



Lógica de Programação Aplicada à Linguagem (online)



Editora
IMPACTA

Introdução

Utilizamos a lógica no dia a dia mesmo sem perceber, mas transferir esse conhecimento à programação demanda muito estudo e paciência.

Os capítulos foram organizados de uma forma que o aluno perceba a importância de valorizar e exercitar o raciocínio e não somente decorar sintaxe de uma linguagem específica.

Utilizaremos diversas formas de construir e testar um algoritmo. Primeiro iremos conceituá-lo, depois iremos construí-lo visualmente com o Scratch, em seguida aprenderemos a usar uma forma de visualização do algoritmo com o VisualG e finalmente utilizaremos a linguagem de programação Python para trabalharmos com as estruturas aprendidas.

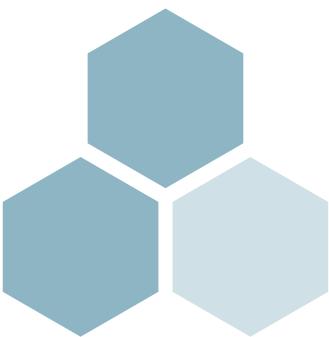
É recomendado que você organize os seus estudos, crie pastas por capítulos, pois alguns exercícios alteram exercícios anteriores. Refaça os exercícios de maneiras diferentes. Exercite o seu raciocínio!



1

Lógica de Programação

- ◆ O que é lógica?
- ◆ O que é programar?
- ◆ O que é um algoritmo?
- ◆ Estrutura e fase de um algoritmo.



1.1. O que é lógica?

Quando a palavra **lógica** é empregada no dia a dia, relacionamos ao raciocínio mental, mas esse raciocínio deverá ser aquele que corrige o pensamento. Em outras palavras, a lógica estuda e ensina a organizar e estruturar os pensamentos. Há inúmeros exemplos no nosso cotidiano, como: ao chupar uma bala, primeiro é necessário tirá-la da embalagem; se precisamos de algo que está numa gaveta trancada, deve-se abri-la primeiro.

A lógica é utilizada em diversas áreas, tais como a física, a informática, entre outras. E, na computação, ela está presente em tudo, sendo utilizada no hardware (parte física) e no software (parte lógica/programas).

1.2. O que é programar?

No universo da computação, programar é "ensinar" o computador a fazer ou resolver algum problema. O computador basicamente realiza cálculos e armazena dados, sendo capaz de entender instruções e executá-las. Uma forma de "ensinar" o computador é por meio dos algoritmos, e é por meio do raciocínio lógico que os construímos que depois se tornarão software.

O principal objetivo de estudar a Lógica de Programação é formular algoritmos coerentes e válidos.

1.3. O que é um algoritmo?

Utilizamos algoritmos sem perceber. Por exemplo, ao fazer aquele bolo maravilhoso de cenoura, se você não seguir uma receita respeitando os produtos e suas quantidades e o modo de fazer, o bolo não ficará bom. Observe com atenção o modo de preparo da receita a seguir:

- **Bolo de cenoura com cobertura**

Ingredientes:

3 cenouras médias e picadas
3 ovos
1 xícara de óleo
2 xícaras de açúcar
2 xícaras de farinha de trigo
1 colher (sopa) de fermento em pó
1 pitada de sal

Modo de preparo:

Em um liquidificador, adicione as cenouras descascadas e picadas, o óleo e os ovos. Depois, adicione a farinha de trigo, o açúcar e o fermento em pó, e leve para assar em forno médio (180° C) preaquecido por aproximadamente 35 a 40 minutos.

O algoritmo é uma sequência de passos em que a ordem dos fatores é importante. Como no caso da receita do bolo que foi apresentada, em que há uma sequência de passos no modo de preparo que deverá ser seguida. Importante: Esses passos deverão sempre ser apresentados de forma clara e precisa com o objetivo de trazer o resultado almejado após a execução dessa "norma". Esse entendimento é muito importante, então apresentaremos mais um exemplo de algoritmo ainda utilizando o português nosso de cada dia:

- **Pneu perfurado**

- Encostar o carro;
- Averiguar qual dos pneus está com problema;
- Apanhar o pneu de estepe, o macaco e a chave de roda;
- Apanhar e sinalizar com o triângulo a via movimentada;
- Situar o macaco para erguer o carro;
- Desprender os parafusos;
- Trocar o pneu;
- Reposicionar os parafusos;
- Abaixar o carro;
- Guardar o pneu furado, o macaco e a chave de roda;
- Retirar o triângulo da via;
- Guardar o triângulo;
- Ir embora.

Ao observar essa sequência de ações, você poderá questionar: "Mas e se algum dia eu tiver a sorte de o pneu estourar ao lado de um borracheiro ou parar num estacionamento para trocá-lo? Mesmo assim terei que colocar o triângulo?" Fazendo esse ajuste de realidade, o nosso algoritmo ficaria da seguinte forma:

- **Pneu perfurado**

- Encostar o carro;
- Averiguar qual dos pneus está com problema;
- Apanhar o pneu de estepe, o macaco e a chave de roda;
- **SE o carro estiver parado numa via movimentada:**
- Pegar e sinalizar com o triângulo;
- Situar o macaco para erguer o carro;
- Desprender os parafusos;
- Trocar o pneu;
- Reposicionar os parafusos;
- Abaixar o carro;
- Guardar o pneu furado, o macaco, a chave de roda.
- **SE colocou o triângulo na via:**
- Retirar o triângulo da via;
- Guardar o triângulo;
- Ir embora.

Os blocos de SE que foram incluídos no nosso algoritmo somente serão seguidos se a condição for afirmativa. Não se preocupe se você não entendeu muito bem o uso do SE, pois será explicado depois com mais detalhes. Agora, o importante é você perceber que, no algoritmo, precisaremos utilizar algumas estruturas ao longo dos passos para atender algumas exigências da realidade, como foi o caso do nosso exemplo da troca de pneu que precisou usar uma estrutura SE para avaliar se colocaria ou não o triângulo. As principais estruturas serão estudadas ao longo deste curso.

1.3.1. Formas de apresentação

Considerando que o algoritmo é uma linha de raciocínio, podemos descrevê-lo de diversas formas, como descrição narrativa, pseudocódigo e fluxograma.

1.3.1.1. Descrição narrativa

A descrição narrativa é o formato que estávamos usando até agora, mas não é muito utilizada na computação, pois pode gerar más interpretações e ambiguidades.

1.3.1.2. Pseudocódigo ou linguagem algorítmica

O pseudocódigo é um tipo de algoritmo intermediário entre a linguagem natural (idioma) e a linguagem de programação. Significa "falso código". Serve para organizar o raciocínio lógico para a resolução de um problema.

Para uma melhor compreensão, utilizaremos o seguinte exemplo:

```
DECLARA nota1,nota2,media:NUMERO
ESCREVA "Digite a nota1:"
LEIA nota1
LEIA nota2
media = (nota1 + nota2)/2
ESCREVA "Sua média foi:"
ESCREVA media
SE media >= 7 ENTÃO
    ESCREVA "APROVADO"
SENÃO
    ESCREVA "REPROVADO"
```

Desenvolva em pseudocódigo para mostrar a situação do aluno considerando que somente será **Aprovado** se a média das duas notas for maior ou igual a 7.

1.3.1.3. Fluxograma

O fluxograma é um tipo de algoritmo que utiliza símbolos gráficos para representar os passos a serem seguidos. A simbologia utilizada é padrão. Seguem os mais utilizados:

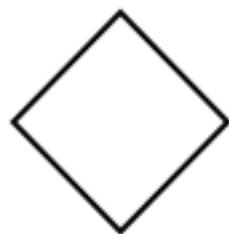
- Representa o início ou o fim do fluxograma:



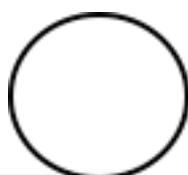
- Atividade, processamento ou ações:



- Verificação ou decisão. Escolha uma das sequências de instruções:



- Conector. Serve para interligar partes do fluxograma:



- Linha de fluxo. Seta de orientação. Pode ser horizontal ou vertical:



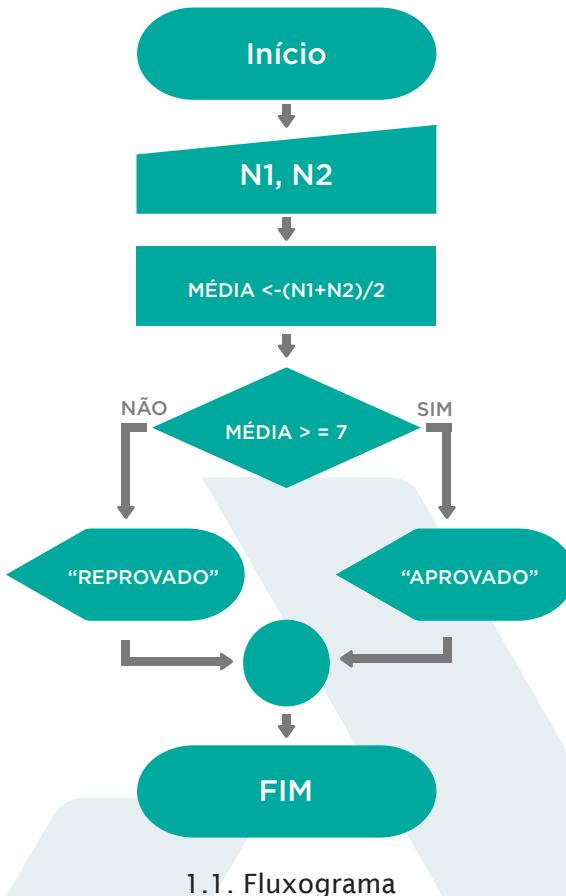
- Representa a entrada de dados manual:



- Representa a saída de informações ou exibir na tela:



- Utilizando o exemplo anterior do cálculo da **Média do Aluno**, o fluxograma ficará da seguinte forma:



1.1. Fluxograma

1.4. Estrutura e fase de um algoritmo

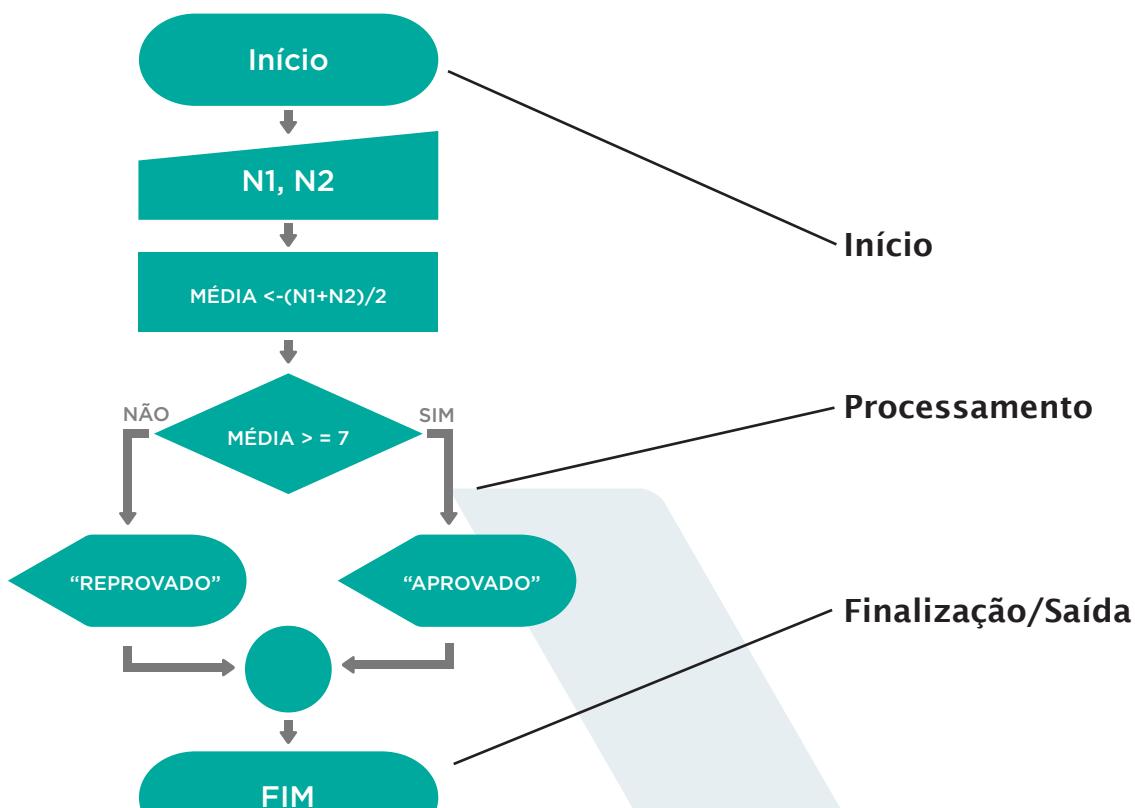
Se o seu objetivo é aprender uma linguagem de programação que são formas padronizadas de se comunicar/ensinar um computador, será necessário entender primeiramente como organizar as ações que serão realizadas pelo computador, ou seja, precisa-se aprender a organizar os pensamentos de maneira lógica e organizada, tornando-se um algoritmo. E, como o algoritmo são instruções que resolvem um problema específico, a sua estrutura deve possuir um começo e um fim.

Os algoritmos possuem a seguinte estrutura:

- Início;
- Processamento;
- Finalização.

Atenção: Todo o algoritmo deve atender essa divisão.

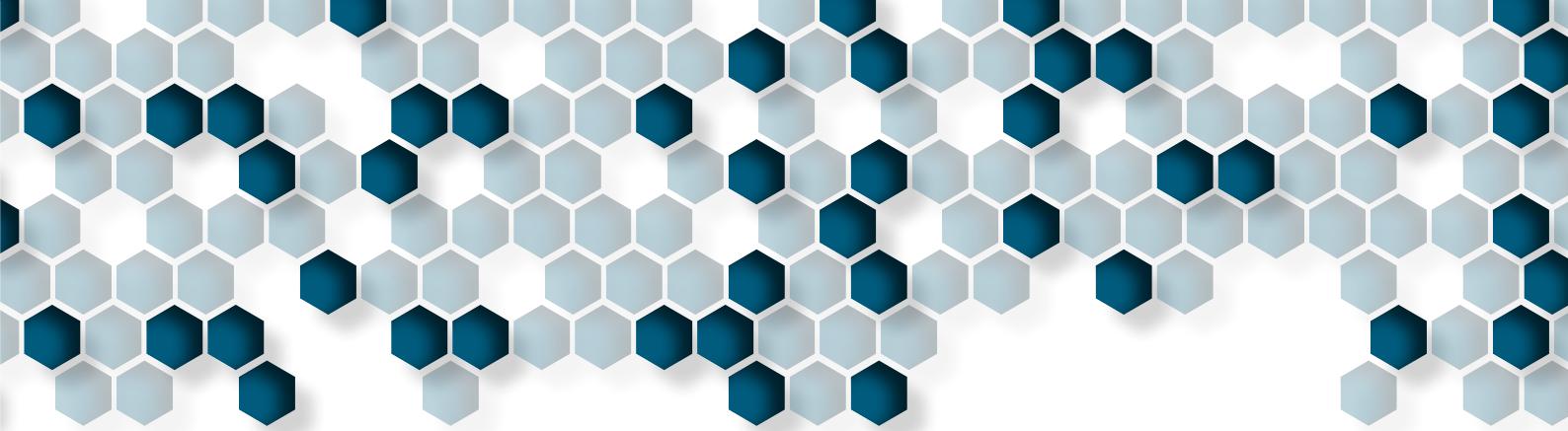
Se analisarmos o fluxograma apresentado anteriormente, é possível identificar cada parte.



1.2. Fluxograma e partes

Em resumo, o algoritmo transforma (processa) as entradas, então as fases dos algoritmos são:

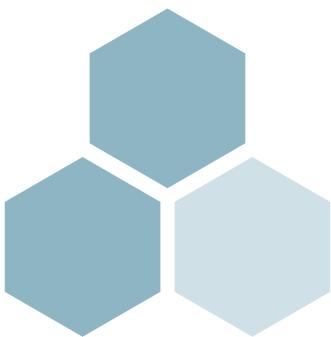
Entrada -> Processamento -> Saída.



2

Variáveis, constantes e tipos de dados

- ◆ O que são variáveis?
- ◆ Tipos de dados;
- ◆ Diferença entre variáveis e constantes;
- ◆ Atribuição;
- ◆ Bloco de instruções;
- ◆ Entrada e saída de dados.



2.1. O que são variáveis?

As entradas no algoritmo, citado anteriormente, são consideradas **dados** que serão processados e se tornarão **informações**.

Para ocorrer esse processamento de acordo com as especificações do algoritmo, os dados são armazenados temporariamente em **variáveis** na memória do computador. E essas variáveis são classificadas de acordo com o tipo do valor a ser armazenado a fim de não gerar problemas no processamento. Então, para definirmos uma variável, é preciso definir um nome e tipo.

Por exemplo, se definirmos uma variável, a batizamos como **resultadosoma** e escolhermos que será do tipo **Inteiro**, então poderemos guardar somente números inteiros (sem casas decimais) nessa variável e não adiantará tentar guardar qualquer outro tipo de dados nela; não será possível.

2.2. Tipos de dados

Escolher o tipo de dado mais adequado para ser armazenado em uma variável é muito importante. Ao desenvolver um algoritmo, precisa-se analisar qual o tipo de dado que será utilizado para resolver o problema proposto.

Todavia, somente as linguagens de programação fortemente tipadas possuem esse conceito, como C#, Java, entre outros.

De maneira geral, há alguns tipos de variáveis:

- **String** (texto): Cadeia de caracteres, por exemplo, uma frase;
- **Inteiro**: Números inteiros (negativo, positivo);
- **Reais**: Números reais (negativo, positivo);
- **Booleano** (lógico): Falso ou verdadeiro.

2.3. Diferenças entre variáveis e constantes

Além das variáveis, existem as constantes. E qual será a diferença entre elas? Variáveis e constantes são repositórios de elementos, e a diferença entre elas é que o elemento armazenado numa constante é definido no início do programa e não é mais modificado, todavia, o elemento na variável pode ser alterado durante a execução do programa.

Um exemplo de constante é o valor de Pi. Uma vez passado o valor para o programa de 3,1416, não será modificado.

2.4. Atribuição

Quando definimos uma variável, para que possamos manipulá-la, é necessário "incluir" um valor dentro dela no início do algoritmo (declaração) ou ao longo da execução (no caso das constantes, o valor já é definido em sua declaração).

O ato de atribuir o valor é representado pelo símbolo \leftarrow , mas, na maioria das linguagens de programação, o símbolo utilizado é o igual ($=$).

Exemplos:

```
Altura ← 1.80  
Peso ← 65  
Ou  
Altura = 1,80  
Peso = 65
```

Nos dois casos, as variáveis **Altura** e **Peso** receberam os seguintes dados: 1.80 e 65, respectivamente.

2.5. Bloco de instruções

As instruções de uma linguagem de programação são executadas em sequência. A utilização de blocos de instruções apresenta onde inicia e termina o conjunto de instruções que serão executadas em sequência.

Então, o bloco de instrução geralmente se apresenta neste formato:

```
<declaração de variaveis>  
inicio  
<primeira instrução do bloco>  
...  
<ultima instrução do bloco>  
fim
```

2.6. Entrada e saída de dados

Os programas recebem entradas e apresentam saídas. Durante a execução do programa, o usuário poderá informar valores, que são as entradas, para que estes sejam processados e depois gerem uma saída ou resultado. As entradas poderão ser feitas por meio dos dispositivos de entrada, como teclado, mouse, microfone, entre outros, e a saída poderá ser feita pelo vídeo, impressora, entre outros.

Exemplo de código de entrada e saída:

Entrada: leia (<variável>);
Saída: Escreva (<valor>);

Exemplo: Apresentar a média aritmética de três notas.

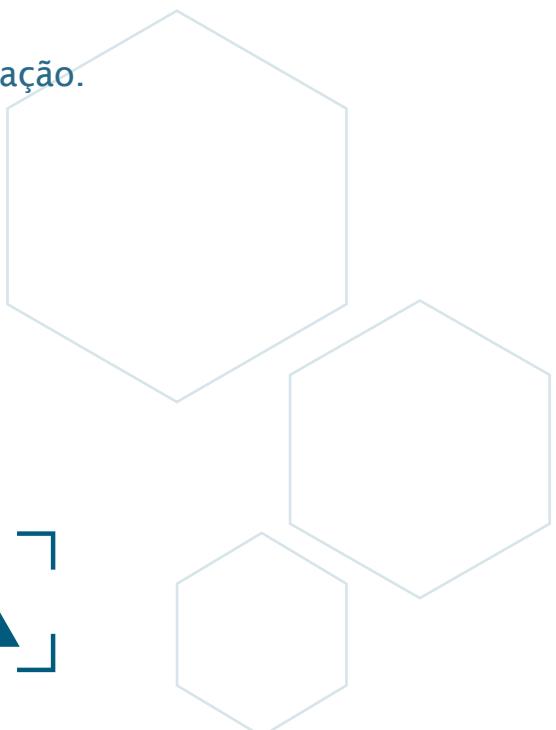
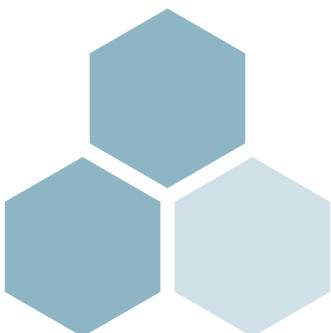
```
inicio
//tipo da variável e a declaração de variáveis que serão utilizadas no
programa//
real: N1, N2, N3, MA;
//em seguida o programa irá ler (entrada) do usuário//
leia (N1,N2,N3);
//depois, o programa "atribuirá" na variável MA o cálculo da média
utilizando as variáveis citadas anteriormente//
MA ← (N1+N2+N3)/3;
//finalmente o programa apresenta a saída dos dados (resultado)//
escreva (MA);
fim
```

O símbolo // neste exemplo significa um comentário, ou seja, a frase entre esse símbolo não será processada ou lida pelo programa.

3

Scratch

- ◆ Conhecendo o Scratch;
- ◆ Primeira animação;
- ◆ Manipulando variáveis;
- ◆ Operadores aritméticos, relacionais e lógicos;
- ◆ Estruturas de decisão;
- ◆ Estruturas de repetição ou iteração.



3.1. Conhecendo o Scratch

Scratch é uma linguagem de programação desenvolvida pelo Lifelong Kindergarten Group, no MIT Media Lab, com o apoio financeiro de diversas empresas.

Com ele, é permitido criar diversas animações interativas, jogos, entre outros, facilmente, pois a programação é por meio de blocos de comandos muito simples que se encaixam a fim de produzirem as ações esperadas. Em resumo, os projetos dessa linguagem são baseados em objetos gráficos. É possível mudar a sua aparência (diversos personagens), dar instruções, como mover-se, tocar música, entre outras interações.

A importância de utilizarmos essa ferramenta nesse curso está na facilidade de aprendizagem da lógica da programação. Vocês verão! Mas não adianta somente ler o material. Sem dúvida é necessário manusear a ferramenta precisa, interagir.

Atenção: Esse material não esgota todos os recursos dessa linguagem; somente apresentaremos o necessário para você compreender as estruturas mais importantes da Lógica de Programação. É possível utilizar essa linguagem on-line ou instalar em seu computador. A versão que instalaremos será a 2.0.

Para instalar, entre no site: <<https://scratch.mit.edu/download>>.

A imagem 3.1 apresenta a tela de download do site. Caso não tenha o Adobe Air, também, deverá instalá-lo. As instalações de ambos são simples.

The screenshot shows the Scratch download page. At the top, there's a navigation bar with links for 'Criar', 'Explorar', 'Dicas', 'Acerca', 'Pesquisa', 'Aderir ao Scratch', and 'Entrar'. Below the navigation bar, a note for Mac users is displayed: 'Nota para utilizadores Mac: a última versão do Scratch 2.0 Desconectado requer o Adobe AIR 20. Para actualizar manualmente para o Adobe AIR 20, clique [aqui](#)'. The main content area is divided into three sections: 1. Adobe AIR (with a note about Mac users needing Adobe AIR 20), 2. Editor Desconectado do Scratch (with a note about installing the Scratch 2.0 editor), and 3. Materiais de Apoio (with links to project templates, getting started guides, and scratch cards). Each section has a large blue numbered circle above it.

1

Adobe AIR

Se ainda não a tem, descarregue e instale a última versão do [Adobe Air](#)

Mac OS X - [Descarregar](#)
Mac OS 10.5 e Mais Antigos - [Descarregar](#)
Windows - [Descarregar](#)

2

Editor Desconectado do Scratch

De seguida, descarregue e instale o Editor Desconectado do Scratch 2.0

Mac OS X - [Descarregar](#)
Mac OS 10.5 e Mais Antigos - [Descarregar](#)

3

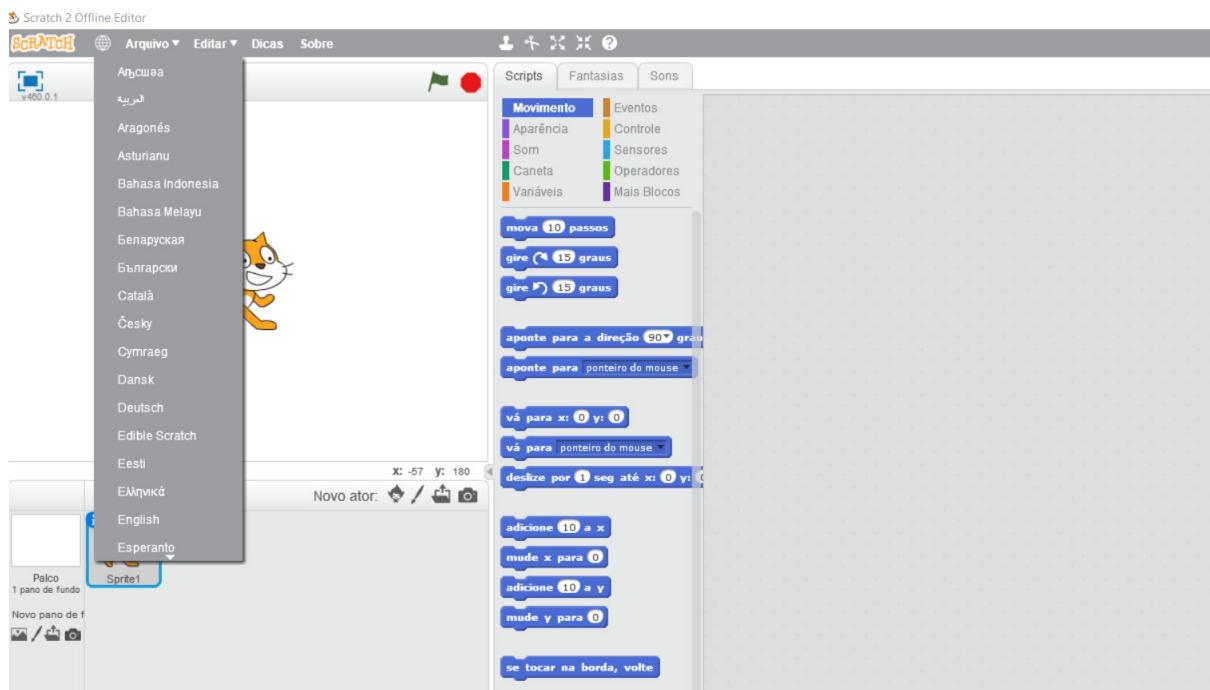
Materiais de Apoio

Precisa de ajudar para começar? Aqui estão alguns recursos úteis.

Projectos de Arranque - [Descarregar](#)
Guia de Iniciação - [Descarregar](#)
Cartas Scratch - [Descarregar](#)

3.1. Download Scratch

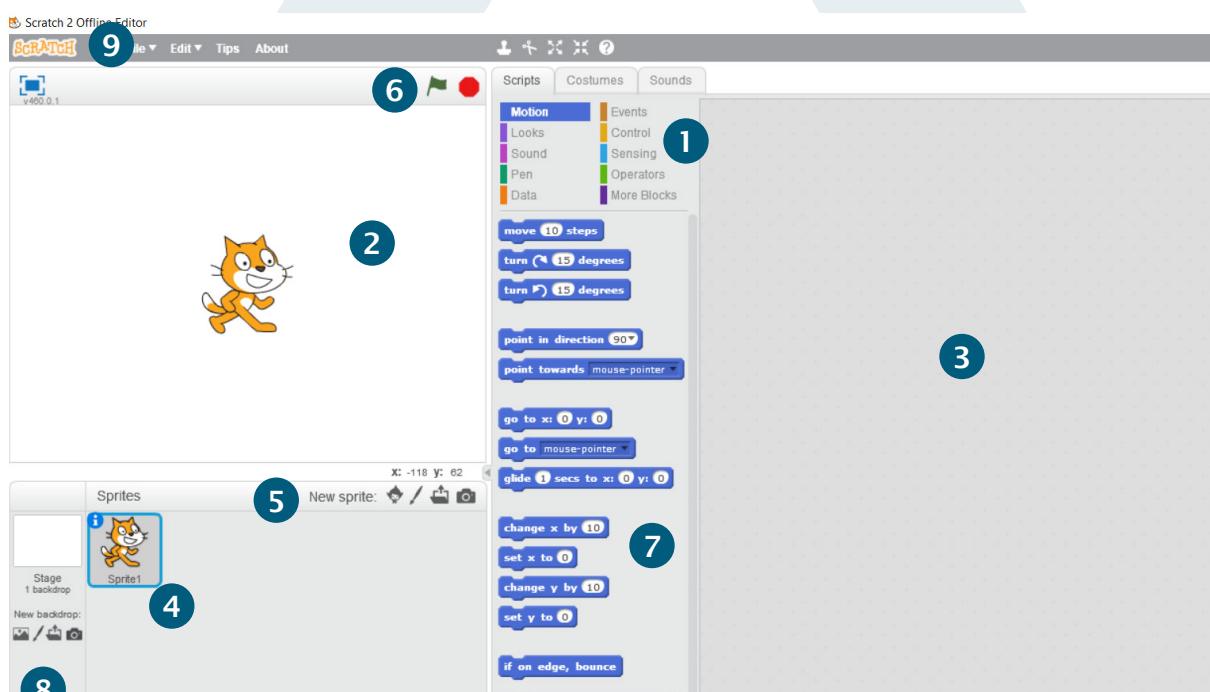
Após a instalação, caso queira, é possível alterar o idioma (clicar no ícone globo) conforme mostra a imagem 3.2.



3.2. Alterar idioma

3.1.1. Tela inicial

A tela inicial do Scratch é um ambiente para montar o algoritmo de programação por meio de diagramas de blocos.



3.3. Tela inicial do Scratch

A tela inicial do Scratch (imagem 3.3) é composta por: **Categoria de comandos** (1), **Palco onde ocorrerão as animações** (2), **Área de edição** (3), **Objetos de animação** (4), **Menu para novos objetos de animação** (5), **Botões de Iniciar e Parar a animação** (6), **Blocos de comando** (7), **Escolher cenários** (8) e **Menu** (9).

! Em todos os exemplos deste capítulo, utilizaremos essa sequência numérica para localizar os itens na tela inicial do Scratch.

3.1.1.1.Categoria de comandos

Na **Categoria de comandos** (1), há três abas: **Scripts**, **Fantasias** e **Sons**.

Na aba **Scripts**, há vários itens importantíssimos que usaremos muito ao longo do nosso curso:

- **Movimento**: Define o movimento do Ator;
- **Aparência**: Manipula as fantasias e inclui diálogos;
- **Som**: Adiciona sons;
- **Caneta**: Traços originados pelo Ator ao se movimentar;
- **Variáveis**: Cria variáveis para armazenamento de valores;
- **Eventos**: Adiciona algumas situações;
- **Controle**: Estruturas lógicas;
- **Sensores**: Situações que podem ser combinadas com outros comandos;
- **Operadores**: Operações matemáticas;
- **Mais Blocos**: Pode criar um bloco personalizado.

E cada item contém os seus **Blocos de comandos** (7) agrupados.

Na aba **Fantasia**, encontramos as "fantasias" do objeto gráfico escolhido (Ator). Essas fantasias são quadros estáticos de movimentação do personagem escolhido e, por meio do algoritmo, será possível movimentar o Ator utilizando esses quadros. É possível escolher esses quadros (costumes) da biblioteca do Scratch, pintar ou desenhar, carregar a partir de um arquivo do computador ou utilizar uma Web cam para tirar uma foto e depois utilizá-la como fantasia.

A última aba, **Sons**, possibilita escolher alguma música que poderá ser usada (tocada) pelo Ator.

3.1.1.2. Menu

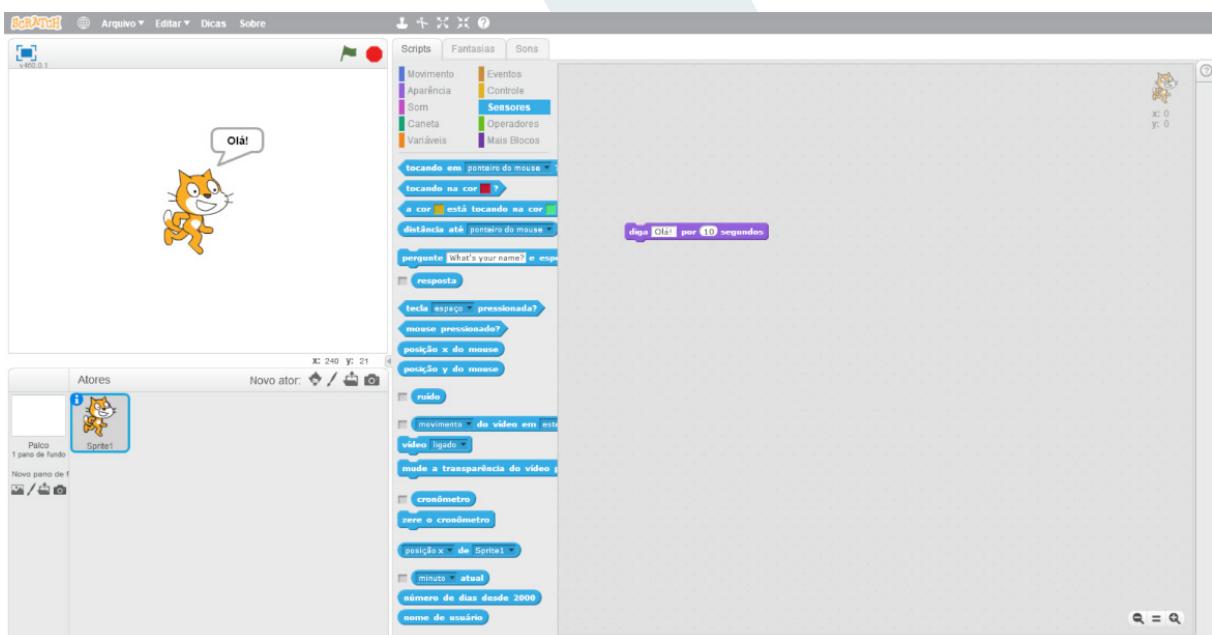
No menu (9), encontramos **Arquivo**, **Editar**, **Dicas** e **Sobre**.

O item que mais utilizaremos será o **Arquivo – Novo, Abrir, Salvar e Salvar Como**.

Curiosidade: É possível compartilhar um projeto, ou seja, colocá-lo on-line.

3.2. Primeira animação

Como primeiro projeto, o gato, objeto gráfico padrão do Scratch, dirá um **Olá!**, como na imagem 3.4.



3.4. Primeiro projeto

Para isso, vá na **Categoria de comandos** (1), escolha **Aparência** e arraste o bloquinho "diga Hello! por 2 segundos" para a **Área de edição** (3). Depois, troque **Hello!** por **Olá!** e **2 segundos** por **10 segundos**; para o bloquinho ser executado, aplique um duplo-clique sobre ele. Pronto!

3.3. Manipulando variáveis

No próximo exemplo, manipularemos uma variável para armazenar uma informação e, depois, emitir uma mensagem para o usuário. O gato perguntará o nome e, depois, cumprimentará utilizando o nome recebido. Diferente do primeiro exemplo, a inicialização dos blocos deverá ser pelo botão **Iniciar** (6).

O bloco de algoritmo ficará conforme a imagem 3.5 na **Área de Edição** do Scratch.



3.5. Manipulando variáveis

Nesse exemplo, utilizaremos somente a aba **Script** da **Categoria de comandos** (1).

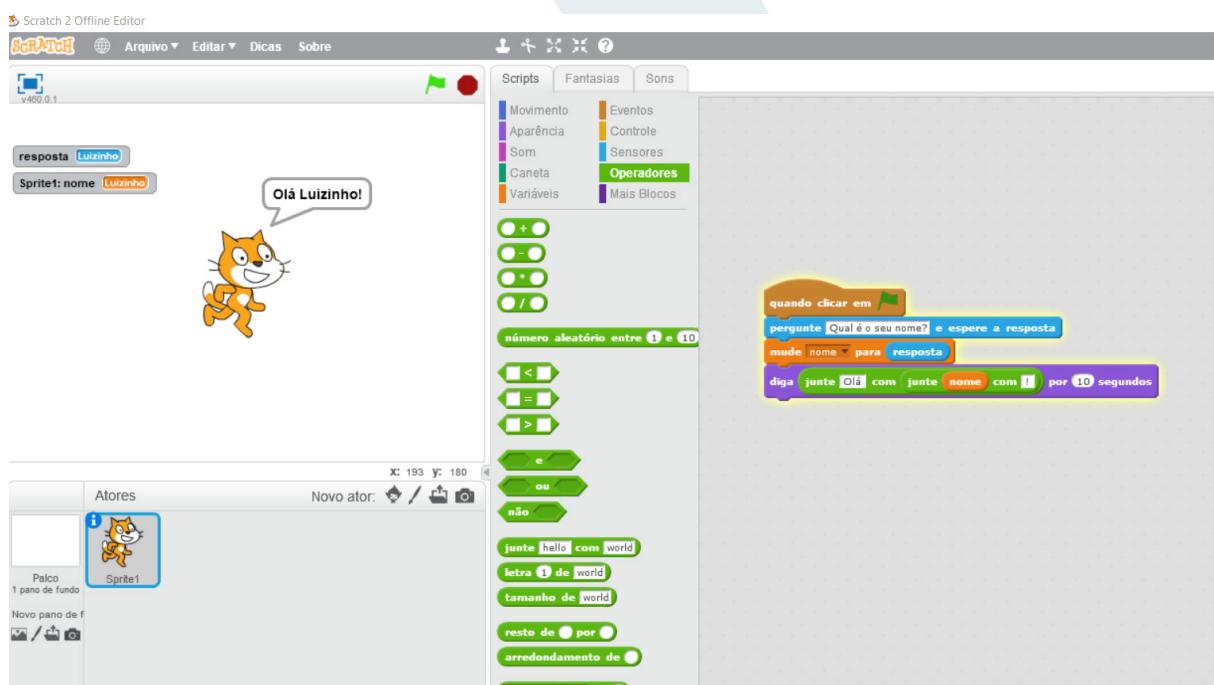
Adiante, a sequência dos blocos que deverão ser arrastados e montados na área de edição, sendo que a variável primeiramente é criada e, depois, utilizada:

Aba	Item	Bloco	Qtd.	Obs.
Script	Variáveis	Criar uma variável	1	Nomear como "nome".
Script	Eventos	Quando clicar em (bandeira verde)	1	
Script	Sensores	pergunte ... e espere a resposta	1	
Script	Variáveis	mude (nome) para ...	1	
Script	Sensores	resposta	1	Encaixe no bloco anterior.
Script	Aparência	Diga ... por ... segundos	1	
Script	Operadores	junte ... com ...	2	Encaixe no bloco anterior.
Script	Variáveis	nome	1	Arraste a variável criada e encaixe no bloco anterior.

Atenção:

- Os três pontos (...) significam que é um campo de preenchimento ou também serve para encaixar os blocos;
- Os blocos se encaixam como peças de quebra-cabeça;
- Os blocos são coloridos de acordo com os itens da Categoria de comando, por exemplo, o bloco diga por 10 segundos é roxo, assim como todos os blocos do item Aparência.

Depois de montar o algoritmo, clique no botão **Iniciar** (6) e o gatinho perguntará o seu nome e responderá como na imagem 3.6.



3.6. Emitindo mensagem

3.4. Operadores aritméticos, relacionais e lógicos

Utilizamos os operadores para comparar, calcular e associar expressões. Os operadores mais utilizados são:

- **Operadores aritméticos:** São utilizados para a realização dos diversos cálculos matemáticos. Seguem alguns deles:
 - **Incremento:** Adiciona 1 ao valor anterior;
 - **Decremento:** Subtrai 1 ao valor anterior;
 - **Multiplicação (*):** Exemplo: $a*b$;
 - **Divisão (/):** Exemplo: $1/b$;
 - **Adição (+):** Exemplo: $a+b$;
 - **Subtração (-):** Exemplo: $a-b$;
 - **Módulo:** Retorna o resto da divisão inteira de A por B ($A\%B$). Exemplo: $9\%2 = 1$.
- **Operadores relacionais:** Estabelece uma relação de comparação entre valores ou expressões, sendo que o resultado é sempre um valor lógico (booleano) verdadeiro (true) ou falso (false).
 - **Maior:** $>$;
 - **Maior ou igual:** \geq ;
 - **Menor:** $<$;
 - **Menor ou igual:** \leq ;
 - **Igual a:** $=$;
 - **Diferente:** \neq .

 Alguns símbolos se diferem entre as linguagens de programação.

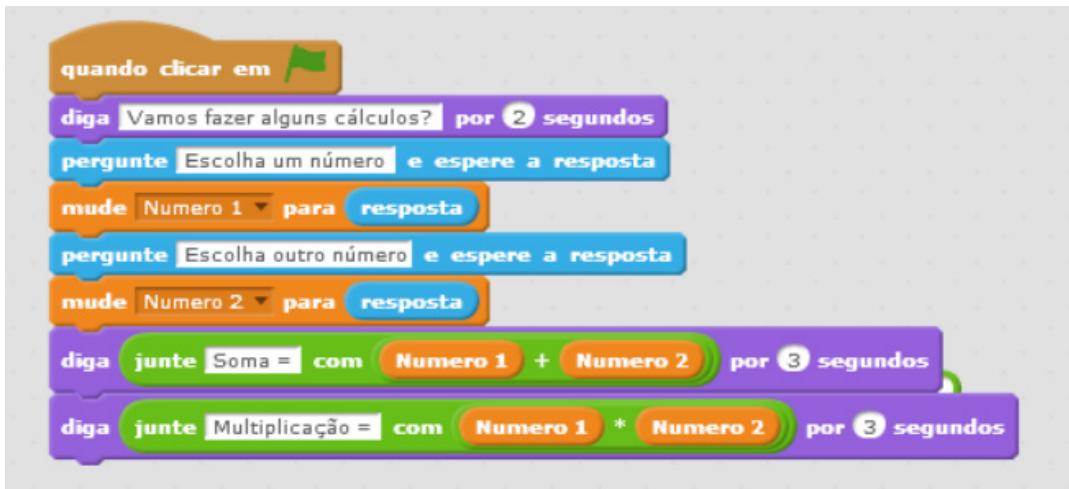
- **Operadores lógicos:** Associa expressões comparando valores e o resultado dessas expressões também retorna verdadeiro ou falso. Os operadores são **e**, **ou** e **não**. A relação entre eles pode ser visualizada na **Tabela Verdade**:

P	Q	P e Q	P ou Q	Não P
True	True	True	True	False
True	False	False	True	False
False	False	False	False	True

Exemplo: A sentença no operador E somente é verdadeira se os dois itens que a compõe são verdadeiros.

3.4.1. Operadores aritméticos no Scratch

Agora, um novo personagem (Ator) do Scratch irá fazer dois cálculos a partir de dois números que o usuário digitou. O algoritmo ficará conforme a imagem 3.7.



3.7. Algoritmo – Operadores aritméticos

Antes de montar os blocos na Área de edição, deve-se alterar o Ator e o cenário (palco) para atingir o mesmo resultado da imagem 3.8.

O Ator utilizado foi o **Giga**, e o Palco, o **Winter**, e, conforme já mencionado anteriormente (3.1.2 Tela inicial), estão respectivamente nas posições 5 e 8.

Segue a lista dos blocos que foram utilizados nesta animação:

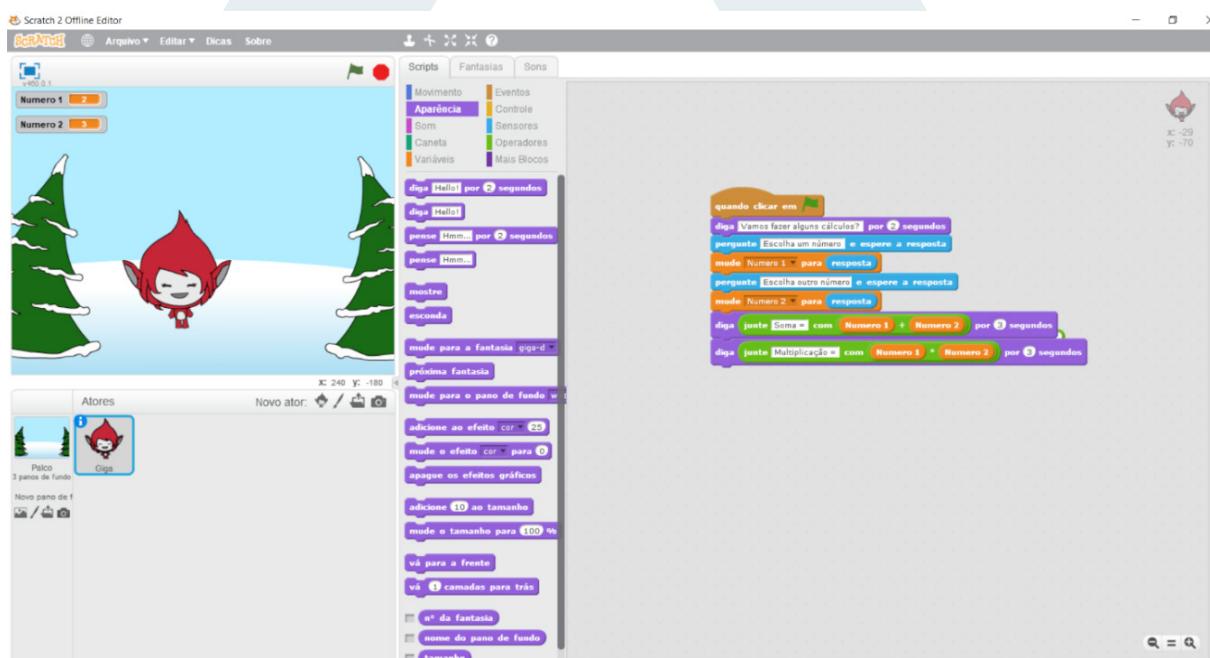
Lembrando:

As variáveis devem ser criadas para serem utilizadas.

Lógica de Programação Aplicada à Linguagem (online)

Aba	Item	Bloco	Qtd.	Obs.
Script	Variáveis	Criar uma variável ()	2	Criar duas variáveis: numero1; numero2
Script	Eventos	quando clicar em (bandeira verde)	1	
Script	Aparência	diga ... por ...	3	
Script	Sensores	pergunte ... e espere a resposta	2	
Script	Variáveis	Mude () para ...	2	
Script	Sensores	resposta	2	Encaixe no bloco anterior
Script	Operadores	junte ... com ...	2	Encaixe no bloco diga ... por ...
Script	Operadores	...+...	1	Encaixe no bloco Junte ... com ...
Script	Operadores	...*...	1	Encaixe no bloco Junte ... com ...
Script	Variáveis	Número 1	2	Encaixe nos operadores
Script	Variáveis	Número 2	2	Encaixe nos operadores

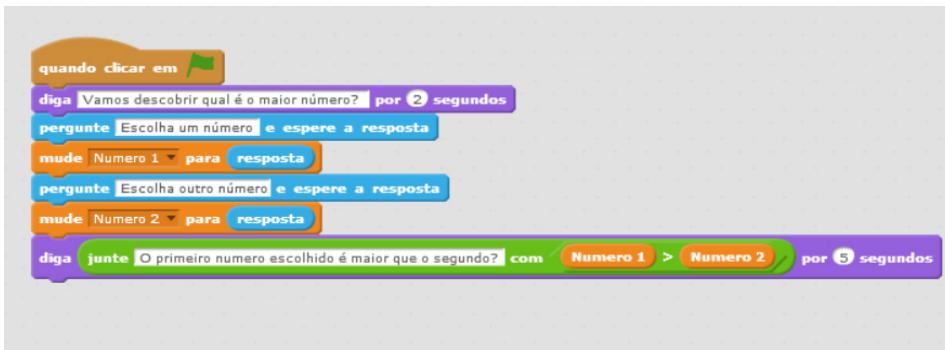
Após todas as montagens e alterações de ator e cenário, a tela do Scratch ficará conforme a imagem 3.8.



3.8. Tela do Scratch – Operadores aritméticos

3.4.2. Operadores relacionais no Scratch

Para demostrar como usar os operadores relacionais no Scratch, faremos a seguinte animação: o mesmo Ator do exemplo anterior verificará se o primeiro número informado pelo usuário é maior do que o segundo número também informado pelo usuário. Os blocos serão montados conforme a imagem 3.9.



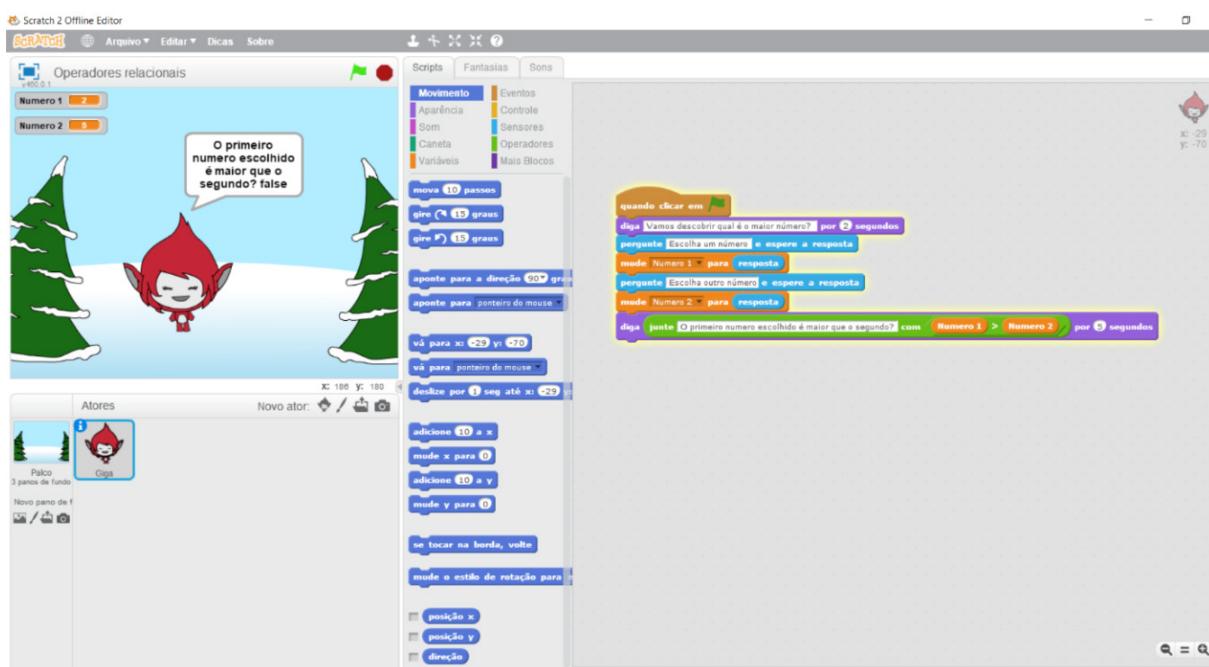
3.9. Algoritmo – Operadores relacionais

Como já comentado anteriormente, o Ator e o Cenário permanecem o mesmo do exemplo anterior.

Segue a lista de blocos utilizados neste exemplo:

Aba	Item	Bloco	Qtd.	Obs.
Script	Variáveis	Criar uma variável ()	2	Criar duas variáveis: numero1; numero2
Script	Eventos	quando clicar em (bandeira verde)	1	
Script	Aparência	diga ... por ...	2	
Script	Sensores	pergunte ... e espere a resposta	2	
Script	Variáveis	Mude () para ...	2	
Script	Sensores	resposta	2	Encaixe o bloco anterior
Script	Operadores	junte ... com ...	2	Encaixe no bloco diga ... por ...
Script	Operadores	...>...	1	Encaixe no bloco Junte ... com ...
Script	Variáveis	Número 1	1	Encaixe nos operadores
Script	Variáveis	Número 2	1	Encaixe nos operadores

Após a montagem do algoritmo, a tela do Scratch ficará conforme a imagem 3.10.



3.10 Tela do Scratch – Operadores relacionais

Observe que, após o usuário informar os dois números, o nosso personagem informa se a condição (variável 1 > variável 2) é true (verdadeira) ou false (falsa).

3.4.3. Operadores lógicos no Scratch

Podemos utilizar os operadores lógicos (e, ou, não) no Scratch e, para isso, utilizaremos o mesmo ator dos exemplos anterior, o Giga que, após o usuário escolher dois números, fará algumas associações lógicas.

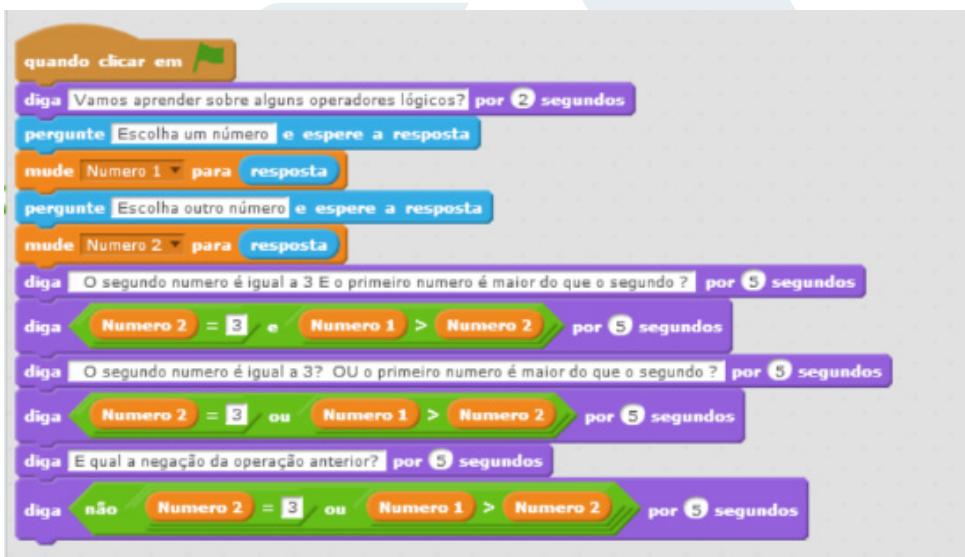
As duas sentenças que serão associadas nos operadores lógicos E e OU serão as seguintes:

- O segundo número informado pelo usuário é igual a 3?
- O primeiro número informado é maior do que o segundo?

Atenção: Como já vimos na Tabela Verdade, quando usamos o operador lógico E, somente será verdadeira se TODAS as sentenças que estão sendo analisadas forem atendidas. E, no caso do operador lógico OU, será verdadeira se ao menos UMA das sentenças envolvidas for verdadeira.

Para finalizar, o operador NÃO negará a sentença utilizada na operação anterior (OU).

O algoritmo ficará como a imagem 3.11.



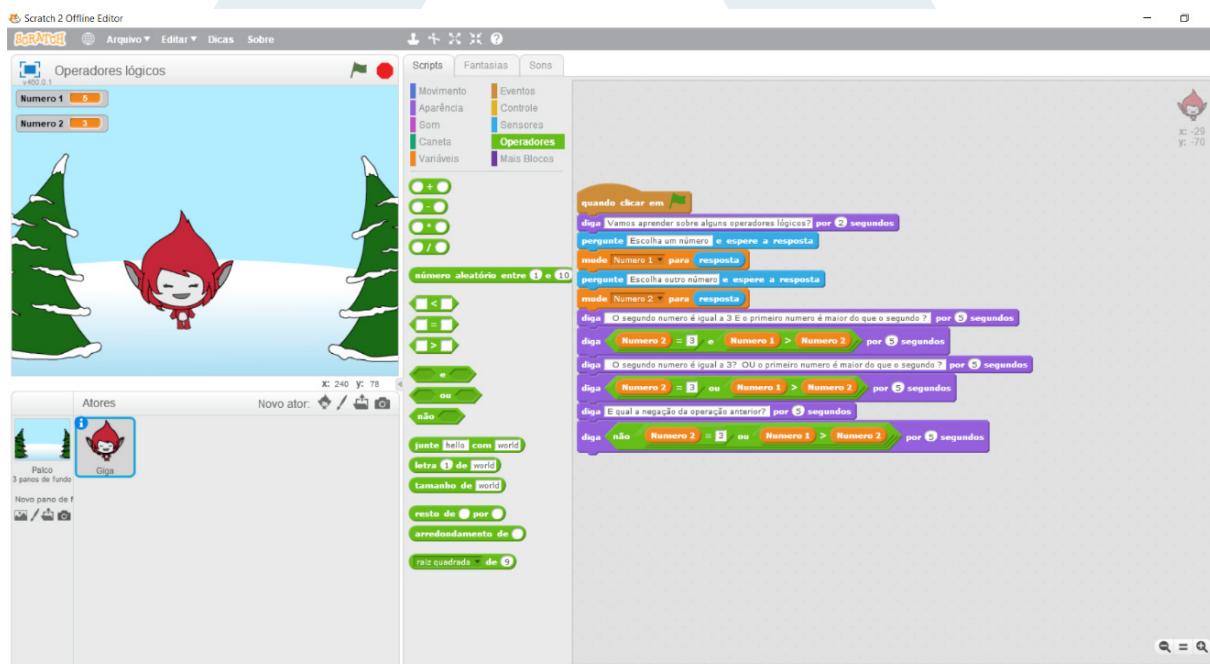
3.11. Algoritmo – Operadores lógicos

Segue a lista de blocos utilizados neste exemplo:

Continuaremos com o mesmo ator e cenário.

Aba	Item	Bloco	Qtd.	Obs.
Script	Variáveis	Criar uma variável ()	2	Criar duas variáveis: numero1; numero2
Script	Eventos	quando clicar em (bandeira verde)	1	
Script	Aparência	diga ... por ...	7	
Script	Sensores	pergunte ... e espere a resposta	2	
Script	Variáveis	Mude () para ...	2	
Script	Sensores	resposta	2	Encaixe no bloco anterior
Script	Operadores	... e ...	1	Encaixe no bloco diga ... por ...
Script	Operadores	... ou ...	1	Encaixe no bloco diga ... por ...
Script	Operadores	não...	1	Encaixe no bloco diga ... por ...
Script	Operadores	...=...	3	
Script	Operadores	...>...	3	
Script	Variáveis	Numero 1	3	Encaixe nos operadores
Script	Variáveis	Numero 2	6	Encaixe nos operadores

Após a montagem do algoritmo, a tela do Scratch ficará conforme a imagem 3.12.



3.12. Tela do Scratch – Operadores lógicos

3.5. Estruturas de decisão

Utilizamos a estrutura de decisão ou seleção para verificar opções de escolha. Essas escolhas são condições representadas por condições lógicas ou relacionais.

O seguinte modelo é utilizado na estrutura de Seleção Simples:

Se (condição) então (sequência de comandos)
Fim se;

Na Seleção Composta, é utilizado o modelo a seguir:

Se (condição) então (sequência de comandos)
Senão (sequência de comandos)
Fim se;

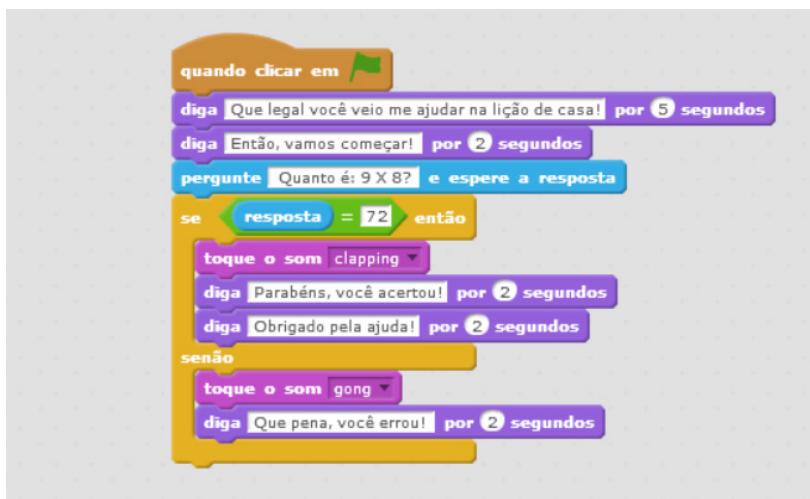
Na Estrutura Simples, se a condição for atendida, a instrução (sequência de comandos) será executada, ou seja, neste caso, se a condição NÃO for atendida, NÃO processará. Todavia, na Estrutura Composta, se a condição for atendida, a instrução do ENTÃO será executada, mas se NÃO for atendida, serão processadas as instruções de comando do SENÃO.

Existe, também, a estrutura de seleção de múltiplas escolhas, onde há um conjunto de opções para escolher. Pode-se utilizar encadeamento da instrução SE (vários SE em sequência) ou utilizar a instrução ESCOLHA CASO, disponíveis em algumas linguagens.

3.5.1. Estrutura de decisão no Scratch

Vamos visualizar como essa estrutura é utilizada no Scratch. Para isso, ajudaremos uma nova personagem a fazer a lição de casa respondendo apenas uma pergunta de multiplicação. Aproveitaremos para utilizar um recurso dessa linguagem, a inclusão de um som durante a execução do algoritmo.

O algoritmo ficará como a imagem 3.13.



3.13. Algoritmo – Estrutura de decisão

A personagem utilizada foi a Abby, e o Palco, bedroom 1, e, conforme já mencionado anteriormente (3.1.2 Tela inicial), estão respectivamente nas posições 5 e 8.

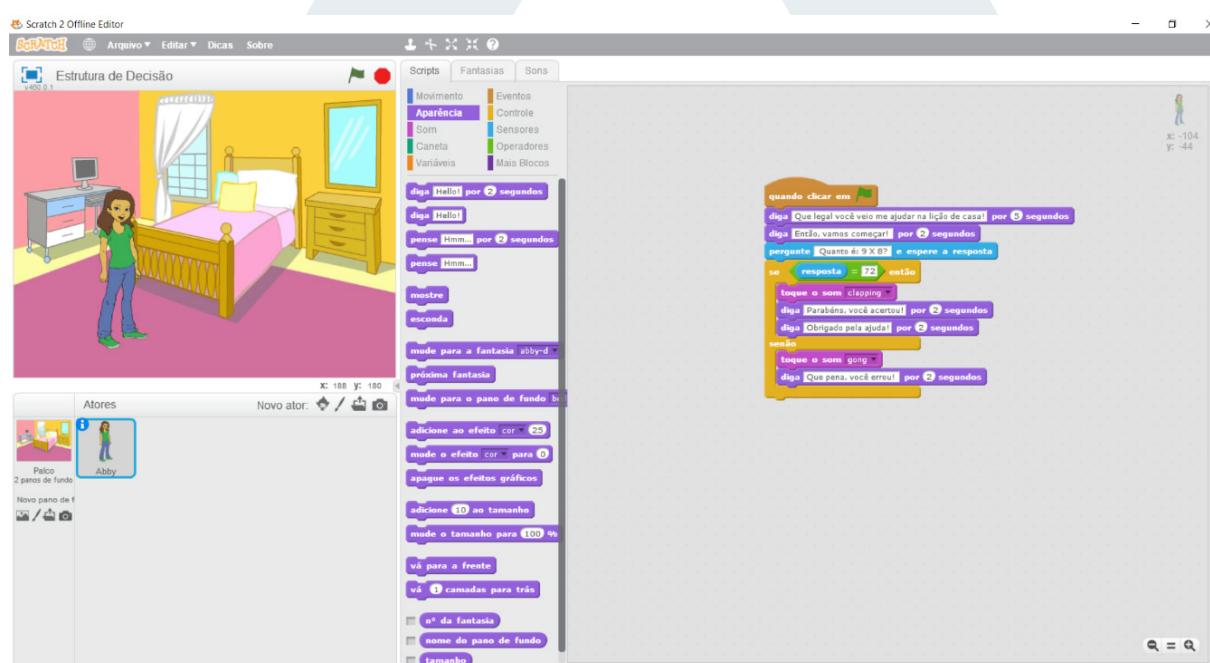
Antes de utilizar o Som no algoritmo, deve-se escolhê-lo na aba **Som**, conforme já mencionado anteriormente (3.1.2.2 Categoria de comandos). Neste exemplo, utilizaremos sons contidos na biblioteca do Scratch.

Segue a lista dos blocos que foram utilizados nesta animação:

Aba	Item	Bloco	Qtd.	Obs.
Script	Eventos	quando clicar em (bandeira verde)	1	
Script	Aparência	Diga ... por ...	5	
Script	Sensores	pergunte ... e espere a resposta	1	
Script	Controle	Se ... então ... senão ...	1	
Script	Operadores	... = ...	1	Encaixe no bloco anterior.
Script	Sensores	resposta	1	Encaixe no bloco se Então ... Senão
Som				Escolha novo som da biblioteca: clapping; gong
Script	Som	Toque O som...	2	Som diferente: no se o clapping e no senão o gong

Ao executar esse algoritmo, dependendo da resposta do usuário, um som diferente será executado.

Após a montagem do algoritmo, a tela do Scratch ficará conforme a imagem 3.14.



3.14. Tela do Scratch – Estrutura de Decisão

3.6. Estruturas de repetição ou iteração

A estrutura de repetição ou iteração também são conhecidas como laços ou "looping", onde sequências de comandos são executadas repetidamente conforme uma quantidade de vezes pré-determinadas no algoritmo ou até que uma condição lógica permita.

As estruturas mais utilizadas nas linguagens de programação são: enquanto (while), para (for) e faça enquanto (do while). As principais características dessas estruturas são:

- **Enquanto (while):** Repete uma sequência de comandos enquanto uma expressão lógica ou relacional é verdadeira;
- **Para (for):** Repete até uma quantidade de vezes e é muito utilizado um contador para contar quantas vezes aquela instrução será repetida. Quando o contador alcança a quantidade máxima de repetição estabelecida no algoritmo, a repetição é cessada;



Quando falamos de um contador, estamos nos referindo a uma variável que está sendo incrementada (variável que é igual a ela mesma mais 1, ou seja, $v=v+1$).

- **Faça enquanto (do while):** Antes de verificar se a condição lógica é verdadeira para "entrar" no "looping", o trecho é executado uma vez obrigatoriamente.

Não se preocupe se esses conceitos ainda não ficaram claros, pois, ao longo desse curso, veremos vários exemplos.

3.6.1. Estrutura de repetição no Scratch

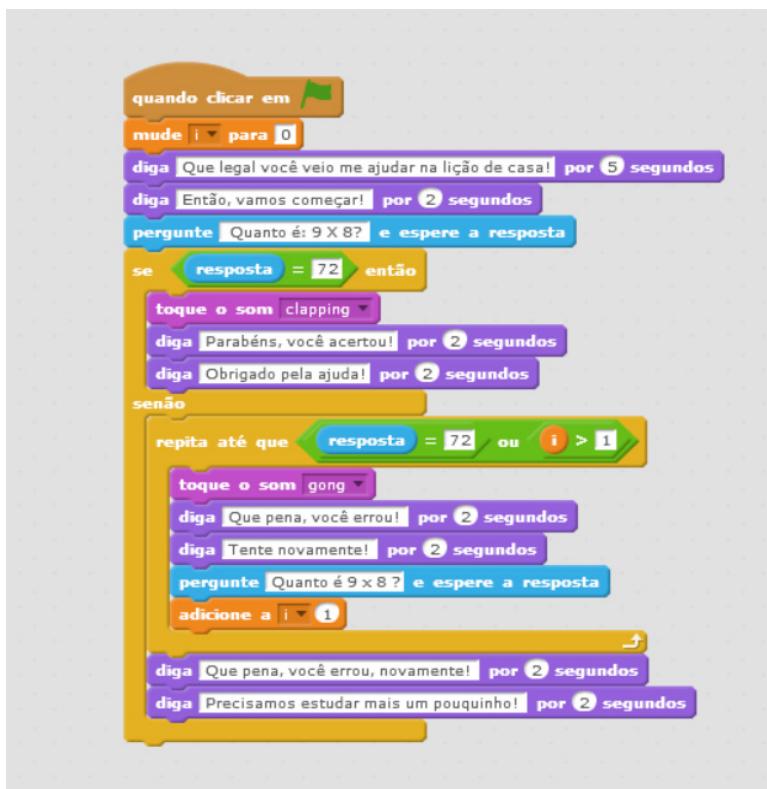
Veremos como a estrutura de repetição se comporta no Scratch. No próximo exemplo, utilizaremos o mesmo personagem e o mesmo cenário do exemplo anterior. Na verdade, utilizaremos quase toda a estrutura; somente acrescentaremos um único viés: se a resposta do usuário estiver errada, haverá mais algumas chances para responder à mesma questão, contudo, o número de chances será limitado.

Nesse caso, a estrutura de repetição estaria encaixada no SENÃO, pois somente haverá chances para responder se a primeira resposta foi errada. Em resumo: após a primeira resposta estar incorreta, o "processamento entrará" no SENÃO e encontrará a estrutura de repetição que repetirá até que a resposta esteja correta ou a quantidade de vezes de chances seja atingida.

Para passarmos essa limitação para o algoritmo, será necessário criar uma variável (no caso, foi o i) que será incrementada a cada processamento, e determinaremos quanto essa variável poderá acumular e, uma vez ultrapassado esse limite, o processamento "sairá" do bloco de instrução da estrutura de repetição.

Não podemos nos esquecer de inicializar a variável com o valor 0 por se tratar de uma estrutura de repetição.

O algoritmo ficará como a imagem 3.15.

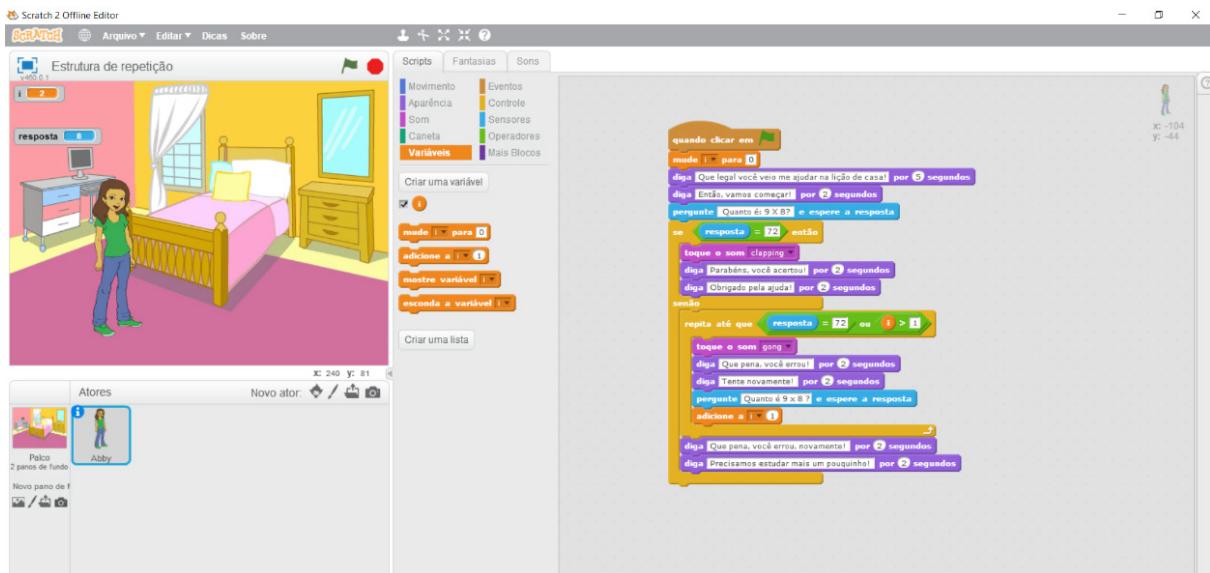


3.15. Algoritmo – Estrutura de repetição

Segue a lista dos blocos que foram utilizados nesta animação:

Aba	Item	Bloco	Qtd.	Obs.
Script	Variáveis	Criar uma variável ()	1	Nomear: i
Script	Eventos	quando clicar em (bandeira verde)	1	
Script	Variáveis	Mude () para ...	1	
Script	Aparência	Diga ... por ...	8	
Script	Sensores	pergunte ... e espere a resposta	2	
Script	Controle	se ... então ... senão ...	1	
Script	Operadores	... = ...	1	Encaixe no bloco anterior.
Script	Operadores	repita até que ...	1	Encaixe no senão ...
Script	Operadores	... ou ...	1	Encaixe no bloco repita até que ...
Script	Operadores	... = ...	1	Encaixe no bloco anterior.
Script	Operadores	... > ...	1	Encaixe no bloco ... ou ...
Script	Variáveis	i	1	Arraste a variável criada e encaixe no bloco anterior
Script	Sensores	resposta	2	Encaixe nos blocos ... = ...
Som				Escolha novo som da biblioteca: clapping; gong
Script	Som	Toque o som ...	2	Som diferente: no bloco se o clapping e no bloco repita o gong
Script	Variáveis	Adicione a ()	1	

Após a montagem do algoritmo, a tela do Scratch ficará conforme a imagem 3.16.



3.16. Tela do Scratch – Estrutura de repetição

Como já mencionado, o Scratch contém muitos outros recursos que podem ser explorados, então, agora que você viu como utilizar algumas estruturas, aproveite para manipulá-los de diferentes formas e realizar testes.

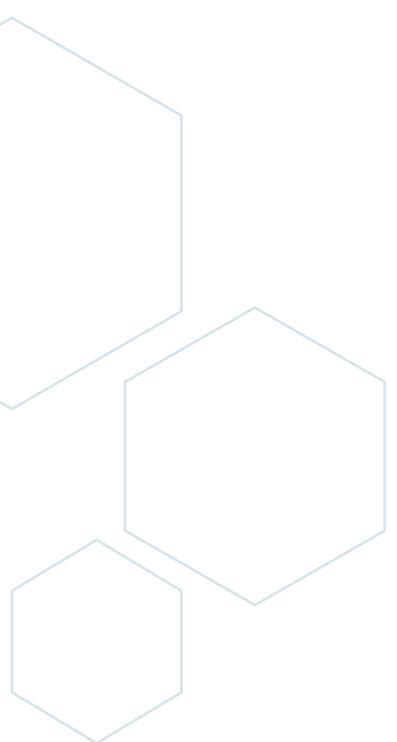
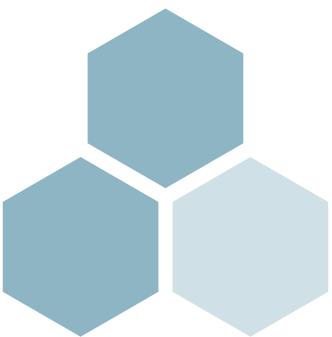
Não se esqueça: somente é possível aprender lógica de programação por meio de treinos e descobertas de novas formas de raciocinar logicamente.



4

VisualG

- ◆ Conhecendo o VisualG;
- ◆ Manipulação de variáveis no VisualG;
- ◆ Operadores aritméticos, relacionais e lógicos no VisualG;
- ◆ Estrutura de decisão no VisualG;
- ◆ Estrutura de repetição no VisualG.





4.1. Conhecendo o VisualG

Já vimos anteriormente o que é o pseudocódigo, e agora vamos aprender como testá-lo utilizando o software VisualG, que edita e interpreta algoritmos simulando uma "tela" do MS-DOS para mostrar o "resultado" dos trechos dos códigos. Ele é gratuito e de domínio público.

Para usá-lo, deve-se fazer o download no site: <<http://visualg3.com.br/baixar-o-visualg3-0/>>, como mostra a imagem 4.1, e a versão que iremos utilizar nesse curso é 3.0.

The screenshot shows a web browser displaying the URL visualg3.com.br/baixar-o-visualg3-0/. The page has a blue header bar with the VisualG logo and navigation links: Home, Baixar o VisualG, Tabelas Úteis, Sobre, Prof Antônio, Notícias, and Fale Conosco. Below the header, there's a search bar labeled 'BUSCA' with the placeholder 'Pesquisar ...' and a 'Pesquisar' button. A sidebar on the right lists categories under 'POR ASSUNTO' such as Curso, Dicas, Diversos, Materiais, and VisualG 3.0. Another sidebar on the right lists 'LINKS' including Baixar o VisualG 3.0, Blog, Contato, Home, and O Professor Antônio. The main content area features the title 'Baixe o VisualG 3.0.6.5' and a section titled 'BAIXE O VISUALG 3.0' with the instruction 'Descompactar o arquivo no c:' followed by an icon of a red folder with a computer monitor inside.

4.1. Download VisualG

Esse programa não é instalado. Após fazer o download, deve-se utilizar um outro programa para "descompactar" o conteúdo baixado e salvá-lo no computador.

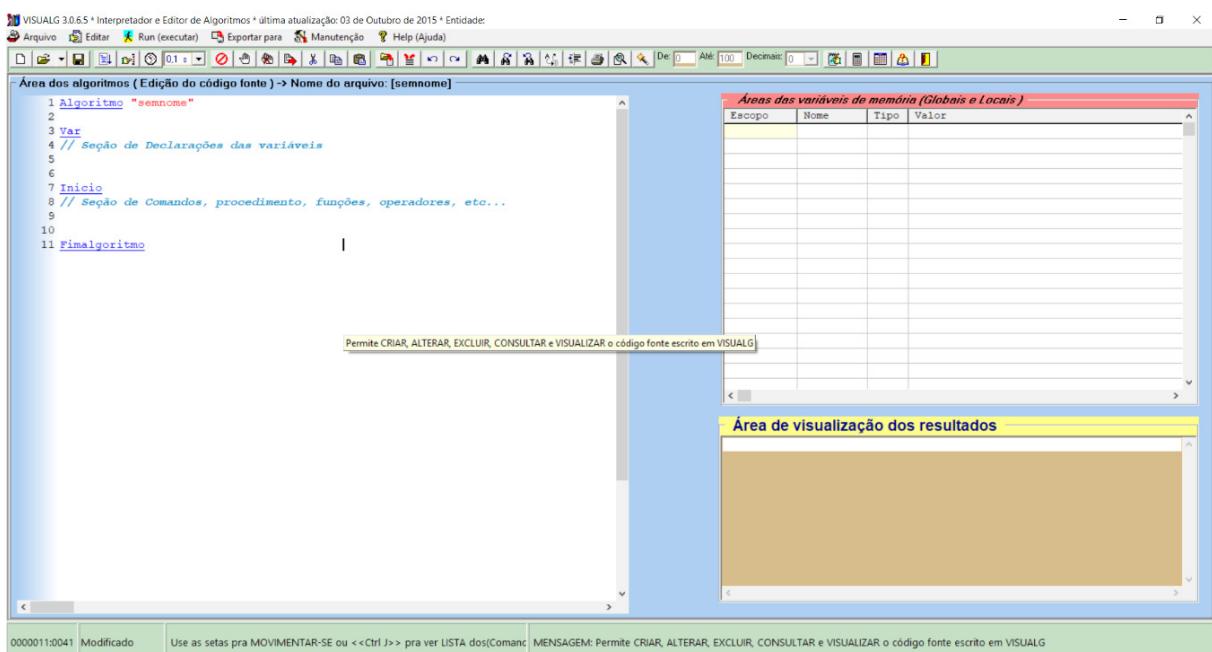
A imagem 4.2 mostra o conteúdo da pasta e o ícone para abrir o VisualG.

Nome	Data de modific...	Tipo	Tamanho
Exemplos	26/06/2018 23:22	Pasta de arquivos	
help	26/06/2018 23:22	Pasta de arquivos	
skins	26/06/2018 23:22	Pasta de arquivos	
dicas	20/06/2008 08:59	Parâmetros de con...	11 KB
help	13/07/2015 19:13	Arquivo de Ajuda ...	691 KB
LEIA-ME	12/09/2015 00:23	Documento de Te	1 KB
LEIA-ME	12/09/2015 00:23	Documento de Te	1 KB
listas	27/06/2018 18:49	Documento de Te	5 KB
Menu do Visualg aualizado	20/07/2015 02:47	Adobe Acrobat D...	1.236 KB
README	12/09/2015 00:23	Documento de Te	1 KB
RELAÇÃO DOS COMANDOS DO VISU...	04/10/2015 01:59	Documento de Te	7 KB
visualg3.tmp	27/06/2018 22:31	Arquivo TMP	1 KB
VISUALG30	13/07/2015 19:13	Arquivo de Ajuda ...	691 KB
visualg30	03/02/2017 11:22	Aplicativo	3.054 KB
visualg30	27/06/2018 21:47	Parâmetros de con...	2 KB

4.2. Conteúdo pasta VisualG

Depois de clicar no ícone, o programa é carregado e surge a Tela inicial como a imagem 4.3. Quando o programa é carregado a primeira vez, aparecem "telas" contendo dicas importantes.

Há uma pasta que contém inúmeros exemplos de algoritmos; aproveite-os para treinar. Como já dissemos algumas vezes, o melhor caminho para aprender lógica é fazendo.



4.3. Tela inicial VisualG

Mostraremos a utilização de vários recursos do VisualG, mas não esgotaremos todas as possibilidade que essa ferramenta disponibiliza ao estudante de programação.

Antes de começarmos a trabalhar, vamos observar a Tela inicial do VisualG, imagem 4.3.

Como esse programa é direcionado para os alunos de programação, a tela é autoexplicativa, pois traz vários comentários explicando o uso de cada parte, por exemplo, onde é a Área do algoritmo, ou seja, o local onde será montado o código; a Área de visualização dos resultados e a Área das variáveis que mostrará como será o comportamento das variáveis ao longo do processamento.

No topo da tela, há o menu que podemos encontrar em muitos programas, como o Arquivo - Abrir, Novo, Salvar; porém, o que mais utilizaremos será o Run (executar) que serve para executar ou "rodar" o algoritmo.

4.2. Manipulação de variáveis no VisualG

Antes de fazermos o primeiro algoritmo, é necessário relembrarmos alguns conceitos importantes.

O formato básico do pseudocódigo é o seguinte:

```
algoritmo "semnome"
// Seção de Declarações
inicio
// Seção de Comandos
fimalgoritmo
```

 As duas barras // significam um comentário, ou seja, aquela linha explica alguma coisa e não interfere no processamento.

Os tipos de dados que o VisualG processa (compreende) são:

- **Inteiro:** Variáveis numéricas do tipo inteiro, isto é, sem casas decimais;
- **Real:** Variáveis numéricas do tipo real com casas decimais;
- **Caractere:** Variáveis do tipo string (texto) e deve-se usar "" (aspas duplas);
- **Lógico:** Variáveis do tipo booleano (Verdadeiro / Falso).

 **Atenção:**

Os nomes das variáveis devem iniciar sempre utilizando letras, mas, depois, podem-se utilizar letras, números ou underline. O limite será de 30 caracteres. E, não pode haver duas variáveis com o mesmo nome para se referir a elementos diferentes. Ao utilizar uma variável, deverá declará-la na seção "declaração de variáveis" e, depois, indicar qual é o tipo de dado (inteiro, real, caractere ou lógico) que a variável processará.

Os comandos de entrada e saída são os seguintes:

Entrada:

Leia (<lista-de-variável>)

Saída:

Escreva (<lista-de-expressões>)

Ou, se quiser que o programa inicie o próximo comando na outra linha, use:

Escreval (<lista-de-expressões>)

Lembrando:

A entrada (`leia`) é equivalente ao bloquinho **mude (nome variável) para...** que utilizamos no Scratch, ou seja, a entrada quer dizer que a variável "irá receber" um dado que o usuário irá informar. E, a saída é o resultado do processamento, ou melhor, como e o que o algoritmo irá apresentar, mas o **Escreva** ou **Escreval**, também, pode ser utilizado para mostrar uma frase ao usuário ou fazer uma pergunta.

Agora, vamos para o nosso primeiro algoritmo.

Elabore um algoritmo que troca de posição dos seguintes números 2,5. Mas, no processamento, nenhum dos números poderá se "perder".

Uma possível resolução desse problema, tendo em mente não perder nenhum número no processamento, seria a criação de três variáveis, onde `x` e `y` (os nomes escolhidos) recebem os números dados e a variável `y` seria uma variável que serviria para cambiar os conteúdos. Para elucidar, seria como se eu tivesse uma maçã numa mão e uma laranja na outra e eu preciso que elas troquem de lugar, mas não consigo segurar as duas ao mesmo tempo, então eu aproveito o auxílio de uma mesinha que também "segura" uma de cada vez até eu conseguir alcançar o objetivo.

Falamos em possível resolução porque, ao longo do estudo, você irá perceber o real significado da palavra lógica que, por tratar-se de um raciocínio ordenado, é possível resolver o mesmo problema de diferentes maneiras. E a melhor solução dependerá de alguns quesitos, como a solução mais rápida ou mais adequada ou mais barata, entre outros.

```
ea dos algoritmos (Edição do código fonte) -> Nome do arquivo: [sem nome]
1 Algoritmo "Troca de posição"
2
3 Var
4 // Seção de Declarações das variáveis
5   x: inteiro
6   y: inteiro
7   z: inteiro
8
9 Inicio
10 // Seção de Comandos, procedimento, funções, operadores, etc...
11   x <- 2
12   y <- 5
13   z <- x
14   x <- y
15   y <- z
16   escreval (x,y)
17
18 Fimalgoritmo
Permite CRIAR, ALTERAR, EXCLUIR, CONSULTAR e VISUALIZA
```

4.4. Primeiro algoritmo

Após escrever o algoritmo e clicar em **Run**, o algoritmo é processado e será apresentada uma tela parecida com o MS-DOS com o resultado (saída), depois aperte a tecla ESC no teclado e a tela do programa ficará como a imagem 4.5.

! A setinha (<-) quer dizer que a variável recebe tal conteúdo.

Lembrando:

O conteúdo da variável pode ser alterado ao longo "dos passos" do processamento, e o processamento ocorre linha a linha de cima para baixo, como a leitura de um texto.

The screenshot shows the VisualG 3.0.65 interface. The code editor on the left contains the following pseudocode:

```

1 Algoritmo "Troca de posição"
2
3 Var
4 // Seção de Declarações das variáveis
5   x: inteiro
6   y: inteiro
7   z: inteiro
8
9 Início
10 // Seção de Comandos, procedimento, funções, operadores, etc...
11   x <- 2
12   y <- 5
13   z <- x
14   x <- y
15   y <- z
16   escreval (x,y)
17
18 FimAlgoritmo

```

The memory viewer on the right shows the state of variables:

Escopo	Nome	Tipo	Valor
GLOBAL	X	int	5
GLOBAL	Y	int	2
GLOBAL	Z	int	2

The results viewer at the bottom shows the execution log:

```

Início da execução
5 2
Fim da execução.

```

4.5. Tela do VisualG – Primeiro algoritmo

4.3. Operadores aritméticos, relacionais e lógicos no VisualG

No VisualG, é possível trabalhar com os operadores aritméticos, relacionais e lógicos. A seguir, veremos como podemos utilizá-los e posteriormente resolveremos alguns exercícios:

Operadores aritméticos: Soma (+), Subtração (-), Divisão (/), Multiplicação (*), Resto da Divisão (MOD ou %), Potenciação (^);

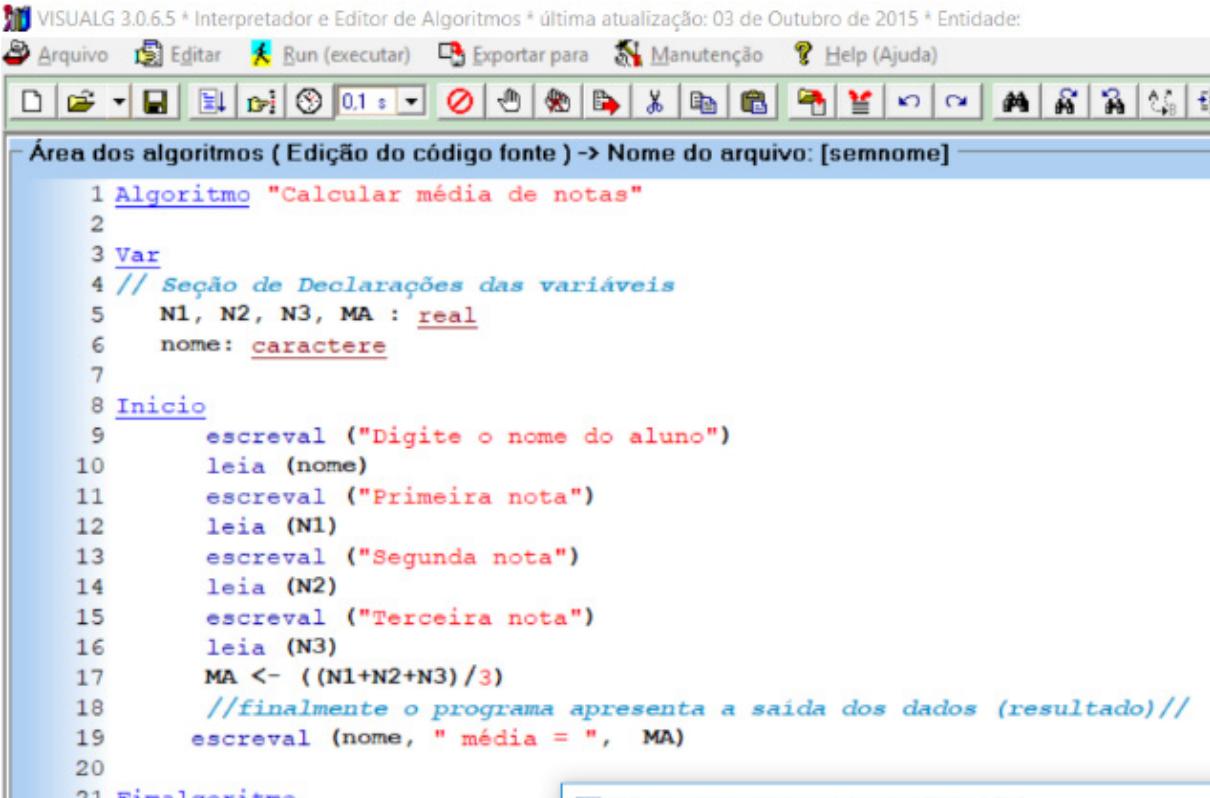
Operadores relacionais: Igual (=), Maior (>), Maior ou Igual (>=), Menor (<), Menor ou Igual (<=), Diferente de (<>). Sendo que o resultado será apresentado como **Verdadeiro** ou **Falso** (Booleano);

Operadores lógicos: E, OU, NÃO. O resultado será booleano. No VisualG, também existe o XOU que resulta Verdadeiro, quando os dois itens que estão sendo comparados forem diferentes, e Falso, se forem iguais.

No próximo exercício, usaremos alguns operadores numéricos:

Elabore um algoritmo que calcule a média das três notas de um aluno, sendo que o nome do aluno e as notas serão informados pelo usuário.

Podemos resolver esse exercício com o algoritmo mostrado na imagem 4.6.



VISUALG 3.0.6.5 * Interpretador e Editor de Algoritmos * última atualização: 03 de Outubro de 2015 * Entidade:
Arquivo Editar Run (executar) Exportar para Manutenção Help (Ajuda)

Área dos algoritmos (Edição do código fonte) → Nome do arquivo: [semnome]

```
1 Algoritmo "Calcular média de notas"
2
3 Var
4 // Seção de Declarações das variáveis
5   N1, N2, N3, MA : real
6   nome: caractere
7
8 Inicio
9   escreval ("Digite o nome do aluno")
10  leia (nome)
11  escreval ("Primeira nota")
12  leia (N1)
13  escreval ("Segunda nota")
14  leia (N2)
15  escreval ("Terceira nota")
16  leia (N3)
17  MA <- ((N1+N2+N3)/3)
18  //finalmente o programa apresenta a saída dos dados (resultado)//
19  escreval (nome, " média = ", MA)
20
21 Fimalgoritmo
```

4.6. Algoritmo – Operadores numéricos

Observe que usamos o "escreval" para solicitar ao usuário o que precisamos e, quando ele digita, a resposta é armazenada na variável. Quando temos todas as respostas de que precisamos, inserimos numa variável chamada MA o cálculo da média e depois é mostrado ao usuário ("escreval") o conteúdo da variável nome, a palavra média e o símbolo = (texto ou string deve-se colocar entre aspas) e o conteúdo da variável MA. Não se esqueça da vírgula para separar os componentes dentro dos parênteses.

Após o processamento, a tela ficará como a imagem 4.7.

The screenshot shows the VisualG 3.0.6.5 interface. The code editor window displays an algorithm named "Calcular média de notas". The code declares three variables (N1, N2, N3) and one character variable (nome). It prompts for the student's name and three marks, calculates the average (MA), and prints the results. The memory viewer shows variable values: N1=2, N2=8, N3=9, MA=6.33333333333333, and nome='Zezinho'. The console window shows the program's output, including the input values and the calculated average. The status bar at the bottom indicates the file is modified.

```

1 Algoritmo "Calcular média de notas"
2
3 Var
4 // Série de Declarações das variáveis
5 N1, N2, N3, MA : real
6 nome: caractere
7
8 Início
9   escreval ("Digite o nome do aluno")
10  leia (nome)
11  escreval ("Primeira nota")
12  leia (N1)
13  escreval ("Segunda nota")
14  leia (N2)
15  escreval ("Terceira nota")
16  leia (N3)
17  MA <- ((N1+N2+N3) / 3)
18  //Finalmente o programa apresenta a saída dos dados (resultado)//
19  escreval (nome, " média = ", MA)
20
21 Fimalgoritmo
22
23
24
25
26
27
28

```

4.7. Tela do VisualG – Operadores numéricos

Usaremos o mesmo exercício anterior e acrescentaremos somente uma linha no código e uma variável, o ME (média escolar), onde será verificado se a média calculada é maior do que 7 (que foi atribuída à variável ME). A linha acrescida ao algoritmo foi: escreval ("A média calculada é maior do que 7?", MA > ME).

Você deve ter observado que, na linha 21 (escreval), foi possível realizar "diretamente" uma comparação utilizando um operador relacional sem a necessidade de armazenar o resultado numa outra variável.

Essas observações são importantes para descobrir formas diferentes de resolver o mesmo problema ou exercício.

O algoritmo ficará como a imagem 4.8.

```

1 Algoritmo "Calcular média de notas"
2
3 Var
4 // Seção de Declarações das variáveis
5 N1, N2, N3, MA, ME : real
6 nome: caractere
7
8 Inicio
9     ME <- 7
10    escreval ("Digite o nome do aluno")
11    leia (nome)
12    escreval ("Primeira nota")
13    leia (N1)
14    escreval ("Segunda nota")
15    leia (N2)
16    escreval ("Terceira nota")
17    leia (N3)
18    MA <- ((N1+N2+N3)/3)
19    //finalmente o programa apresenta a saída dos dados (resultado)//
20    escreval (nome, " média = ", MA)
21    escreval ("A média calculada é maior do que 7? ", MA > ME)
22
23
24 Fimalgoritmo

```

Console simulando o modo texto do MS-DOS

4.8. Algoritmo – Operadores relacionais

A imagem 4.9 apresenta como ficará o resultado do algoritmo após a inserção de alguns dados quaisquer.

Escopo	Nome	Tipo	Valor
GLOBAL	N1	R	6,00000000000000
GLOBAL	N2	R	5,00000000000000
GLOBAL	N3	R	9,00000000000000
GLOBAL	MA	R	6,66666666666667
GLOBAL	ME	R	7,00000000000000
GLOBAL	NOME	C	"Luizinho"

Área dos resultados (Edição do código fonte) - Nome do arquivo: [semnome]

Área das variáveis da memória (Globais e Locais)

Área de visualização dos resultados

Console simulando o modo texto do MS-DOS

```

Digite o nome do aluno
Luizinho
Primeira nota
6
Segunda nota
5
Terceira nota
9
Luizinho média = 6.66666666666667
A média calculada é maior do que 7? FALSO
>>> Fim da execução do programa !

```

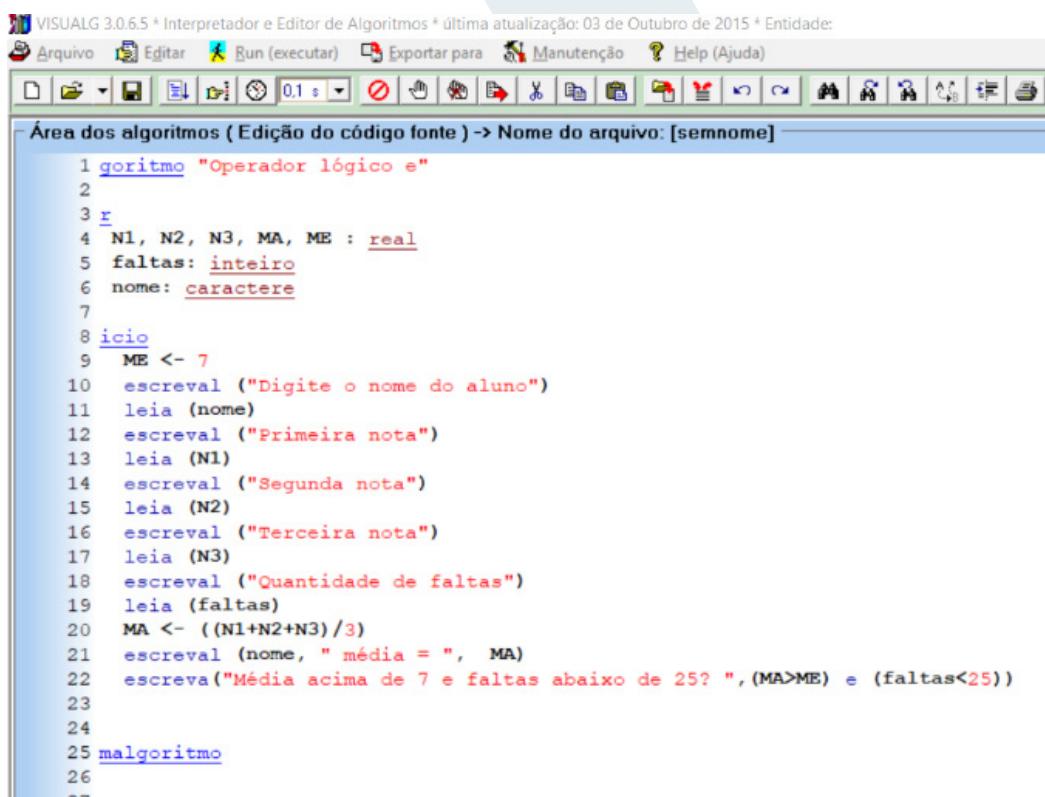
4.9. Tela do VisualG – Operadores relacionais

Para utilizarmos o operador lógico E, acrescentaremos mais uma variável (faltas) e será apresentada ao usuário mais uma comparação, onde será verificado se a média calculada do aluno está acima de 7 e se as faltas estão abaixo de 25.

Observe que, nesse caso, não criamos uma variável que "recebeu" ou foi atribuído o 25, como foi no caso anterior referente à variável ME, ou seja, é uma outra forma de fazer a mesma coisa. Claro, se for analisar o quesito de utilização de memória, é melhor verificar se realmente é necessária a criação de uma variável.

O algoritmo que utilizará o operador lógico e a tela do programa ficarão respectivamente como as imagens 4.10 e 4.11.

! O comportamento (resultado) dos operadores lógicos é conforme a Tabela Verdade já apresentada neste material.



```

1 goritmo "Operador lógico e"
2
3 E
4 N1, N2, N3, MA, ME : real
5 faltas: inteiro
6 nome: caractere
7
8 icio
9   ME <- 7
10  escreval ("Digite o nome do aluno")
11  leia (nome)
12  escreval ("Primeira nota")
13  leia (N1)
14  escreval ("Segunda nota")
15  leia (N2)
16  escreval ("Terceira nota")
17  leia (N3)
18  escreval ("Quantidade de faltas")
19  leia (faltas)
20  MA <- ((N1+N2+N3)/3)
21  escreval (nome, " média = ", MA)
22  escreva("Média acima de 7 e faltas abaixo de 25? ", (MA>ME) E (faltas<25))
23
24
25 algoritmo
26
27

```

4.10. Algoritmo – Operadores lógicos

The screenshot shows the VisualG 3.0.6.5 IDE interface. On the left, the code editor displays a program named "Operador lógico.e" with the following pseudocode:

```

1 goritmo "Operador lógico.e"
2
3 e
4 N1, N2, N3, MA, ME : real
5 faltas: inteiro
6 nome: caractere
7
8 icio
9 ME <- 7
10 escreval ("Digite o nome do aluno")
11 leia (nome)
12 escreval ("Primeira nota")
13 leia (N1)
14 escreval ("Segunda nota")
15 leia (N2)
16 escreval ("Terceira nota")
17 leia (N3)
18 escreval ("Quantidade de faltas")
19 leia (faltas)
20 MA <- ((N1+N2+N3)/3)
21 escreval (nome, " média = ", MA)
22 escreva("Média acima de 7 e faltas abaixo de 25? ",(MA>ME) e (faltas<25))
23
24
25 algoritmo
26
27
28
29
30
31
32

```

On the right, the "Área das variáveis de memória (Globais e Locais)" window shows variable values:

Escopo	Nome	Tipo	Valor
GLOBAL	N1	R	6,00000000000000
GLOBAL	N2	R	7,00000000000000
GLOBAL	N3	R	9,00000000000000
GLOBAL	MA	R	7,33333333333333
GLOBAL	ME	R	7,00000000000000
GLOBAL	FALTAS	I	10
GLOBAL	NOME	C	"Marquinhos"

The "Área de visualização dos resultados" window shows the execution output:

```

6.0000000000
Segunda nota
7.0000000000
Terceira nota
9.0000000000
Quantidade de faltas
10
Marquinhos média = 7.33333333333333
Média acima de 7 e faltas abaixo de 25? VERDADEIRO
Fim da execução.

```

4.11. Tela do VisualG – Operadores lógicos

4.4. Estrutura de decisão no VisualG

Já vimos como funciona a Estrutura de Decisão, e, no VisualG, não é muito diferente. Segue a estrutura básica que somente é executada se a expressão lógica for verdadeira:

```

se <expressão-lógica> então
    <sequência-de-comandos>
fimse

```

E, no caso da próxima estrutura, se a expressão lógica não for verdadeira, a sequência de comandos do SENÃO será processada.

```

se <expressão-lógica> então
    <sequência-de-comandos-1>
senão
    <sequência-de-comandos-2>
fimse

```

Continuaremos a modificar o algoritmo de lançamento de notas utilizado em outros exercícios. Mas, agora, utilizaremos a estrutura de decisão para mostrar ao usuário a condição do aluno de **Aprovado** ou **Reprovado**. Observe que, na condição lógica do SE, utilizamos o operador lógico E. O algoritmo e a tela do VisualG ficarão respectivamente como as imagens 4.12 e 4.13.

```

1 Algoritmo "Operador lógico e"
2
3 Var
4   N1, N2, N3, MA, ME : real
5   faltas: inteiro
6   nome: caractere
7
8 Inicio
9   ME <- 7
10  escreval ("Digite o nome do aluno")
11  leia (nome)
12  escreval ("Primeira nota")
13  leia (N1)
14  escreval ("Segunda nota")
15  leia (N2)
16  escreval ("Terceira nota")
17  leia (N3)
18  escreval ("Quantidade de faltas")
19  leia (faltas)
20  MA <- ((N1+N2+N3)/3)
21  escreval (nome, " média = ", MA)
22  SE (MA>ME) e (faltas < 25) ENTÃO
23  escreval ("Aprovado")
24  SENÃO
25  escreval ("Reprovado")
26  FIMSE
27
28
29 Fimalgoritmo
30
31
32

```

4.12. Algoritmo – Estrutura de decisão

The screenshot shows the VisualG IDE with the algorithm code from image 4.12. The execution window displays the following interaction:

```

Digite o nome do aluno
Huguenho
Primeira nota
6
Segunda nota
6.5
Terceira nota
7
Quantidade de faltas
15
Huguenho média = 7.166666666666667
Aprovado
>>> Fim da execução do programa !

```

On the right, the "Área das variáveis de memória (Globais e Locais)" table shows the variable values:

Escopo	Nome	Tipo	Valor
GLOBAL	N1	R	6,000000000000000
GLOBAL	N2	R	6,500000000000000
GLOBAL	N3	R	7,000000000000000
GLOBAL	MA	R	7,166666666666667
GLOBAL	ME	R	7,000000000000000
GLOBAL	FALTAS	I	15
GLOBAL	NOME	C	"Huguenho"

The "Área de visualização dos resultados" window shows the output of the program execution:

```

Segunda nota
6.500000000000000
Terceira nota
7.000000000000000
Quantidade de faltas
15
Huguenho média = 7.166666666666667
Aprovado
Fim da execução.

```

4.13. Tela do VisualG – Estrutura de decisão

O VisualG também tem o comando escolha que apresenta várias opções para o usuário (seleção encadeada) como se fosse um menu. Segue a sintaxe ou estrutura:

```
escolha <expressão-de-seleção>
caso <exp11>, <exp12>, ..., <exp1n>
    <sequência-de-comandos-1>
caso <exp21>, <exp22>, ..., <exp2n>
    <sequência-de-comandos-2>
...
outrocaso
    <sequência-de-comandos-extra>
fimescolha
```

Para mostrar como funciona essa estrutura no VisualG, faremos o seguinte exercício:

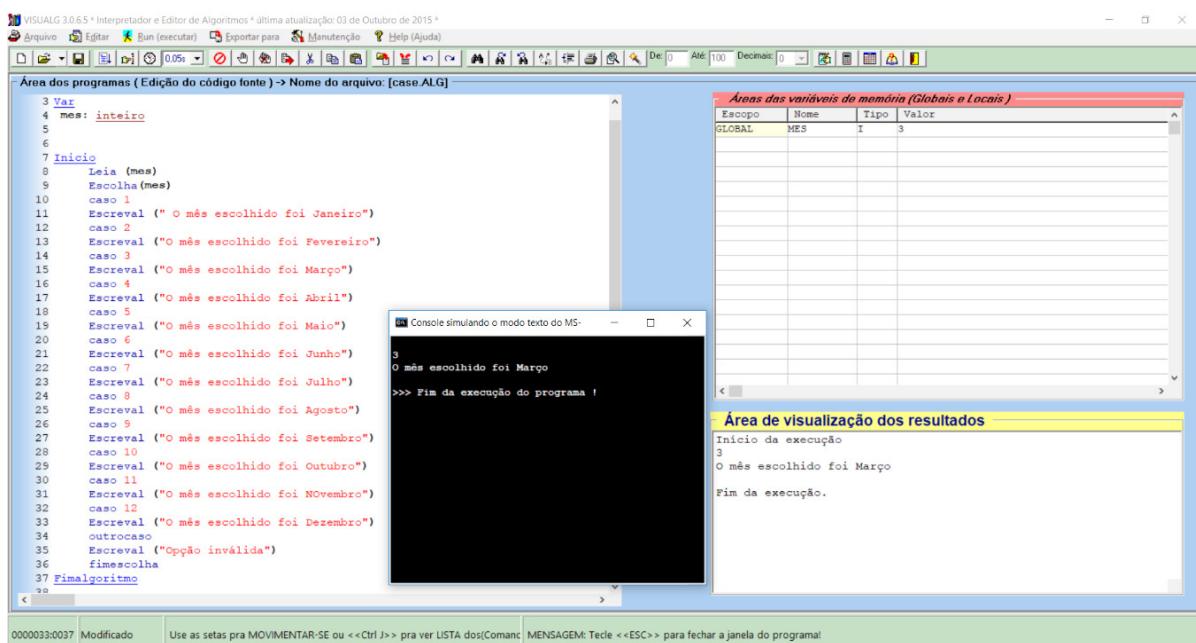
Quando o usuário escolher um número de 1 a 12, aparecerá como resultado o Mês correspondente ao número escolhido, por exemplo, se ele escolheu 5, então aparecerá Maio, e assim por diante.

O algoritmo ficará como a imagem 4.14. Caso escolha o número 3, a tela ficará como a imagem 4.15.

The screenshot shows the VisualG 3.0.6.5 software interface. The title bar reads "VISUALG 3.0.6.5 * Interpretador e Editor de Algoritmos * última atualização: 03 de Outubro". The menu bar includes "Arquivo", "Editar", "Run (executar)", "Exportar para", and "Manutenção". The main window is titled "Área dos programas (Edição do código fonte) -> Nome do arquivo:". The code in the editor is:

```
3 Var
4 mes: inteiro
5
6
7 Inicio
8     Leia (mes)
9     Escolha (mes)
10    caso 1
11        Escreval ("O mês escolhido foi Janeiro")
12    caso 2
13        Escreval ("O mês escolhido foi Fevereiro")
14    caso 3
15        Escreval ("O mês escolhido foi Março")
16    caso 4
17        Escreval ("O mês escolhido foi Abril")
18    caso 5
19        Escreval ("O mês escolhido foi Maio")
20    caso 6
21        Escreval ("O mês escolhido foi Junho")
22    caso 7
23        Escreval ("O mês escolhido foi Julho")
24    caso 8
25        Escreval ("O mês escolhido foi Agosto")
26    caso 9
27        Escreval ("O mês escolhido foi Setembro")
28    caso 10
29        Escreval ("O mês escolhido foi Outubro")
30    caso 11
31        Escreval ("O mês escolhido foi Novembro")
32    caso 12
33        Escreval ("O mês escolhido foi Dezembro")
34    outrocaso
35        Escreval ("Opção inválida")
36    fimescolha
37 Fimalgoritmo
```

4.14. Algoritmo – Estrutura de Decisão



4.15. Tela – Estrutura de Decisão

4.5. Estrutura de repetição no VisualG

O VisualG possui três estruturas de repetição:

- Para... faça;
- Enquanto... faça;
- Repita... até.

Segue a sintaxe de cada uma delas:

- **Para... faça**

Repete uma sequência de comandos um determinado número de vezes.

```

para <variável> de <valor-inicial> até <valor-limite> passo <incremento> faça
  <sequência-de-comandos>
fimpara
  
```

Em que:

- **<variável>**: É a variável contadora que controla a quantidade de repetições;
 - **<valor inicial>**: Especifica o valor de inicialização da variável contadora;
 - **<valor limite>**: Valor máximo do contador;
 - **<incremento>**: É opcional a sua determinação, pois, no caso do VisualG, o incremento padrão é 1, ou seja, toda vez que a execução do algoritmo chegar no **fimpara** será acrescido (incrementado) 1 na variável contadora.
- **Enquanto... faça**

Repete uma sequência enquanto a expressão lógica for verdadeira.

```
enquanto <expressão-lógica> faça  
    <sequência-de-comandos>  
fimenquanto
```

- **Repita... até**

Repete uma sequência de comando até que a condição seja verdadeira. Nessa estrutura, ocorre a obrigatoriedade que haja pelo menos um processamento, a sequência de comando que está entre o repita e o até, sendo essa a diferença entre essa estrutura e a estrutura **enquanto**.

```
repita  
    <sequência-de-comandos>  
até <expressão-lógica>
```

É possível utilizarmos essa estrutura no exercício anterior, cálculo das médias das notas dos alunos, que, em vez do "programa" encerrar após o processamento dos dados de um aluno, "automaticamente" serão solicitados os dados do próximo aluno. Com o intuito de o programa não solicitar os dados de muitos alunos, determinaremos uma quantidade fixa, ou seja, um limitador.

Nas imagens 4.16, utilizamos a estrutura **Para... faça** e, nas imagens 4.17, é possível ver o mesmo exercício sendo resolvido com a estrutura **Enquanto... faça**.

Nas duas estruturas, é necessário criar um contador (i). E não esquecer de, na estrutura Enquanto, no término do processamento, incluir uma linha que o contador (variável) soma ele mais um, ou seja, cada vez que o processamento passa pelo laço, o contador conta uma vez, ocorrendo o incremento.

The screenshot shows the VisualG IDE interface. On the left, the code for a 'Para' loop is displayed:

```

1 Algoritmo "Estrutura de repetição"
2
3 Var
4   N1, N2, N3, MA, ME : real
5   faltas: inteiro
6   nome: caractere
7   i: inteiro
8
9 Início
10 PARA i de 1 ate 3 faça
11   ME <- 7
12   escreval ("Digite o nome do aluno")
13   leia (nome)
14   escreval ("Primeira nota")
15   leia (N1)
16   escreval ("Segunda nota")
17   leia (N2)
18   escreval ("Terceira nota")
19   leia (N3)
20   escreval ("Quantidade de faltas")
21   leia (faltas)
22   MA <- ((N1+N2+N3)/3)
23   escreval (nome, " média = ", MA)
24   SE (MA>ME) e (faltas < 25) ENTÃO
25     escreval ("Aprovado")
26   SENÃO
27     escreval ("Reprovado")
28   FIMSE
29 FIMPARA
30
31
32 Fimalgoritmo
33
34
35
36

```

On the right, the 'Área das variáveis de memória (Globais e locais)' shows variable values:

Escopo	Nome	Tipo	Valor
GLOBAL	N1	R	1,0000000000000000
GLOBAL	N2	R	5,0000000000000000
GLOBAL	N3	R	6,0000000000000000
GLOBAL	MA	R	4,0000000000000000
GLOBAL	ME	R	7,0000000000000000
GLOBAL	FALTAS	I	5
GLOBAL	NAME	C	"jôão"
GLOBAL	i	I	3

The 'Área de visualização dos resultados' displays the execution output:

```

Segunda nota
5.0000000000
Terceira nota
6.0000000000
Quantidade de faltas
5
jôão média = 4
Reprovado

Fim da execução.

```

4.16. Tela – Estrutura de Repetição – Para

The screenshot shows the VisualG IDE interface. On the left, the code for an 'Enquanto' loop is displayed:

```

1 Algoritmo "Estrutura de repetição"
2
3 Var
4   N1, N2, N3, MA, ME : real
5   faltas: inteiro
6   nome: caractere
7   i: inteiro
8
9 Início
10 i<- 1
11 ENQUANTO i <= 3 faça
12   ME <- 7
13   escreval ("Digite o nome do aluno")
14   leia (nome)
15   escreval ("Primeira nota")
16   leia (N1)
17   escreval ("Segunda nota")
18   leia (N2)
19   escreval ("Terceira nota")
20   leia (N3)
21   escreval ("Quantidade de faltas")
22   leia (faltas)
23   MA <- ((N1+N2+N3)/3)
24   escreval (nome, " média = ", MA)
25   SE (MA>ME) e (faltas < 25) ENTÃO
26     escreval ("Aprovado")
27   SENÃO
28     escreval ("Reprovado")
29   FIMSE
30   i<- i+1
31 FIMENQUANTO
32
33
34 Fimalgoritmo
35
36

```

On the right, the 'Área das variáveis de memória (Globais e locais)' shows variable values:

Escopo	Nome	Tipo	Valor
GLOBAL	N1	R	8,0000000000000000
GLOBAL	N2	R	4,0000000000000000
GLOBAL	N3	R	9,0000000000000000
GLOBAL	MA	R	6,6666666666666667
GLOBAL	ME	R	7,0000000000000000
GLOBAL	FALTAS	I	5
GLOBAL	NAME	C	"Rosa"
GLOBAL	i	I	4

The 'Área de visualização dos resultados' displays the execution output:

```

Quantidade de faltas
10
Rosa média = 6
Reprovado
Digite o nome do aluno
Rosa
Primeira nota
5
Segunda nota
4
Terceira nota
9
Quantidade de faltas
5
Rosa média = 6,6666666666666667
Reprovado

>>> Fim da execução do programa !

```

4.17. Algoritmo – Estrutura de Repetição – Enquanto

Agora, faremos um outro exercício:

Crie um algoritmo que mostre ao usuário se o número digitado é ímpar ou par e somente deverá parar o processamento quando o número digitado for ímpar.

The screenshot shows the VISUALG 3.0.6.5 IDE interface. The code editor window displays the following pseudocode:

```
1 Algoritmo "Numeros pares e impares"
2
3 Var
4 num1:real
5
6
7 Início
8     Escreval ("Digite alguns números até encontrar um número ímpar")
9     enquanto num1 % 2 = 0 Faca
10        Escreval ("Digite um número")
11        leia(num1)
12        Se (num1 % 2 = 0) ENTAO
13            escreval (num1, " É número par. Mais uma vez!")
14        SENAO
15            escreval (num1, " É número Impar. Você conseguiu")
16        FimSE
17    fim enquanto
18
19 Fimalgoritmo
20
21
22
```

The variable table (Área das variáveis de memória) shows:

Escopo	Nome	Tipo	Valor
GLOBAL	NUM1	R	7,00000000000000

The results visualization area (Área de visualização dos resultados) shows the execution output:

```
4.0000000000
4 É número par. Mais uma vez!
Digite um número
6.0000000000
6 É número par. Mais uma vez!
Digite um número
7.0000000000
7 É número Impar. Conseguiu
Fim da execução.
```

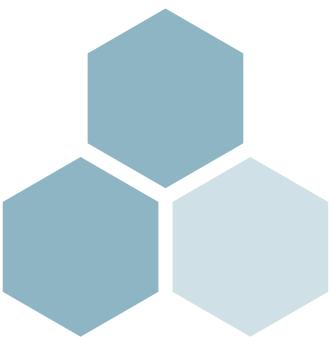
4.18. Tela - Exemplo Par ou Ímpar



5

Linguagem Python

- ◆ Conhecendo o Python;
- ◆ Estrutura básica em Python;
- ◆ Operadores aritméticos, lógicos e relacionais;
- ◆ Estrutura de decisão;
- ◆ Estrutura de repetição;
- ◆ Estrutura de dados.



5.1. Conhecendo o Python

A programação de computadores ocorre por meio de uma ou mais linguagens de programação e, a partir de agora, veremos como as estruturas que aprendemos ao longo deste curso são utilizadas numa linguagem que cada vez mais está crescendo: Python. Mas, antes de começarmos a falar sobre a linguagem Python, veremos mais alguns conceitos importantes.

Linguagem de programação é um meio padronizado de instruir o computador. E essa padronização depende de cada linguagem, ou seja, se um programador usa a estrutura de repetição na linguagem Java e, depois, precisa usar a mesma estrutura na linguagem Python, não poderá "copiar e colar" o código, mesmo que o raciocínio e o resultado sejam o mesmo. A estrutura de cada linguagem é diferente, mesmo havendo algumas semelhanças.

Existem centenas de linguagens de programação desde que o computador foi inventado, sendo que muitas já não são mais usadas. Há várias diferenças entre elas. Comentaremos sobre algumas dessas diferenças.

As linguagens são de alto nível ou de baixo nível. No primeiro caso, o nível de abstração é relativamente elevado, ou seja, a programação é mais próxima da linguagem humana e não da linguagem da máquina, e, no segundo caso, tratam-se das linguagens que o programador trabalha diretamente com os registradores do processador, como é o caso da linguagem Assembly.

Há os compiladores e interpretadores que traduzem um código geralmente de alto nível para um código que o computador interprete (código binário).

Algumas linguagens são compiladas, ou seja, sofrem um processo de tradução (compilação) para serem transformadas para um outro código que o processador do computador entenderá (código de máquina). O programa que faz essa "conversão" é o compilador.

O código compilado (código objeto) pode ser um arquivo executável que é reproduzido em um sistema operacional, e este tipo de tradutor é um dos mais utilizados. Outras linguagens são interpretadas onde são executadas (interpretadas) por um programa que chama interpretador, depois, é executada pelo processador do computador. O interpretador traduz o programa linha a linha e o programa vai sendo utilizado na medida em que vai sendo traduzido.

Escolhemos a linguagem Python porque é uma linguagem que está crescendo muito e é extremamente versátil, usada tanto no desenvolvimento Web quanto em muitos outros tipos de aplicação. Sua sintaxe é simples e clara, oferecendo recursos que não estão aquém de outras linguagens consideradas mais complicadas, como Java e C++. Python é um software livre podendo ser utilizado gratuitamente, graças ao trabalho da Python Foundation e de seus colaboradores.

Atenção:

Neste curso, mostraremos somente algumas sintaxes (estruturas) básicas em Python, então, caso haja interesse em se aprofundar, recomendamos cursos mais avançados.

O Python foi criado por Guido van Rossum, no Centro de Matemática e Tecnologia da Informação na Holanda, sendo uma linguagem de alto nível e interpretada.

Antes de iniciarmos a programar, é necessário instalar o interpretador da linguagem Python. O interpretador Python não vem instalado com o Microsoft Windows, mas, se você utiliza Mac ou Linux, não precisará instalá-lo. Essas informações e o download do interpretador estão no site: <<https://python.org.br/>>.

No Windows, após a instalação do interpretador, vá até o menu Iniciar e abra o IDLE, que é uma interface gráfica para o interpretador do Python.

Há outros programas que são utilizados no lugar do IDLE como interface gráfica para o Python.

Quando a janela inicial do IDLE é aberta, você já pode escrever e executar o seu algoritmo, mas essa tela, chamada de Python Shell, é o local onde o algoritmo será executado. Para facilitar o aprendizado, os algoritmos do nosso curso serão escritos no Editor de Texto do Python e, depois, executados no Python Shell.

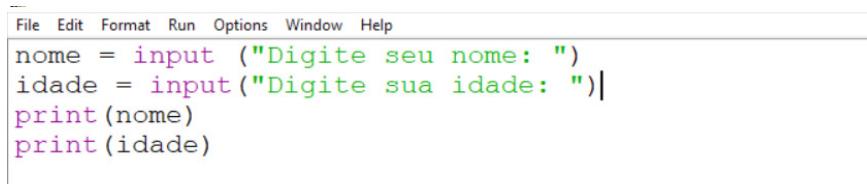
Para abrir esse Editor de Texto, no menu da tela do Python Shell, clique em **File / New File**; em seguida, abrirá um editor de texto.

No editor de texto, antes de executar o código, é obrigatório salvar o arquivo no computador (**File / Save as**) e, depois, poderá executá-lo (**Run / Run Module F5**).

5.2. Estrutura básica em Python

O comando de entrada e saída em Python são respectivamente input e print.

No editor de texto do Python, escreveremos o algoritmo conforme a imagem 5.1.



```
File Edit Format Run Options Window Help
nome = input ("Digite seu nome: ")
idade = input("Digite sua idade: ")
print(nome)
print(idade)
```

5.1. Tela - Idade

Observe que foram criadas duas variáveis (nome e idade) que receberão (comando input) a resposta do usuário referente às solicitações que estão dentro dos parentes. Veja que no Python não há necessidade de declarar as variáveis no início, e a entrada de dados é simples e direta, sem rodeios.

O comando print (saída) é responsável por apresentar ao usuário o resultado do processamento. Nesse caso será apresentado o conteúdo de uma variável, então não usaremos as aspas, mas se fosse strings (texto), as usaríamos.



Não se esqueça:

Após a digitação do algoritmo no editor de texto do Python, é obrigatório salvar o arquivo no computador (**File / Save as**) e, depois, poderemos executá-lo (**Run / Run Module F5**) e, no Shell, aparecerá o resultado do processamento.

5.3. Operadores aritméticos, relacionais e lógicos

Os operadores aritméticos em Python funcionam da mesma forma como já conhecemos no nosso curso:

- Adição +
- Subtração -
- Divisão /
- Multiplicação *
- Exponencial **
- Resto ou módulo %

No próximo exercício, solicitaremos ao usuário de uma empresa as seguintes informações do funcionário: nome, sobrenome, salário, cargo, aumento que o funcionário irá ter, e o programa deverá retornar ao usuário todos os dados do funcionário, inclusive o salário atual e o salário com o aumento. O algoritmo poderá ficar como a imagem 5.2.

É possível resolver esse exercício utilizando uma quantidade menor de linhas de código, e isso dependerá da sua lógica (raciocínio), então tente resolver de outras maneiras. Exercite!

```
File Edit Format Run Options Window Help
nome = input ("Informe o nome do funcionário: ")
sobrenome = input("Informe o sobrenome do funcionário : ")
salario = float (input("Informe o salário: "))
cargo = input("Informe o cargo: ")
aumento = float (input ("Informe a porcentagem do aumento salarial: "))
salario_aum = salario + (salario * (aumento/100))
print(nome, sobrenome)
print("Cargo: ", cargo," |Salário atual: R$ ",salario)
print ("Cargo: ", cargo," |Salário com aumento: R$ ",salario_aum)
```

5.2. Algoritmo – Aumento salarial – Operador numérico

Você deve ter percebido que, no algoritmo da imagem 5.2, apareceu uma palavra diferente, o **float**. Você se lembra de quando falamos sobre o tipo de dados no capítulo anterior? Então, o **float** é um tipo de dados numéricos que também aceita casas decimais, e o tipo de dados que não aceita números decimais é o **int**.

Como já comentamos, em Python não declaramos as variáveis antes do processamento, então, quando a variável irá receber algo (input), definimos o tipo de dados que será recebido. Mas atenção: se não determinarmos o tipo de dados, será processado o tipo de dados padrão, **String** (texto), ou seja, o programa não conseguirá realizar cálculos com o conteúdo dessas variáveis.

Os operadores relacionais suportados em Python não são muito diferentes daqueles que já vimos no nosso curso:

- Igualdade ==
- Maior que >
- Menor que <
- Diferente !=
- Maior ou igual >=
- Menor ou igual <=

Continuando com o exercício de aumento de salário, agora apresentaremos ao usuário se o salário após o aumento é maior que R\$ 1000,00. O algoritmo é demonstrado na imagem 5.3.

```
File Edit Format Run Options Window Help
nome = input ("Informe o nome do funcionário: ")
sobrenome = input("Informe o sobrenome do funcionário : ")
salario = float (input("Informe o salário: "))
cargo = input("Informe o cargo: ")
aumento = float (input ("Informe a porcentagem do aumento salarial: "))
salario_aum = salario + (salario * (aumento/100))
print(nome, sobrenome)
print("Cargo: ", cargo, " |Salário atual: R$ ",salario)
print ("Cargo: ", cargo, " |Salário com aumento: R$ ",salario_aum)
print ("O salário com aumento é maior que R$ 1000.00?", salario_aum > 1000)
```

5.3. Algoritmo – Aumento salarial – Operador relacional

Os operadores lógicos em Python são o not (não), and (e), or (ou). E o comportamento deles é conforme a Tabela Verdade, que já discutimos anteriormente.

Na imagem 5.4, podemos ver a utilização do operador and para verificar se o aumento foi acima de 2% e se o campo cargo está diferente de vazio ("").

```
File Edit Format Run Options Window Help
nome = input ("Informe o nome do funcionário: ")
sobrenome = input("Informe o sobrenome do funcionário : ")
salario = float (input("Informe o salário: "))
cargo = input("Informe o cargo: ")
aumento = float (input ("Informe a porcentagem do aumento salarial: "))
salario_aum = salario + (salario * (aumento/100))
print(nome, sobrenome)
print("Cargo: ", cargo, " |Salário atual: R$ ",salario)
print ("Cargo: ", cargo, " |Salário com aumento: R$ ",salario_aum)
print ("O salário com aumento é maior que R$ 1000.00?", salario_aum > 1000)
print ("O aumento foi acima de 2% e o cargo foi informado: ", (aumento > 2) and (cargo != ""))
```

5.4. Algoritmo – Aumento salarial – Operador lógico

5.4. Estrutura de decisão

Em Python, também é possível utilizarmos as estruturas de decisão. Seguem as sintaxes:

```
If <condição>:  
    Bloco verdadeiro
```

- Se (if);
- Se... então (If... else...);
- Estrutura aninhada: if... elif... else.

! Não há um limite de elif, podemos usar quantos forem necessários.

```
If <condição>:  
    Bloco verdadeiro  
Else:  
    Bloco
```

```
If <condição>:  
    Bloco verdadeiro  
Elif <condição>:  
    Bloco verdadeiro  
Elif <condição>:  
    Bloco verdadeiro  
Elif <condição>:  
    Bloco verdadeiro  
Else:  
    Bloco
```

Atenção:

Em Python, os blocos de instrução são delimitados ou agrupados pela endentação (tabulação), ou seja, utiliza o deslocamento do texto à direita (recuo/tabulação) para marcar o início ou o fim de um bloco de instrução. Você verá nos próximos exercícios. Por exemplo: as instruções If e Else devem estar alinhados, senão não serão processados. Não devem ser esquecidos, também, os dois-pontos (:); sem eles o Python não reconhecerá a instrução.

Na imagem 5.5, utilizamos a estrutura **if** para emitir uma mensagem para o usuário se os campos cargo ou sobrenome estiverem vazios (`==""`).

```
File Edit Format Run Options Window Help
nome = input ("Informe o nome do funcionário: ")
sobrenome = input("Informe o sobrenome do funcionário : ")
salario = float (input("Informe o salário: "))
cargo = input("Informe o cargo: ")
if cargo == "" or sobrenome == "":
    print(" Atenção: O Sobrenome e o Cargo devem ser informados")
aumento = float (input ("Informe a porcentagem do aumento salarial: "))
salario_aum = salario + (salario * (aumento/100))
print(nome, sobrenome)
print("Cargo: ", cargo, " |Salário atual: R$ ",salario)
print ("Cargo: ", cargo, " |Salário com aumento: R$ ",salario_aum)
```

5.5. Algoritmo – Estrutura Decisão – If

Na imagem 5.6, a estrutura **if... else** verifica a seguinte condição: Se o aumento for maior que 2%, é apresentado o salário calculado, mas se o aumento for menor que 2%, então o cálculo não é efetuado e emite-se uma mensagem ao usuário.

```
File Edit Format Run Options Window Help
nome = input ("Informe o nome do funcionário: ")
sobrenome = input("Informe o sobrenome do funcionário : ")
salario = float (input("Informe o salário: "))
cargo = input("Informe o cargo: ")
if cargo == "" or sobrenome == "":
    print(" Atenção: O Sobrenome e o Cargo devem ser informados")
aumento = float (input ("Informe a porcentagem do aumento salarial: "))
salario_aum = salario + (salario * (aumento/100))
print(nome, sobrenome)
print("Cargo: ", cargo, " |Salário atual: R$ ",salario)
if aumento > 2:
    print ("Cargo: ", cargo, " |Salário com aumento: R$ ",salario_aum)
else:
    print ("O aumento deverá ser acima de 2% ")
```

5.6. Algoritmo – Estrutura Decisão – If... Else

Utilizaremos, também, a estrutura aninhada, quando são apresentadas várias opções (condições) ao usuário. Na imagem 5.7, o algoritmo apresentado classifica os salários após o aumento em diferentes faixas de acordo com o valor calculado anteriormente.



Na estrutura **Elif**, não é obrigatório utilizar o **Else**.

```
File Edit Format Run Options Window Help
nome = input ("Informe o nome do funcionário: ")
sobrenome = input("Informe o sobrenome do funcionário : ")
salario = float (input("Informe o salário: "))
cargo = input("Informe o cargo: ")
if cargo == "" or sobrenome == "":
    print(" Atenção: O Sobrenome e o Cargo devem ser informados")
aumento = float (input ("Informe a porcentagem do aumento salarial: "))
salario_aum = salario + (salario * (aumento/100))
print(nome, sobrenome)
print("Cargo: ", cargo, " |Salário atual: R$ ",salario)
if aumento > 2:
    print ("Cargo: ", cargo, " |Salário com aumento: R$ ",salario_aum)
else:
    print ("O aumento deverá ser acima de 2% ")
if salario_aum < 1000:
    print ("O salário está na faixa 1")
elif salario_aum < 2000:
    print ("O salário está na faixa 2")
elif salario_aum < 3000:
    print ("O salário está na faixa 3")
elif salario_aum > 4000:
    print ("O salário está na faixa 4")
```

5.7. Algoritmo – Estrutura Decisão – If... Elif

5.5. Estrutura de repetição

As estruturas de repetição em Python também são utilizadas para executar a mesma parte de um programa várias vezes de acordo com uma condição.

A estrutura **for in range** possibilita ao programador determinar quantas vezes o código será "percorrido". No caso da imagem 5.8, até não atingir a quantidade determinada, o programa solicitará os dados dos funcionários. E, na figura 5.9, utilizamos a estrutura **While**, que também determinou uma quantidade, mas, nessa estrutura, precisa-se inicializar a variável que será o contador e, no final do processamento, deverá acrescentar 1 ao contador.

```
File Edit Format Run Options Window Help
for i in range(2):
    nome = input ("Informe o nome do funcionário: ")
    sobrenome = input("Informe o sobrenome do funcionário : ")
    salario = float (input("Informe o salário: "))
    cargo = input("Informe o cargo: ")
    if cargo == "" or sobrenome == "":
        print(" Atenção: O Sobrenome e o Cargo devem ser informados")
    aumento = float (input ("Informe a porcentagem do aumento salarial: "))
    salario_aum = salario + (salario * (aumento/100))
    print(nome, sobrenome)
    print("Cargo: ", cargo," |Salário atual: R$ ",salario)
    if aumento > 2:
        print ("Cargo: ", cargo," |Salário com aumento: R$ ",salario_aum)
    else:
        print ("O aumento deverá ser acima de 2% ")
    if salario_aum < 1000:
        print ("O salário está na faixa 1")
    elif salario_aum < 2000:
        print ("O salário está na faixa 2")
    elif salario_aum < 3000:
        print ("O salário está na faixa 3")
    elif salario_aum > 4000:
        print ("O salário está na faixa 4")
```

5.8. Algoritmo – Estrutura Repetição – for

```
File Edit Format Run Options Window Help
i=1
while i<=2:
    nome = input ("Informe o nome do funcionário: ")
    sobrenome = input("Informe o sobrenome do funcionário : ")
    salario = float (input("Informe o salário: "))
    cargo = input("Informe o cargo: ")
    if cargo == "" or sobrenome == "":
        print(" Atenção: O Sobrenome e o Cargo devem ser informados")
    aumento = float (input ("Informe a porcentagem do aumento salarial: "))
    salario_aum = salario + (salario * (aumento/100))
    print(nome, sobrenome)
    print("Cargo: ", cargo," |Salário atual: R$ ",salario)
    if aumento > 2:
        print ("Cargo: ", cargo," |Salário com aumento: R$ ",salario_aum)
    else:
        print ("O aumento deverá ser acima de 2% ")
    if salario_aum < 1000:
        print ("O salário está na faixa 1")
    elif salario_aum < 2000:
        print ("O salário está na faixa 2")
    elif salario_aum < 3000:
        print ("O salário está na faixa 3")
    elif salario_aum > 4000:
        print ("O salário está na faixa 4")
    i=i+1
```

5.9. Algoritmo – Estrutura Repetição – While

5.6. Estrutura de dados

Listas são um tipo de variável que pode armazenar vários valores, e cada valor corresponderá a um índice que iniciará com o zero. Por exemplo: eu tenho uma lista de ingredientes para fazer um bolo [ovos, leite, farinha de trigo], então, na posição (índice) 0, estão os ovos; na posição 1, está o leite; na posição 2, está a farinha de trigo. As listas são muito utilizadas na programação e, neste curso, apresentaremos somente algumas propriedades.

Em Python, as listas são apresentadas dentro de colchetes [] e separadas entre vírgulas.

Na imagem 5.10, é mostrado o algoritmo que cria uma lista chamada **compras** e, posteriormente, adiciona elementos nessa lista (append), depois deleta um elemento da lista que está posição 0 (del) e, em seguida, substitui um elemento da lista.

! Em Python os comentários são inseridos utilizando o #.

```
File Edit Format Run Options Window Help
compras=["banana", "abacaxi", "laranja"]
print (compras) # toda lista
print (compras[1]) # apenas o elemento da posição indicada
compras.append("uva") # incluir elemento na lista
print (compras)
del compras[0] # remove o elemeto da posição indicada
print (compras)
compras[1]="abacate" # substitui o elemento da posição indicada
print (compras)
```

5.10. Algoritmo – Lista

