub-bloco digite os seguintes comandos:

```
define :meuSynth do
  use_synth :prophet
```

```
  play 53, attack: 0.2, release: 1.3
  sleep 0.5
end
```

O que acontece se pressionarmos o botão Run? Porque isso aconteceu?

Para chamar este sub-bloco devemos criar um bloco principal:

```
live_loop :meuBloco do
  play_my_synth
end
```

Agora o que acontece quando apertamos o botão Run? Porque isso aconteceu?

A vantagem em se utilizar de sub-blocos é que podemos deixar nosso código mais organizado dividindo um programa que teria muitas linhas e seria de difícil manutenção em algo mais simples e bem organizado.

Números aleatórios

Sonic PI inclui várias funções interessantes para compor peças musicais, porém um bom efeito é tentar algo totalmente novo (inclusive inesperado), imagine que tem na mão uma caixa e nessa existem notas musicais, aleatoriamente selecionamos uma nota e devemos tocá-la.
Veja o seguinte código:

```
live_loop :seqLa do
  play chord(:a3, :minor).choose
  sleep 0.2
end
```

O acorde Lá Menor é formado pelas notas 57 (:A3), 60 (:C4) e 64 (:E4), em cada passada do laço uma dessas notas é escolhida (choose) e tocada.

Agora vamos proceder uma outra modificação:

```
live_loop :seqLa do
  play chord(:a3, :minor).choose
  sleep rrand(0, 1)
end
```

Agora além de ser escolhida uma nota ao acaso o tempo de parada será também escolhido com um valor entre 0 e 1.

Exercício 5

5.1. Sortear um número entre 1 e 10 (inteiro), tocar o resultado vezes a seguinte sequencia de notas: 53, parar por 0.9 segundos, 55 parar por 0.3 segundos, 53 parar por 0.6 segundos e 50 parar por 1.8 segundos.

Aprendizado Extra

Porque o aprendizado deve ser chato? Se divirta com o programa. Comece por digitar cada um desses trechos de código com o Sonic PI e veja o que eles produzem.

A. Batida Simples

```
live_loop :ondrum do
  cue :ondrums
  sample :loop_amen, rate: 0.75
  sleep sample_duration :loop_amen, rate: 0.75
end
live_loop :guit do
  sync :ondrum
  sample :guit_e_fifths, rate: 0.6
  sleep sample_duration :guit_e_fifths
end
```

B. Drumminbeis

```
live_loop :amen_break do
  use_bpm 80
  with_fx :slicer,
    phase: 0.5,
  wave: 0, mix: 1 do
    sample :loop_amen, beat_stretch: 2, cutoff: 100
  end
  sleep 2
end
```

C. Eletrônica

```
use_bpm 60
use_debug false

live_loop :amen_break do
  p = [0.125, 0.25, 0.5].choose
  with_fx :slicer, phase: p, wave: 0, mix: rrand(0.7, 1),
  reps: 4 do
    r = [1, 1, 1, -1].choose
    sample :loop_amen, beat_stretch: 2, rate: r , amp: 2
    sleep 2
  end
end
live_loop :bass_drum do
  sample :bd_haus, cutoff: 70, amp: 1.5
  sleep 0.5
end
```

```
live_loop :landing do
  bass_line = (knit :e1, 3, [:c1, :c2].choose, 1)
  with_fx :slicer, phase: [0.25, 0.5].choose,
  invert_wave: 1, wave: 0 do
    s = synth :square, note: bass_line.tick, sustain: 4,
      cutoff: 60
    control s, cutoff_slide: 4, cutoff: 120
  end
  sleep 4
end
```

D. Disco

```
use_bpm 80

live_loop :beat do
  sample :bd_haus, amp: 2
  sleep 1
end

hat_pattern = [9, 0, 5, 5, 0, 0, 5, 0, 9, 0, 5, 5, 0, 0, 9, 0]

live_loop :hat do
  sync :beat
  hat_pattern.each do |p|
    sample :drum_cymbal_closed, amp: p/6.0, rate: (rrand(0,0.05)+0.5+
(p/9.0)), start: rrand(0,0.05)
    sleep 0.25
  end
end

sleep 8

live_loop :drum do
  sync :beat
  sample :loop_industrial, beat_stretch: 1, rate: 1
  sleep 2 # Mude para 1 no Refrão
end
```

Modifique-os e gere sua própria música.

Algoritmos estruturados

Neste capítulo, vamos usar o **VisuAlg** para entender como funciona os fluxos lógicos de informação, tipos de dados, mais sobre os comandos de decisão e repetição e entrada e saída de dados.
A linguagem de programação é dizer como se escreve um algoritmo. O grande problema para muitos é o que "dizer" para o computador fazer. Para o aprendizado foi desenvolvido o Software VisualG, que auxilia a programação totalmente em português.
A tela do VisuAlg compõe-se da barra de tarefas, do editor de textos (que toma toda a sua metade superior), do quadro de variáveis (no lado esquerdo da metade inferior), do simulador de saída (no correspondente lado direito) e da barra de status.
O formato básico de um programa é o seguinte:

```
algoritmo "semnome"
// Função :
// Autor :
// Data :
var
  // Seção de Declarações
inicio
  // Seção de Comandos
fimalgoritmo
```

A primeira linha é composta pela palavra-chave **algoritmo** seguida do seu nome delimitado por aspas duplas. Este nome é usado como título nas janelas de leitura de dados, todas as informações dispostas aqui são para fins de documentação. A seção que se segue é a de declaração de variáveis, iniciada com a palavra-chave **var** e termina com na linha que contém a palavra-chave **inicio**. Deste ponto em diante está a seção de comandos, que continua até a linha em que se encontre a palavra-chave **fimalgoritmo**. Esta última linha marca o final do programa.

Sugestão de Digitação

Uma sugestão de digitação é disponibilizada através das teclas Ctrl+J. Essa combinação de teclas fará que o VisuAlg mostre uma lista com sugestões de palavras-chave que completam o que foi digitado. Para escolher, é necessário dar um duplo clique sobre a opção desejada, ou então selecioná-la com as setas e pressionar Enter. Se o usuário continua escrevendo depois que o VisuAlg apresentou a lista de sugestões, o programa continuará procurando palavras que ainda complementem o que foi digitado. Ao se teclar Esc ou clicar "fora da lista", esta desaparece.

Prática

Este é um exemplo de algoritmo, que tem como objetivo somar 3 números inteiros:

```
algoritmo "soma"
var
  num1, num2, num3, resultado: inteiro
inicio
  escreval("Pograma para somar 3 números inteiros a escolha:")
  escreval("Digite um numero:")
  leia(num1)
  escreval("Digite um numero:")
  leia(num2)
  escreval("Digite um numero:")
  leia(num3)
  resultado <- num1 + num2 + num3
  escreval("O resultado é: ")
  escreval(resultado)
fimalgoritmo
```

Aqui estão presentes alguns elementos que são as palavras-chaves específicas da linguagem, tais como: **var**, **leia** ou **escreval**. Essas palavras-chaves têm funções especificas, e um dos objetivos da programação é entender como funcionam. Cada linguagem tem um correspondente a estes comandos, com a mesma função, porém escrita de forma e palavras diferente (sintaxe da linguagem).

Tipos de Dados

Na hora de programar alguns passos são indispensáveis, como por exemplo a declaração de variáveis. **Variáveis** são escritas por letras ou números, que representam um valor que pode ser mudado a qualquer momento.
As variáveis só podem armazenar valores de um mesmo tipo e cada variável possui um espaço na memória para armazenar seus dados.
Existem os seguintes tipos de dados no VisuAlg:

- **inteiro**: para o armazenamento de números sem casas decimais, positivo ou negativo.
- **real**: para o armazenamento de números que possuam casas decimais, podem ser positivos ou negativos.
- **caractere**: para dados que contenham letras e/ou números, são os textos. Qualquer numero pode entrar aqui, porem não terá função matemática.
- **logico**: armazenam somente dados lógicos que pode ser Verdadeiro ou Falso.

Todas as variáveis que serão utilizadas pelo VisuAlg devem ser definidas na seção **var**. Por exemplo:

```
algoritmo "variaveis"
var
```

```
 a: inteiro
 valor: real
 vet: vetor [1..10] de real
 matriz: vetor [0..4,8..10] de inteiro
 nome: caractere
 professor: logico
inicio
 ...
fimalgoritmo
```

Para atribuirmos o valor a uma variável usamos o sinal **<-** do seguinte modo:

```
algoritmo "variaveis"
var
 ...
inicio
 a <- 3
 valor <- 1.5
 vet[1] <- vet[1] + (a * 3)
 matriz[3,9] <- a/4 - 5
 nome <- "Fernando"
 professor <- VERDADEIRO
fimalgori
```

Para pedir ao usuário a informação de um valor usamos o comando **leia** e para mostrar a informação na tela o comando **escreval**, observe:

```
algoritmo "mostrarnum"
var
  num: inteiro
inicio
  escreval("Digite um número:")
  leia(num)
  escreval("O número digitado foi: ", num)
fimalgoritmo
```

O que faz o programa acima? Execute-o e veja o resultado.

Estrutura de Decisão

As estruturas de decisão são desvios que conduzem o fluxo do programa para um determinado conjunto de comandos baseados em uma decisão.

```
se <expressão lógica> entao
 <sequência A>
senao
 <sequência B>
```

```
fimse
```

Ao encontrar este comando, o compilador analisa a <expressão lógica>. Se o resultado for VERDADEIRO, então todos os comandos da <sequência A> são executados. Se o resultado for FALSO, o algoritmo passa a ser executado a partir da linha depois do **senao**, e todos os comandos da <sequência B> são executados.

Exercício 6

6.1. Elaborar um programa que leia um número. Se positivo armazene-o em A, se for negativo, em B. No final mostrar o resultado.

6.2. Elaborar um programa que leia uma variável numérica e mostrá-la somente se seu valor for maior que 100, caso contrário mostrá-la com o valor zero.

6.3. Calcular a média final de um aluno, considerando que este realizará quatro provas: P1, P2, P3 e P4.

Estruturas de Repetição

O VisuAlg implementa as três estruturas de repetição usuais nas linguagens de programação.

A. Laço determinado: **para...ate...faca**

Este comando repete uma sequência de comandos um determinado número de vezes. Possui a seguinte sintaxe:

```
para <variável> de <valor inicial> ate <valor limite> [passo
<incremento>] faca
 <sequência de comandos>
fimpara
```

Onde:

- <*variável*> É a variável contadora que controla o número de repetições do laço. Na versão atual, deve ser necessariamente uma variável do tipo inteiro, como todas as expressões deste comando.
- <*valor inicial*> É uma expressão que especifica o valor de inicialização da variável contadora antes da primeira repetição do laço.
- <*valor limite*> É uma expressão que especifica o valor máximo que a variável contadora pode alcançar.
- <*incremento*> É opcional. Quando presente, precedida pela palavra passo, é uma expressão que especifica o incremento que será acrescentado à variável contadora em cada repetição do laço. Quando esta opção não é utilizada, o valor padrão de <incremento> é 1.

No exemplo a seguir, os números de 1 a 10 são exibidos em ordem crescente.

```
algoritmo "Números de 1 a 10"
var
  j: inteiro
inicio
  para j de 1 ate 10 faca
    escreva (j:3)
  fimpara
fimalgoritmo
```

B. Laço indeterminado: **enquanto...faca**

Este comando repete uma sequência de comandos enquanto uma determinada condição for satisfeita. Possui a seguinte sintaxe:

```
enquanto <expressão lógica> faca
 <sequência de comandos>
fimenquanto
```

Onde:

- <expressão lógica> Esta expressão que é avaliada antes de cada repetição do laço. Enquanto seu resultado for VERDADEIRO a <sequência de comandos> é executada.

O mesmo exemplo anterior pode ser resolvido com esta estrutura de repetição:

```
algoritmo "Números de 1 a 10 (com enquanto...faca)"
var
  j: inteiro
inicio
  j <- 1
  enquanto j <= 10 faca
    escreva (j:3)
    j <- j + 1
  fimenquanto
fimalgoritmo
```

C. Laço indeterminado: **repita...ate**

Este comando repete uma sequência de comandos até que uma determinada condição (especificada através de uma expressão lógica) seja satisfeita. Possui a seguinte sintaxe:

```
repita
 <seqüência de comandos>
ate <expressão lógica>
```

Onde:

- Inicialmente executa a <sequência de comandos> caso o valor da <expressão lógica> for VERDADEIRO executa novamente a <sequência de comandos> caso contrário encerra o laço.

Consideraremos ainda o mesmo exemplo:

```
algoritmo "Números de 1 a 10 (com repita)"
var
  j: inteiro
inicio
  j <- 1
  repita
    escreva (j:3)
    j <- j + 1
  ate j > 10
fimalgoritmo
```

A melhor maneira de aprender essa série de comandos é realizar exercícios, então não amoleça.

Exercício 7

7.1. Faça um programa que determine o maior entre N números. O programa deve verificar o maior valor até que a entrada seja igual a 0 (ZERO).

7.2. Uma rainha requisitou os serviços de um monge e disse-lhe que pagaria qualquer preço. O monge, necessitando de alimentos, indagou à rainha sobre o pagamento, se poderia ser feito com grãos de trigo dispostos em um tabuleiro de xadrez, de tal forma que o primeiro quadro deveria conter apenas um grão e os quadros subsequentes, o dobro do quadro anterior. A rainha achou o trabalho barato e pediu que o serviço fosse executado, sem se dar conta de que seria impossível efetuar o pagamento. Faça um programa para calcular o número de grãos que o monge esperava receber.

7.3. Faça um programa que conte de 1 a 100 e a cada múltiplo de 10 emita uma mensagem: "Múltiplo de 10".

Simbologia da Lógica

Esta é a simbologia comum a todos os programas que tratam sobre lógica.

Operadores Aritméticos

São os utilizados para obter resultados numéricos. Os símbolos para esses operadores são:

Simbolo	Operação
+	Adição
–	Subtração
*	Multiplicação
/	Divisão

Prioridade dos operadores aritméticos:

1º) É resolvido o que estiver entre ( ) Parênteses.

2º) É resolvido entre a multiplicação ou a divisão (o que aparecer primeiro na ordem da esquerda para direita).

3º) É resolvido entre a adição ou a subtração (o que aparecer primeiro na ordem da esquerda para direita).

Por exemplo: a seguinte operação aritmética: 3 + 4 * 0 + 2, resulta no valor **5**.

Algumas linguagens apresentam o operador **%** que significa o resto de uma divisão. Por exemplo: 10 % 3 = 1; 10 % 4 = 2

Operadores Relacionais

Os operadores relacionais são utilizados para comparar valores numéricos. Estes operadores sempre retornam valores lógicos (Verdadeiro ou Falso)

Simbolo	Operação
=	Igual a
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a

Por exemplo: Para duas variáveis A = 5 e B = 3

Os resultados das expressões seriam:

A = B: Falso A > B: Verdadeiro A < B: Falso
A >= B: Verdadeiro A <= B: Falso

Algumas linguagens apresenta ainda o operador **NÃO**, basta apenas inverter o resultado da operação. Por exemplo: NÃO (A = B), o resultado é Verdadeiro.

Concatenadores Lógicos

Servem para combinar resultados de expressões, retornando se o resultado final é Verdadeiro ou Falso.

Simbolo	Operação
E	A expressão é verdadeira se todas as condições forem verdadeiras.
OU	A expressão é verdadeira se pelo menos uma das condições for verdadeira.

Por exemplo: Para três variáveis A = 5, B = 8 e C =1

Os resultados das expressões seriam:

(A = B) E (B > C): Falso (A > B) OU (B < C): Falso

(A < B) E (B > C): Verdadeiro (A < B) OU (B > C): Verdadeiro

(A <= B) E (B > C): Verdadeiro (A >= B) OU (B >= C): Verdadeiro

Exercício 8

8.1. Considere as variáveis **SALARIO** conforme os valores abaixo, **IR** que representa o valor de 10% do salário e **SALLIQ** que é obtido da subtração de SALARIO por IR. Informe se as expressões são verdadeiras ou falsas.

SALARIO	Expressão
100,00	(SALLIQ >= 100,00)
200,00	(SALLIQ <= 190,00)
300,00	(SALLIQ < 280,00)

8.2. Sabendo que **A** = 3, **B** = A + 4 e **C** = B * 0,2. informe se as expressões abaixo são verdadeiras ou falsas.

1. (A+C) > B
2. B >= (A + 2)
3. C = (B –A)
4. (B + A) <= C
5. (C+A) > B

8.3. Sabendo que **A** = 5, **B** = A - 1 e **C** = B - A e **D** = B / A, informe se as expressões abaixo são verdadeiras ou falsas.

1. (A > C) AND (C <= D)
2. (A+B) > 10 OR (A+B) = (C+D)
3. (A>=C) AND (D >= C)