

# Curso Técnico em Informática para Internet



Modulo Introdutório

Lógica de Programação



*Serviço Nacional de Aprendizagem Industrial*  
**PELO FUTURO DO TRABALHO**

Hilton Elias

# LÓGICA DE PROGRAMAÇÃO

## O QUE É LÓGICA?

A lógica é o campo de estudo que utiliza princípios e conhecimentos para se atingir um raciocínio correto.



O filósofo Copi (1978) define lógica como:


***"estudo dos métodos e princípios usados para distinguir o raciocínio correto do incorreto."***

A utilização do raciocínio lógico é tão comum e natural ao ser humano que nem percebemos.

Pense, por exemplo:

Quais são os passos necessários para passar por uma porta que está trancada?

Para isso, deve-se executar a seguinte sequência:

An illustration showing a yellow hand inserting a silver key into a silver door handle with a keyhole. The background is a solid red color.

1. Colocar a chave na fechadura.
2. Girar a chave para destrancar a porta.
3. Girar a maçaneta.
4. Puxar a porta para abri-la.

A ação de abrir uma porta trancada parece simples, mas envolve uma sequência lógica de passos. Se você tentar girar a maçaneta, antes de destrancar a porta com a chave, o resultado não será o mesmo.

Podemos citar várias outras situações cotidianas que envolvem raciocínio lógico, como tomar banho, cozinhar e dirigir um carro.

## LÓGICA DE PROGRAMAÇÃO

A lógica de programação surgiu a partir dos princípios da lógica e consiste em uma técnica de encadeamento do pensamento para atingir um determinado objetivo ou solucionar um problema.

Podemos dizer que a lógica é a primeira etapa da programação em si. Isso quer dizer que, antes de começar a escrever o código, você deve pensar quais as questões que devem ser resolvidas, estudar quais as soluções possíveis e planejar todas as etapas da solução.

Para isso, é necessário utilizar uma sequência lógica, que é um conjunto de passos a serem executados.

Quando falamos em sequência lógica, estamos falando de algoritmos.

Saiba que uma das ações mais comuns de um(a) programador(a) é buscar as melhores soluções e encontrar os algoritmos mais adequados para a criação de programas de computadores, soluções e serviços.

## SOFTWARES

Você viu que um software é um agrupamento de instruções ou comandos escritos em uma linguagem de programação que são lidos pelo computador e possibilitam seu funcionamento.

Em outras palavras, software é um produto virtual usado para descrever programas, aplicativos, scripts, macros e instruções de código embarcado diretamente (firmware), a fim de determinar o que uma máquina deve fazer.



Software embarcado ou *firmware* é um conjunto de instruções operacionais programadas pelo fabricante diretamente no hardware do equipamento, para manter a configuração básica das funções.

Isso significa que os códigos transcritos por esse tipo de programa são fundamentais para iniciar e executar os hardwares e seus recursos, fornecendo informações idênticas sempre que o dispositivo for ligado.

---

E você sabe dizer qual é a diferença entre **hardware** e **software**?

- ✓ **Hardware** é a parte física do computador, ou seja, o conjunto de aparatos eletrônicos, peças e equipamentos que fazem o computador funcionar. Monitor, placa de vídeo, processador, mouse, disco rígido e teclado são exemplos de hardware.
- ✓ **Software** é a parte lógica do computador, ou seja, são os programas que fazem com que a máquina funcione, como aplicativos e sistemas operacionais, desenvolvidos por meio de códigos e linguagem de programação.

---

## TIPOS DE SOFTWARES

Os softwares são divididos em três principais categorias, são elas:

**Programação:** São softwares que, a partir de linguagens de programação, como Java, Python, Swift, são utilizados para o desenvolvimento de outros programas.

**Sistema:** Softwares de sistema ou de base permitem a execução de outros softwares, como os de aplicação e os de programação. São responsáveis pela comunicação entre o computador, que só entende linguagem de máquina, e o usuário, ou seja, interpreta nossas ações e as transforma em códigos binários. Sistemas operacionais, como Windows, MacOS, Linux, iOS, Android são exemplos de softwares de sistema.

**Aplicação:** Os softwares de aplicação realizam ações específicas solicitadas pelos usuários. Trata-se de programas que são executados dentro do sistema operacional. Word, Excel, paint, bloco de notas, calculadora e jogos são exemplos deste tipo de software.

## APLICATIVOS

Aplicativos ou App são softwares presentes em dispositivos móveis, como smartphones, tablets, smart TVs, relógios inteligentes, entre outros, que desempenham vários tipos de tarefas.

Apresentam interface amigável e simples para facilitar a interação com o usuário. Alguns já vêm instalados e outros podem ser adquiridos em lojas de aplicativos e podem, ainda, apresentar versões gratuitas ou pagas.



WhatsApp, Uber e Ifood são alguns exemplos desses aplicativos.

## TIPOS DE APLICATIVOS

Dependendo de como serão utilizados, o desenvolvimento de um aplicativo requer recursos e tecnologias específicas.

Atualmente, existem basicamente 3 tipos de aplicativos móveis: **nativos, web e híbridos.**



---

❖ **Nativos** - São desenvolvidos para um determinado sistema operacional, como Android e iOS.

Devem ser programados na linguagem do seu respectivo sistema, como Java e Kotlin no Android e Objective-C e Swift no iOS - mas há também outras linguagens para outros tipos de sistemas operacionais.

Como são programados exclusivamente para um determinado sistema operacional, podem funcionar sem conexão com internet, acessam todos os sensores do dispositivo (câmeras, GPS etc.), são mais rápidos e apresentam melhor experiência para o usuário, o qual consegue acessar todos os seus recursos.

Nesse tipo de aplicação, os programadores utilizam o **Ambiente de Desenvolvimento Integrado** (IDE - Integrated Development Environment, em inglês). Trata-se de um pacote de software que consolida as ferramentas básicas necessárias para escrever e testar software.

---

❖ **Web Apps** - São desenvolvidos para serem abertos no navegador do smartphone.

Trata-se de uma programação que reconhece que o usuário está acessando de um dispositivo móvel e, por isso, o layout adapta-se a ele.

Diferentemente dos aplicativos nativos, os Web Apps necessitam de acesso à internet, não conseguem utilizar todas as funcionalidades do dispositivo e são mais lentos. Sua programação é feita utilizando a linguagem HTMLS, o Cascading Style Sheets (CSS) e o JavaScript.

❖ **Híbridos** - São uma mistura de aplicativo nativo e Web App desenvolvido para ser reconhecido por qualquer sistema operacional. Também são programados por meio da linguagem HTMLS, CSS e JavaScript, assim como o site mobile, mas seu código é alocado dentro de um container, integrando as funcionalidades do dispositivo móvel, o que proporciona uma experiência melhor ao usuário do que os Web Apps.

Podem ser baixados nas lojas de aplicativos e o acesso à internet pode ou não ser necessário. O objetivo desses aplicativos é apresentar uma única versão para vários sistemas.

## SISTEMAS OPERACIONAIS

Um sistema operacional, também chamado de SO, é um tipo de software de sistema que controla praticamente todos os processos de um computador.

Em outras palavras, sistema operacional é o software responsável por fazer a ponte, a interface entre o usuário e o hardware, gerenciando recursos do sistema e do hardware.



Algumas funções básicas dos sistemas operacionais são:

- controlar acesso ao hardware;
- prover interface ao usuário;
- gerenciar programas;
- gerenciar arquivos e pastas.

O usuário interage com a interface do SO, mas não precisa gerenciar os recursos.

Podem-se destacar três estruturas fundamentais para o funcionamento adequado do computador: **shell, kernell e hardware**.



**SHELL** - É a interface de interação com o usuário, ou seja, onde ele realiza a inserção de dados para solicitar as tarefas e ações desejadas.

Há dois tipos de shell:

- **CLI (Comand Line Interface)** - interface de linha de comando, também chamada de modo texto.
- **GUI (Graphical User Interface)** - interface gráfica do usuário, também chamada de modo gráfico.

**KERNEL** - Realiza o papel de interação entre o hardware e o software.

**HARDWARE** - São os elementos físicos, como os componentes eletrônicos.

## CARACTERÍSTICAS ENTRE AS INTERFACES GUI E CLI

**GUI** - Características da interface gráfica do usuário:

- Exibição de tela em um monitor.
- Utilização do mouse para seleção e execução de programas.
- Utilização do teclado para inserção de dados.



## CLI - Características da interface por linha de comando:

- Exibição de tela em um monitor.
- Utilização do teclado para inserção de textos e comandos.
- Utilização do teclado para execução de programas baseados em CLI.



## CATEGORIAS DE SO

O sistema operacional pode ser classificado em duas categorias: para desktops e para servidores

- ✓ **Sistemas operacionais para desktops** - são amplamente utilizados para ambientes de pequeno porte, conhecidos como SOHO (Small Office and Home Office - Escritórios de Pequeno Porte e Domésticos), nos quais há um número pequeno de estações de trabalho que não requerem um servidor para serviços centralizados.

---

Confira algumas características do SO para desktop:

- É compatível apenas com um único usuário.
- Executa aplicações de um único usuário.
- Compartilha arquivos e pastas em uma rede de pequeno porte, com segurança limitada.
- **Sistemas operacionais para servidores** - também conhecidos por NOS (Networking Operational System - Sistema Operacional de Rede), proporcionam diversas funcionalidades e aprimoramentos para que seja possível gerenciar um ambiente de sistemas operacionais desktops em uma rede.

Com esses sistemas, é possível, por exemplo, centralizar e controlar as ações permitidas de um usuário inserido em um domínio e disponibilizar diversos serviços de rede, como atribuição dinâmica de endereçamento IP, resolução de domínios, compartilhamento de arquivos, filas de impressão, entre outros.

---

Confira algumas características do SO para servidores:

- É compatível com vários usuários.
- Executa aplicações multiusuários.
- Fornece mais segurança, se comparado ao SO para desktops.

Os sistemas operacionais para desktops mais comuns são: Windows (Microsoft), Linux (é de código aberto), MacOS (Apple).

- ✓ **Windows (Microsoft)** - surgiu no começo da década de 1980 e teve diversas versões ao longo dos anos.

Intuitivo, de interface gráfica simples e amigável, com uma suíte de softwares de aplicação ampla e compatível com computadores de diversos fabricantes, o Windows se tornou o SO para desktop residencial mais popular, tanto que a maioria dos computadores domésticos já vem com ele instalado.

A Microsoft também oferece uma versão Windows para servidores.



- 
- ✓ **MacOS (Apple)** - anteriormente Mac OS X e posteriormente OS X - da Apple também surgiu na década de 1980 e é o sistema operacional de todos os computadores Apple.

Um grande impeditivo para a popularização é a falta de compatibilidade com dispositivos de outras marcas. A Apple também oferece uma versão para servidores, chamada macOS Server (anteriormente Mac OS X Server e OS X Server).

- ✓ **Linux** É um sistema operacional de código aberto, o que significa que pode ser modificado e distribuído por qualquer pessoa.

Apesar de gratuito, não é muito usado em computadores domésticos porque exige um certo grau de conhecimento para instalação.

Em contrapartida, o Linux é hegemônico entre os servidores de empresas, pois é fácil de personalizar. As versões mais populares são Ubuntu, Debian, Linux Mint e Fedora.

## DRIVERS

Os drivers são softwares ou pequenos elementos de softwares que ligam o SO aos dispositivos físicos do computador.

Os drivers são responsáveis pela comunicação entre os sistemas operacionais e um componente de hardware.



Então, se você conectar um dispositivo, como uma impressora ou uma placa de vídeo, e não instalar o driver apropriado, o computador pode não reconhecer aquele componente que foi conectado, e não poderá utilizá-lo.

Você pode acessar os drivers de seu computador por meio do gerenciador de dispositivos.

## Importante!



Compreender as diferenças entre hardware, software e driver e os conceitos apresentados irão permitir a você, programador ou programadora, o melhor gerenciamento dos recursos e a manipulação dos itens necessários para você desenvolver o seu projeto. Isso fará com que você compreenda, ao desenvolver um código lógico, como o seu computador estará processando e compartilhando essa informação.

Ao desenvolver um projeto prático, utilizar recursos de maneira errada pode prejudicar a performance do seu software e do seu hardware.

## ALGORITMOS

Conforme você viu, o desenvolvimento de qualquer software é realizado por meio dos algoritmos, descrevendo uma sequência de passos lógicos necessários para a execução de uma tarefa que consiste em:



É importante salientar que uma tarefa pode ser composta de várias pequenas ações e cada uma dessas ações tem seu próprio conjunto de instruções ou algoritmo a ser seguido.

Para exemplificar, considere o seguinte problema:

---

## NOTAS

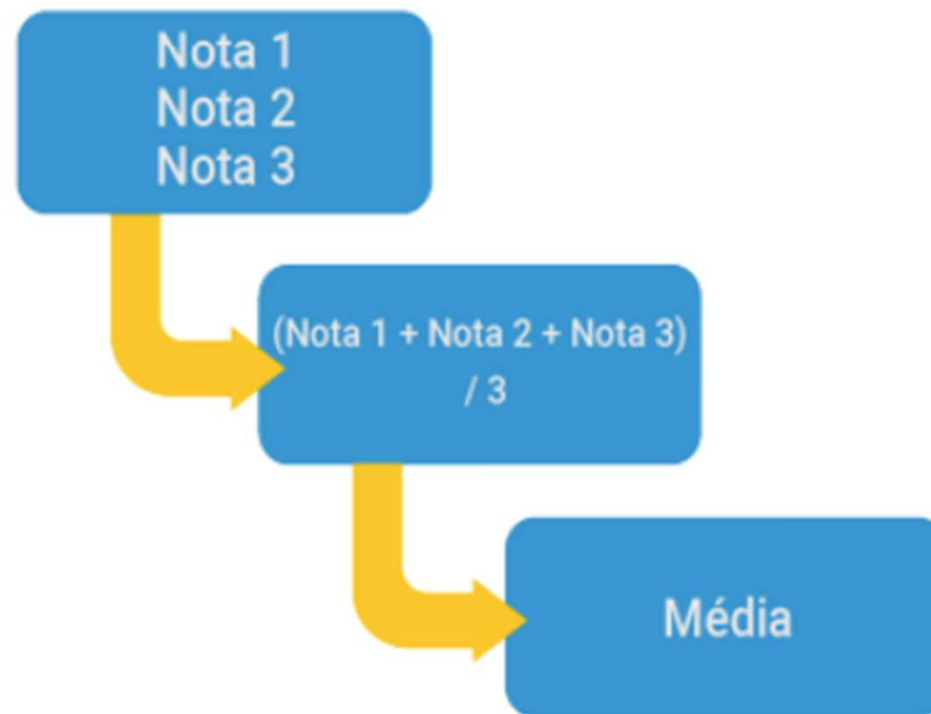
Você precisa criar um algoritmo que, a partir da soma de 3 notas semestrais de um aluno, calcule sua média final para, na sequência, informar se o aluno foi reprovado ou aprovado, considerando as seguintes referências:

- de 7 a 10 - aprovado.
- abaixo de 7 - reprovado.

Para resolução desse problema, serão necessários dois algoritmos, sendo que cada um resolverá um pequeno problema da situação.

## 1º ALGORITMO

Calculará a média do aluno. Ele deve possuir como entrada as 3 notas obtidas durante o semestre. O processamento deverá realizar o cálculo da média, somando as notas e dividindo o resultado por 3. Por fim, a saída (ou resultado) será a média obtida pelo aluno.



## 2º ALGORITMO

Verificará a situação final do aluno. Para isso, a entrada deste algoritmo será a média obtida com o primeiro algoritmo. O processamento será a verificação dessa média, para descobrir se foi maior ou igual a 7 ou menor que 7. Com essa verificação, é possível definir a saída deste algoritmo, informando se o aluno foi aprovado ou não.



## E COMO REALIZAR A REPRESENTAÇÃO DOS ALGORITMOS?

A representação dos algoritmos pode ser realizada por meio de descrição narrativa, fluxograma e pseudocódigos. Siga em frente para conhecer cada uma delas.

### DESCRIÇÃO NARRATIVA

Utiliza palavras para expressar os algoritmos e, por isso, é exclusivamente usada para fins didáticos. O algoritmo apresentado para fazer café é um exemplo desse tipo de representação.

O exemplo abaixo mostra o algoritmo do problema sobre notas, representado por meio da descrição narrativa:





1. Obter as 3 notas do aluno.
2. Somar as 3 notas.
3. Dividir o resultado por 3.
4. Se a média for maior ou igual a 7 = aluno foi aprovado.
5. Se a média for menor que 7 = aluno foi reprovado.

## FLUXOGRAMA




É um tipo de diagrama ou uma representação gráfica que descreve as diferentes ações a serem realizadas durante a execução de um algoritmo.




Ele auxilia na elaboração do raciocínio lógico a ser seguido para a resolução de um problema, mostrando visualmente como o nosso código deve se comportar nas diversas situações e as diferentes saídas que ele terá dentro do nosso programa.

Por exemplo: não temos a água para realizar o café, o que fazer?

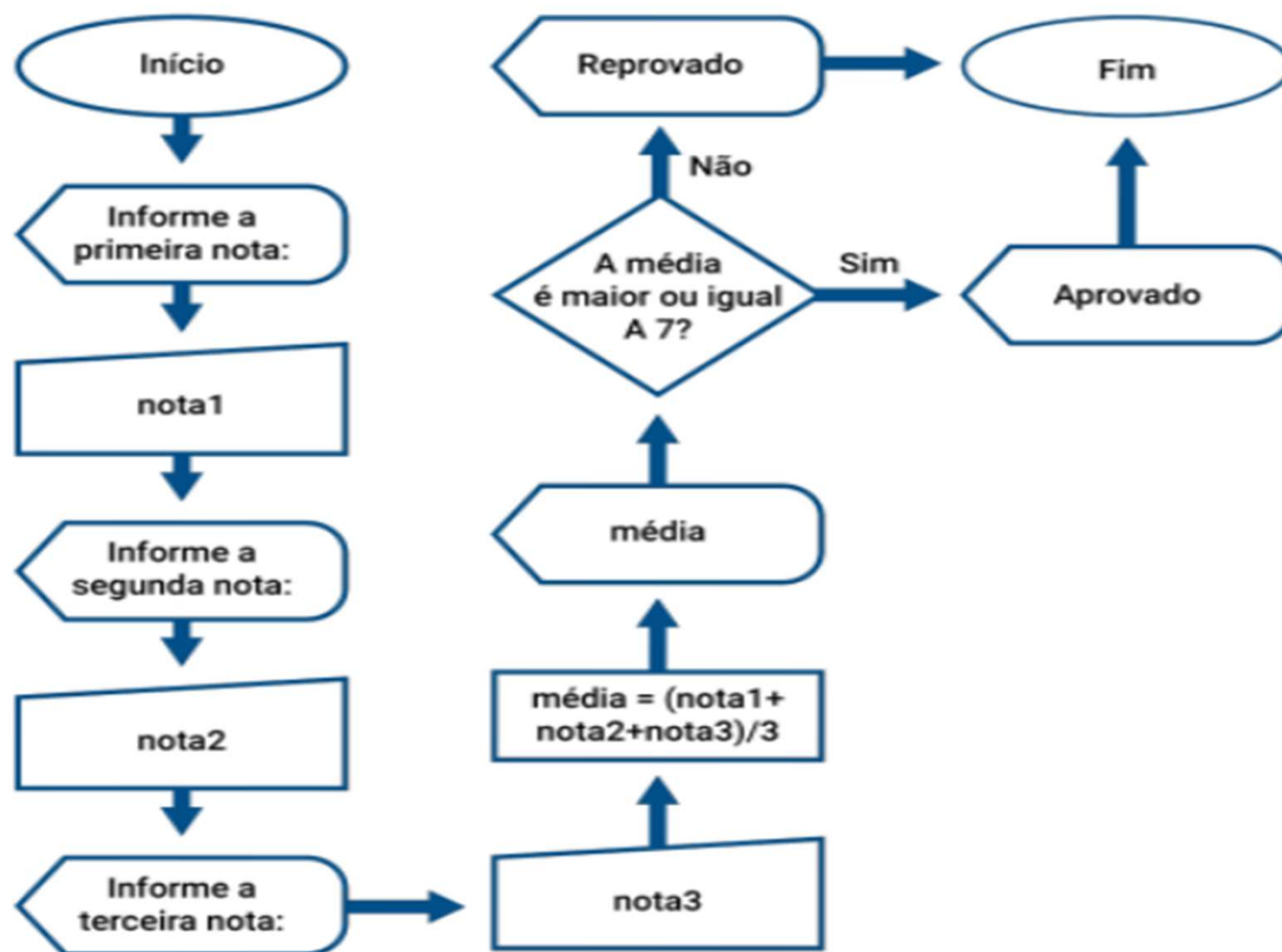
Dessa forma, usando as linguagens de programação, escreveremos ações para que nosso código tenha o comportamento específico, exatamente como exibido no fluxograma, e seja concluído com sucesso.

A tabela mostra as simbologias mais comuns utilizadas para representação de algoritmos por meio de fluxogramas:

Representação gráfica	Item	Descrição
	Início / fim	Todo fluxograma deve iniciar e encerrar com este símbolo. O fluxograma deve conter apenas um início; porém, poderá possuir mais de um fim, pois pode se dividir durante o processo.
	Leitura	Representa uma entrada do usuário, quando o programa fará uma leitura de uma informação digitada pelo usuário.
	Escrita	Representa a impressão de alguma informação na tela pelo programa com o objetivo de informar o usuário.

	Seta de fluxo	Representa o caminho do fluxograma. A partir do início, a leitura do fluxograma é feita seguindo as setas do fluxo.
	Processo	Utilizado quando algo deve ser processado pelo programa, por exemplo, para cálculos matemáticos.
	Decisão	Divide a execução do fluxograma em dois caminhos. Sempre que for utilizado, uma pergunta deve ser feita. Caso a resposta seja verdadeira, o fluxograma segue por um caminho, caso contrário, segue por outro. Essa é a única situação em que devem ser utilizadas duas setas de fluxo a partir de uma figura geométrica.

O fluxograma a seguir mostra a representação do algoritmo do problema sobre notas:



Perceba que o fluxograma mostra o passo a passo de cada ação. Primeiro, solicita-se as 3 notas, utilizando um nome genérico, como notas 1, 2 e 3. Depois, é realizado um processo para calcular a média [ $\text{média} = (\text{nota 1} + \text{nota 2} + \text{nota 3}) / 3$ ], cujo resultado fornece a média do aluno.

Além de calcular a média, o fluxograma também apresentará se o aluno foi reprovado ou aprovado, considerando a média 7 para aprovação.

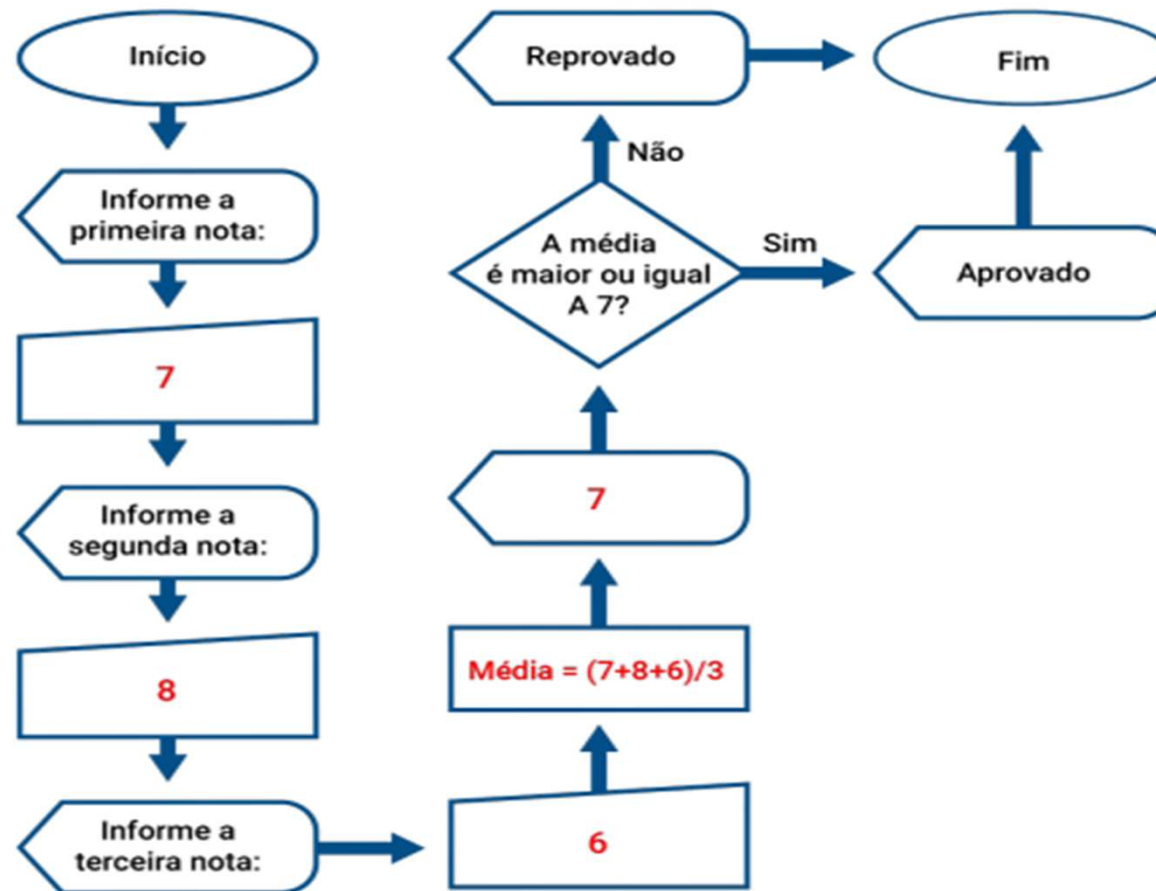
## TESTANDO NA PRÁTICA

Para testar a execução e a eficácia do fluxograma, simule o passo a passo e identifique se o resultado final está correto ou não, utilizando os seguintes valores de notas:

1. Nota 1: 7
2. Nota 2: 8
3. Nota 3: 8

E então? Qual foi a média desse aluno? Ele foi aprovado ou reprovado?

Se você encontrou a média 7, parabéns você acertou! O aluno foi aprovado. Caso você tenha encontrado outro valor, refaça os cálculos e consulte o fluxograma abaixo.



## PSEUDOCÓDIGOS

É uma forma de representação de algoritmo semelhante à linguagem de programação, porém utilizando palavras no idioma escolhido.

O pseudocódigo apresenta 3 etapas:

1. Identificação do algoritmo.
2. Declaração das variantes.
3. Corpo do algoritmo.



A seguir, veja o exemplo do cálculo da média representado pelo pseudocódigo:

```
1. algoritmo "CalcularMédia"  
2. var  
3. nota1, nota2, nota3, media: real  
4. início  
5. escrever ("digite a primeira nota:")  
6. Ler(nota1)  
7. escrever ("digite a segunda nota:")  
8. Ler(nota2)  
9. escrever ("digite a terceira nota:")  
10. Ler (nota3)  
11.  $media \leftarrow (nota1 + nota2 + nota3) / 3$   
12. escrever (media)  
13. se media >= 7 então  
14. escrever ("Aprovado")  
15. senão  
16. escrever ("Reprovado")  
17. fim se  
18. fim
```

**Linha 1:** Identificação do algoritmo.

**Linhas 2 e 3:** declaração das variantes, ou seja, notas 1,2 e 3.

**Linhas 4 a 18:** corpo do algoritmo: notas, cálculo da média, decisão e verificação.



---

## ENTENDENDO ALGUNS TERMOS

Você percebeu que foram utilizados alguns termos e símbolos específicos para representação dos algoritmos em pseudocódigos. Conheça melhor cada um deles:

**Algoritmo** - Comando que define o nome do programa. Deve vir entre aspas duplas.

**Var** - Comando que especifica a área em que as variáveis serão declaradas.

**Início** - Comando que informa o início do programa. É nesse bloco que ficarão os comandos e a lógica que será utilizada para criar o algoritmo.

**Escrever** - Comando que escreve na tela alguma informação ao usuário.

**Ler** - Comando que lê o que foi digitado pelo usuário.

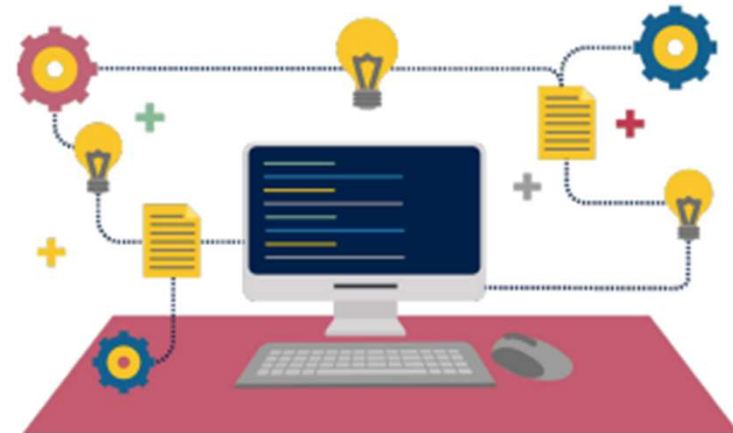
**Fim** - Comando que informa que é o final do algoritmo.

## TÉCNICAS DE PROGRAMAÇÃO

Agora que já sabemos o que é um algoritmo, precisamos conhecer algumas técnicas de programação.

Técnicas de programação são conceitos, recursos e boas práticas para o desenvolvimento de programas de computadores.

Com as técnicas de programação, como sintaxe e recursos específicos ou comuns entre as linguagens de programação, podemos criar as soluções para os nossos projetos, produtos e/ou serviços.

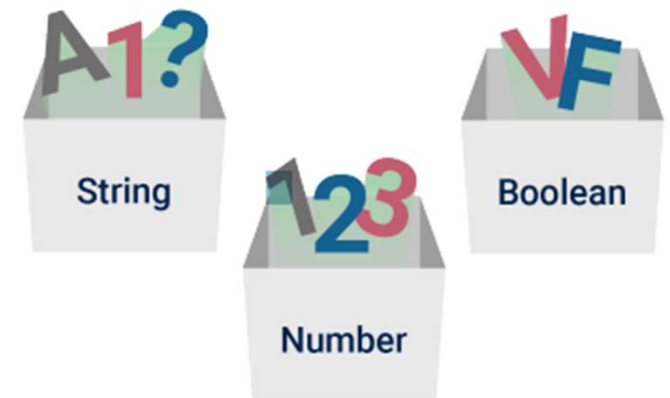


As técnicas de programação que abordaremos agora são:

- tipos e estrutura de dados;
- variáveis e constantes;
- fluxograma;
- estrutura de decisão;
- estrutura de repetição;
- operadores.

## TIPOS DE DADOS

Os dados são todas as informações relevantes para a programação que devem ser armazenadas, pois serão usadas durante a execução do algoritmo ou programa.



Os dados podem ser de vários tipos, como caracteres ou números. Isso varia de acordo com o sistema operacional e a linguagem de programação usada.

Vamos conhecer três tipos de dados comuns para exemplificar.

- ✓ **String** - O dado do tipo "**string**" armazena caracteres e é muito comum nas linguagens de programação. Porém, há diferenças: algumas linguagens de programação são case-sensitive, isto é, diferenciam caracteres maiúsculos de minúsculos, como a linguagem C, o JavaScript e o PHP. Outras são case insensitive, como Pascal e Delphi, que não diferenciam maiúsculas de minúsculas.

- ✓ **number** - Para armazenar números, o JavaScript, por exemplo, usa o dado do tipo "**number**" para números menores que 2<sup>53</sup> e do tipo "**BigInt**" para números maiores que 2<sup>53</sup>. No caso do PHP, o dado do tipo "**int**" armazena os números naturais e seus simétricos negativos, e o dado do tipo "**float**" é usado para números reais.
- ✓ **boolean** O dado do tipo "**boolean**" armazena apenas os valores **true** ou **false** (verdadeiro ou falso) e está presente em quase todas as linguagens de programação.

Os dados são representados pelas informações tratadas (processadas) por um computador. Essas informações estão caracterizadas basicamente por três tipos de dados: dados numéricos (inteiros e reais), dados caracteres e dados lógicos.

- ✓ **TIPOS INTEIROS** - São caracterizados como dados do tipo inteiro numérico positivo ou negativo, excluindo-se destes qualquer número fracionário. Exemplo: 65, 0, -13 e -76.
- ✓ **TIPOS REAIS** - São caracterizados como tipos reais os dados numéricos positivos, negativos e números fracionários. São exemplos: -44, 0, -3, -76, 1.2, -45.897, entre outros.

- ✓ **TIPOS LITERAIS** - São caracterizados como dados literais, as letras, números e símbolos especiais. Toda sequência de caracteres deve ser indicada entre aspas ("). Este tipo de dado é também conhecido como: alfanumérico, string, caracter ou cadeia.
- ✓ **TIPOS LÓGICOS** - São caracterizados como tipos lógicos os dados com valores verdadeiro e falso. É chamado de booleano, devido à contribuição do filósofo e matemático inglês George Boole na área da lógica matemática.

## ESTRUTURA DE DADOS

Estruturas de dados definem a organização e suas operações (métodos de acesso) sobre um determinado conjunto de informações. Essa organização permite um melhor processamento e é usada para grandes volumes de dados.

Por exemplo: você deseja armazenar o nome de todos os alunos da sua turma. O sistema deve definir a organização (como salvar) e, conseqüentemente, o acesso ao conjunto de operações sobre esses dados (inserir ou remover um aluno, editar um nome ou trocar a posição de um dado).

---

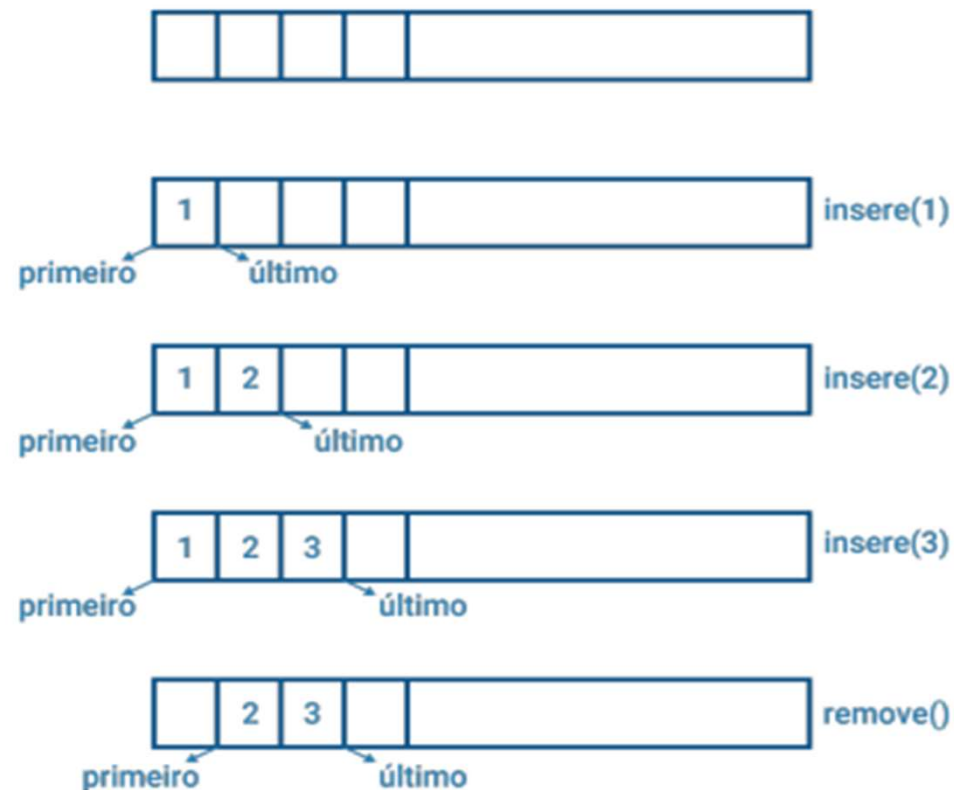
Há vários tipos de estrutura de dados.

**Lista (list)** – Uma lista armazena dados do mesmo tipo em sequência. Os dados não precisam estar necessariamente armazenados em sequência física na memória do computador, mas há uma sequência lógica entre eles. Cada elemento da lista é chamado de nó.

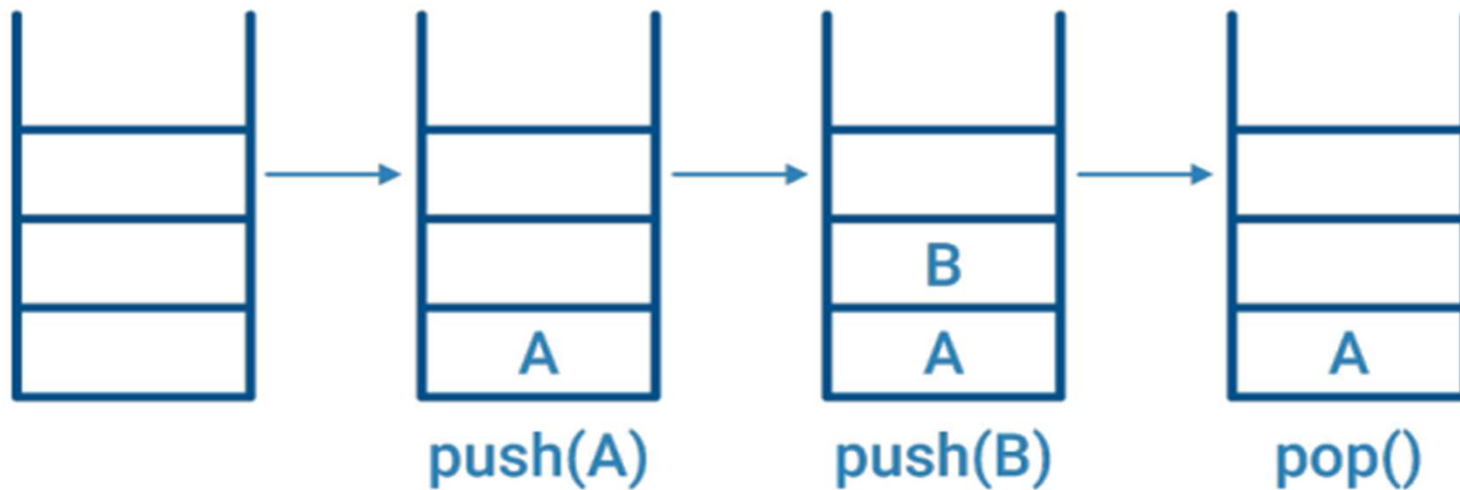
Em uma lista sequencial ou contígua, os nós estão em sequência lógica e física. Em uma lista encadeada, não há a necessidade da sequência física, apenas a sequência lógica deve ser mantida e o último nó é uma célula nula.

As listas são subdivididas em filas, pilhas e vetores, dependendo do acesso aos nós.

**Fila (queue)** - A fila é uma estrutura do tipo *first in first out* (FIFO), na qual o primeiro elemento a ser inserido será o primeiro a ser retirado. Assim como em uma fila de pessoas, o primeiro que chegou no balcão, será o primeiro a ser atendido, e assim por diante.



**Pilha (stacks)** - A pilha é uma estrutura do tipo *last in first out* (LIFO). O último elemento a ser inserido será o primeiro elemento a ser retirado. Podemos ter acesso ao topo dessa pilha, inserir um novo item, remover o item corrente. É como uma pilha de pratos: colocamos ou retiramos pratos no topo da pilha.



**Vetor (array ou matriz)** - Um vetor é uma estrutura que armazena uma sequência de objetos do mesmo tipo, em posição consecutiva lógica e física, sendo possível alcançar qualquer item diretamente. Não é necessário passar pelo primeiro ou último elemento.



## VARIÁVEIS E CONSTANTES

O valor dos tipos de dados é salvo em uma variável, ou seja, em um espaço de memória do computador.

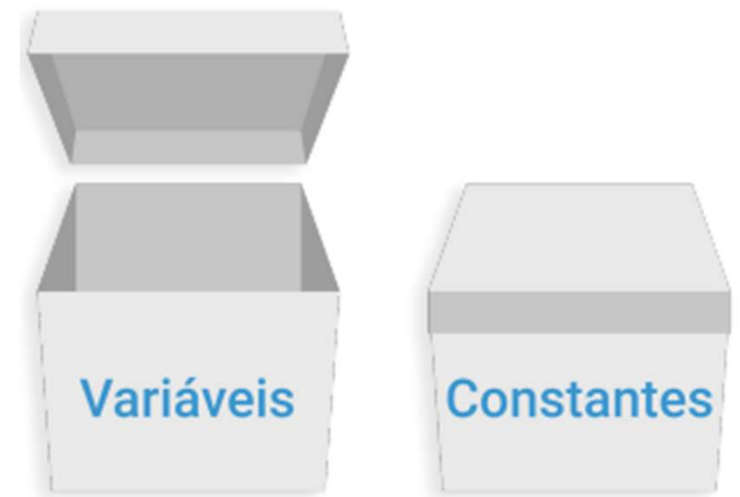
As variáveis devem ser declaradas com nome, valor e tipo.

O nome é atribuído ao declarar a variável.

O valor pode ser atribuído ou calculado.

O tipo depende do valor e da linguagem da programação usada.

A sintaxe dos comandos, inclusive o da declaração de variáveis, também depende da linguagem de programação usada.



Por exemplo: eu desejo armazenar o valor "Helena" em uma variável de nome **“nomeDoEstudante”**. Como o valor é formado de caracteres, o tipo da variável deve ser string. No quadro a seguir, temos dois exemplos de declaração de variáveis em duas linguagens de programação diferentes (em preto). O texto em cinza claro são comentários (indicados pelo sinal //) e não fazem parte do comando.

1.     // Linguagem X
2.     var nomeDoEstudante: String = "Helena";
3.     // var é a declaração de variável
4.     // nomeDoEstudante é o nome atribuído
5.     // string é o tipo de dado, pois é formado de caracteres
6.     // Helena é o valor
7.     //Linguagem Y
8.     \$nomedoestudante = 'Helena'
9.     // \$ é a declaração de variável
10.    // nomedoestudante é o nome atribuído
11.    // atribui o tipo de acordo com o valor
12.    // Helena é o valor

---

Durante a execução do algoritmo ou programa, o valor da variável pode mudar. Em um programa que calcula a idade atual de uma pessoa, por exemplo, a variável que guarda o valor da data atual mudará todos os dias.

**Caso o valor de uma variável não possa ser alterado, indicamos que ela é uma constante.** Como a data de nascimento de uma pessoa, no caso do programa que calcula a idade. Assim como a variável, a declaração da constante varia de acordo com a linguagem de programação usada.

✓ **O USO DE VARIÁVEIS** - É importante ressaltar que os dados a serem processados podem ser variáveis e todo dado armazenado na memória de um computador deve ser identificado, ou seja, é necessário saber qual o seu tipo para depois fazer o armazenamento adequado. Depois de armazenado, o dado poderá ser utilizado e manipulado.

Para exemplificar o conceito de variável, imagine que a memória do computador é um armário com várias gavetas, sendo que cada gaveta pode apenas armazenar um único valor (seja ele numérico, lógico ou caractere).

---

Desta forma, o valor armazenado pode ser utilizado a qualquer momento.

O nome de uma variável é utilizado para sua identificação durante a codificação do programa. Sendo assim, é necessário estabelecer algumas regras quanto ao uso das variáveis:

- Nomes de uma variável poderão ser atribuídos com um ou mais caracteres;
- O primeiro caractere do nome de uma variável não poderá ser um número e, sim, sempre uma letra;
- O nome de uma variável não poderá ter espaços em branco;
- O nome de uma variável não poderá ser uma palavra reservada (uma instrução ou comando utilizado no processo de escrita do código);
- Não poderão ser utilizados outros caracteres a não ser letras, números e sublinhado.

São nomes válidos de variáveis: NOMEDOUSUARIO, telefone, x,z, delta 27,71, entre outros. São nomes inválidos de variáveis: NOME DO USUARIO, 27 delta, telefonf, escreva e leia (são palavras reservadas), no caso do Portugal.

- 
- ✓ **INSTRUÇÕES BÁSICAS** - As instruções são representadas pelo conjunto de palavras-chaves de uma linguagem de programação e tem por finalidade comandar, usando um computador, o seu funcionamento e a forma como os dados armazenados devem ser tratados. 3.8
  - ✓ **REGRAS INICIAIS** - Você aprendeu o conceito e aplicação de uma variável, porém, é necessário ter alguns cuidados para diferenciar uma referência a uma instrução de uma variável.
    - Referências feitas a uma instrução devem ser escritas em letra minúscula em formato negrito;
    - Qualquer valor atribuído a uma variável será feito com o símbolo --, tanto no diagrama de blocos quanto em código português estruturado (Portugol).
  - ✓ **PORTUGOL (PORTUGUÊS ESTRUTURADO)** - Considere a seguinte situação-problema: “Criar um programa que realize a leitura de dois valores numéricos, faça a operação de adição entre os valores e apresente o resultado”.

---

Com base na situação proposta, a interpretação para aplicar a lógica estruturada é o primeiro e importante passo. Isto ocorre com a criação do algoritmo, o qual estabelece os passos necessários na busca de uma solução para a situação apresentada. Lembre-se que a criação de um algoritmo é como uma “receita”.

Para tanto, observe a estrutura do algoritmo com relação ao problema da leitura dos dois valores (A e B) e a sua soma (com base nos valores informados).

## APLICANDO ALGORITMOS

As informações que fornecemos ao computador são os dados de entrada da nossa aplicação, um processamento é feito com base na informação fornecida, e a saída é o resultado a partir do processamento.



Os valores dos dados de entrada e saída são armazenados em variáveis ou constantes. O processamento é feito através da aplicação ou programa de computador, baseado nos algoritmos da programação.

E a partir dessas informações armazenadas, podemos criar condicionais em nosso código de processamento para seguirmos ações distintas em nossa programação.

E, assim como na declaração de variáveis e constantes, a sintaxe para cada condicional depende da linguagem de programação escolhida. Apesar das especificidades de cada linguagem, a lógica de programação é a mesma, ou seja, o planejamento macro do algoritmo é o mesmo, mas a escrita dos comandos é diferente.

### **PENSE NISSO...**

Imagine que uma concessionária vai realizar um evento de test drive e o cadastro de participantes é feito através do site. Independentemente da linguagem de programação escolhida para o sistema de cadastro, todas devem atender ao requisito de só aceitar participantes com habilitação permanente de motorista e, conseqüentemente, maiores de 18 anos.

E se um menor de idade tentar fazer o cadastro? Como fazer essa validação? Vamos aprender?



---

## Algoritmo

1. Ler dois valores, representados pelas variáveis A e B;
2. Efetuar a soma das variáveis A e B e armazenar o resultado na variável X;
3. Apresentar o valor da variável X após a operação de soma dos valores fornecidos.

Observe que o algoritmo é um conjunto de passos que resulta numa solução para um determinado problema. Um detalhe a ser observado é que um algoritmo poderá ser feito de várias formas, não necessariamente como exposto acima, pois ele é a interpretação do problema. Acima está sendo utilizada a forma mais comum para iniciar a compreensão de um problema.

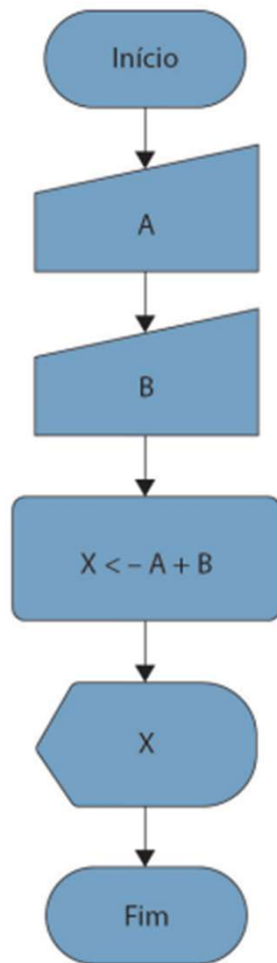


Figura 4 – Diagrama<sup>1</sup> de blocos  
Fonte: SENAI DR PR, 2018.

## Diagrama de blocos

Após a finalização da fase de interpretação do problema e da definição das variáveis, segue a fase de diagramação de blocos.

É importante destacar a indicação de Início e Fim do diagrama com o símbolo Terminal. Observe também a existência de uma seta que conecta um símbolo ao outro. Isto é necessário, pois desta forma sabe-se a direção que o processamento de um programa deverá prosseguir.

O símbolo retângulo significa Processamento e será utilizado para representar diversas operações, principalmente os cálculos matemáticos executados por um programa.

## Português Estruturado

A próxima fase é a codificação. Esta fase obedece ao que está definido no diagrama de blocos, pois é a representação gráfica da lógica de um programa. A codificação sempre deverá ser relacionada às variáveis que são utilizadas no programa. Este relacionamento, além de definir os tipos de dados que serão utilizados, define também o espaço de memória que será necessário para manipular as informações fornecidas durante a execução de um programa.

Desta forma, são utilizadas no exemplo três variáveis: A, B e X, sendo que deverão ser relacionadas antes de sua utilização, estabelecendo-se assim o seu respectivo tipo.

```
algoritmo "soma_numeros"
// Efetuar a soma de dois valores e mostrar o resultado
// Declaração de variáveis
var   X: inteiro   A: inteiro   B: inteiro
inicio
// Seção de Comandos
    leia(A)
    leia(B)
    X ← A + B
    escreva(X)
finalgoritmo
```

- 
- ✓ **TESTE DE MESA** - Para testar os algoritmos, existe uma técnica chamada teste de mesa que consiste em acompanhar passo a passo um algoritmo de forma a procurar falhas na lógica utilizada para resolver uma determinada situação-problema. A aplicação do teste de mesa mostra onde é necessário retificar a lógica.
  
  - ✓ **APLICAÇÃO DO TESTE DE MESA** - Com base na explicação acima, aplique o teste de mesa para calcular o salário a receber seguindo os seguintes itens:
    1. Informe o salário-base;
    2. Considere uma gratificação que é 5% do valor do salário-base;
    3. Considere um imposto de 3% a ser aplicado no valor do salário-base;
    4. O salário a receber é a soma do salário-base mais a gratificação, descontado o imposto.

```
Algoritmo calcular_Salario_Receber;  
var salarioBase, gratificacao, imposto, salarioReceber: real;  
  
início  
    escreva("Informe o salário-base: ");  
    leia(salarioBase);  
    gratificacao := salarioBase * 5 / 100;  
    imposto := salarioBase * 3 / 100;  
    salarioReceber := salarioBase + gratificacao - imposto;  
    escreva("O salário a receber é ", salarioReceber);  
fim
```

Para facilitar a verificação do algoritmo linha a linha será utilizada uma tabela com os valores de cada variável preenchida em um determinado passo do algoritmo. Considere o salário-base de R\$ 1.000,00.

Linha	salarioBase	gratificação	imposto	salárioReceber
1	-	-	-	-
2	1000	-	-	-
3	1000	50	-	-
4	1000	50	30	-
5	1000	50	30	1020
6	1000	50	30	1020

*Tabela 1 – Tabela de valores variáveis*

*Fonte: SENAI DR PR, 2018.*

Para um algoritmo é possível realizar vários testes de mesa informando várias entradas diferentes para saber se o algoritmo trabalha de forma consistente para quaisquer entradas.

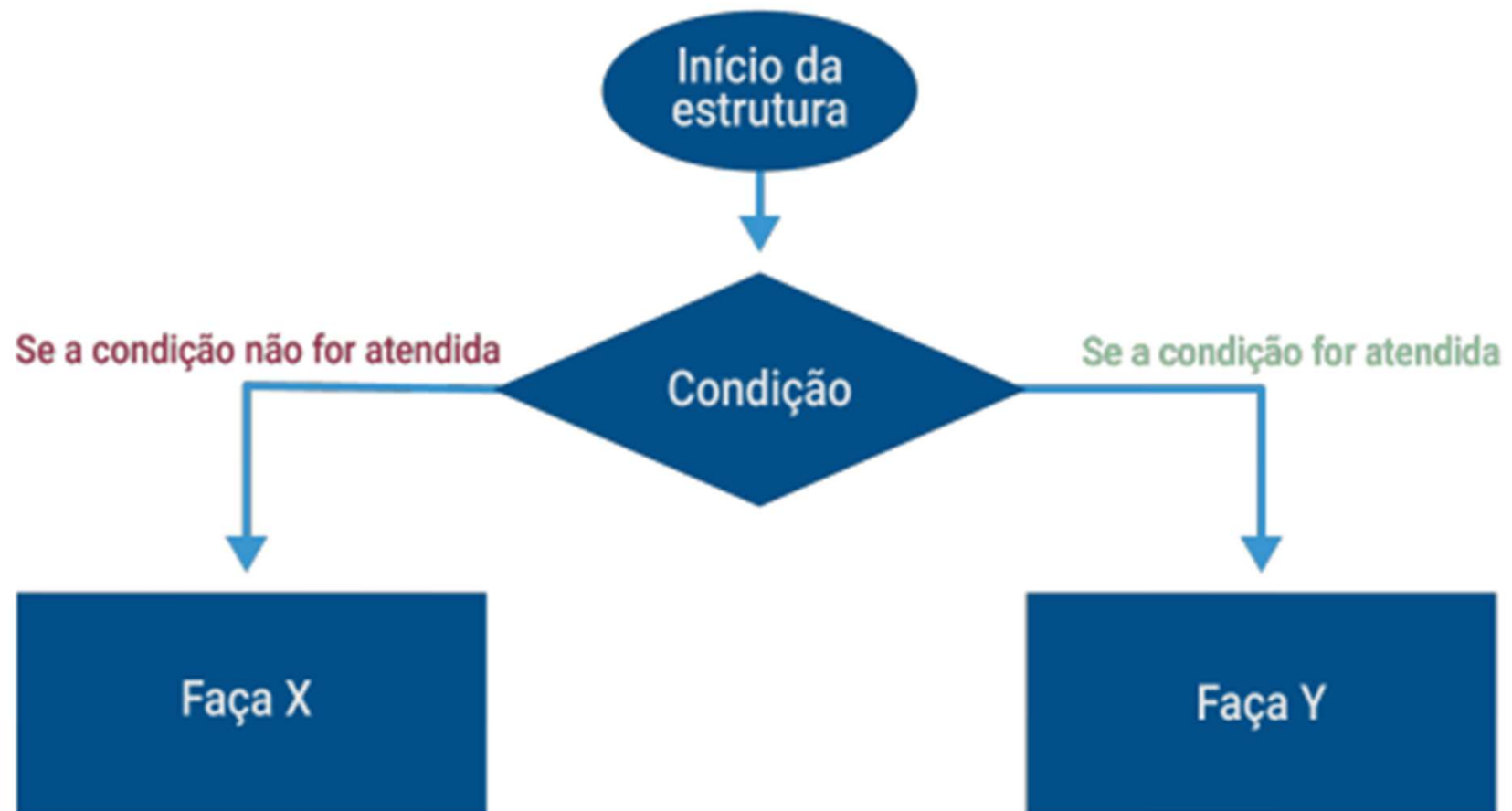
## ESTRUTURAS DE DECISÃO

As estruturas de decisão (também conhecidas como estruturas condicionais) usam comandos de decisão para testar uma ou mais condições e especificam uma instrução (ou um conjunto de instruções) se o resultado do teste for verdadeiro, e outra instrução (ou conjunto de instruções) caso o resultado do teste seja falso.



Em outras palavras, a estrutura de decisão é baseada em uma condição: se a condição for atendida, o programa segue um caminho e, se a condição não for atendida, o programa segue outro caminho.

O fluxograma é a representação gráfica de uma estrutura de decisão.





## COMANDOS DE DECISÃO

A grosso modo, um comando de decisão especifica uma condição a ser testada e indica uma instrução caso a condição seja atendida e outra instrução caso a condição não seja atendida. A sintaxe do comando varia de acordo com a linguagem de programação usada.

Vamos abordar os comandos mais comuns, que são:

- SE (if)
- SE/SE NÃO (if/else)
- SE aninhado (else if)
- CASO (switch)

### COMANDO SE (IF)

O mais simples dos comandos de decisão aparece nas linguagens de programação como if e pode ser traduzido como "SE". Note que aqui só há instrução caso a condição seja atendida, então não há ação específica e o restante do código é executado.

---

## **Código genérico**

if (condição) comando

Texto SE (senha correta) exibir "bem-vindo"

## **Linguagem X**

1. Var senha = a1234
2. if (senha = a1234) console. log "bem-vindo"
3. //console.log é um comando para exibir uma mensagem

---

## COMANDO SE/SE NÃO (IF/ELSE)

Aqui, o comando "SE" vem acompanhado de uma instrução caso a condição não seja atendida, como uma bifurcação no caminho.

### **Código genérico**

```
if (condição) comando  
else comando
```

### **Texto**

```
SE (senha correta) exibir "bem-vindo"  
SE NÃO exibir "senha incorreta"
```

### **Linguagem X**

1. Var senha = a1234
2. if (senha = a1234) console.log "bem-vindo"
3. else console.log "senha incorreta"

---

## COMANDO SE ANINHADO (IF/ELSE IF/ELSE)

E caso nenhuma das duas condições seja atendida, podemos utilizar o comando "SE" aninhado.

### **Código genérico**

```
if (condição) comando  
else if (condição) comando  
else comando
```

### **Texto**

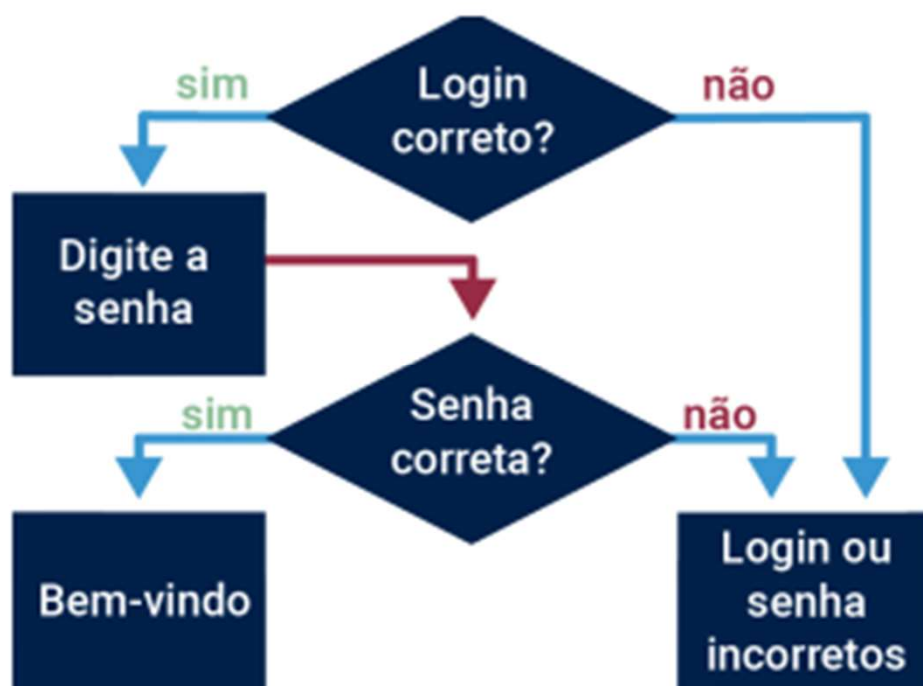
```
SE (login correto) exibir "digite a senha"  
E SE (senha correta) exibir "bem-vindo"  
SE NÃO exibir "login ou senha incorretos"
```

### **Linguagem X**

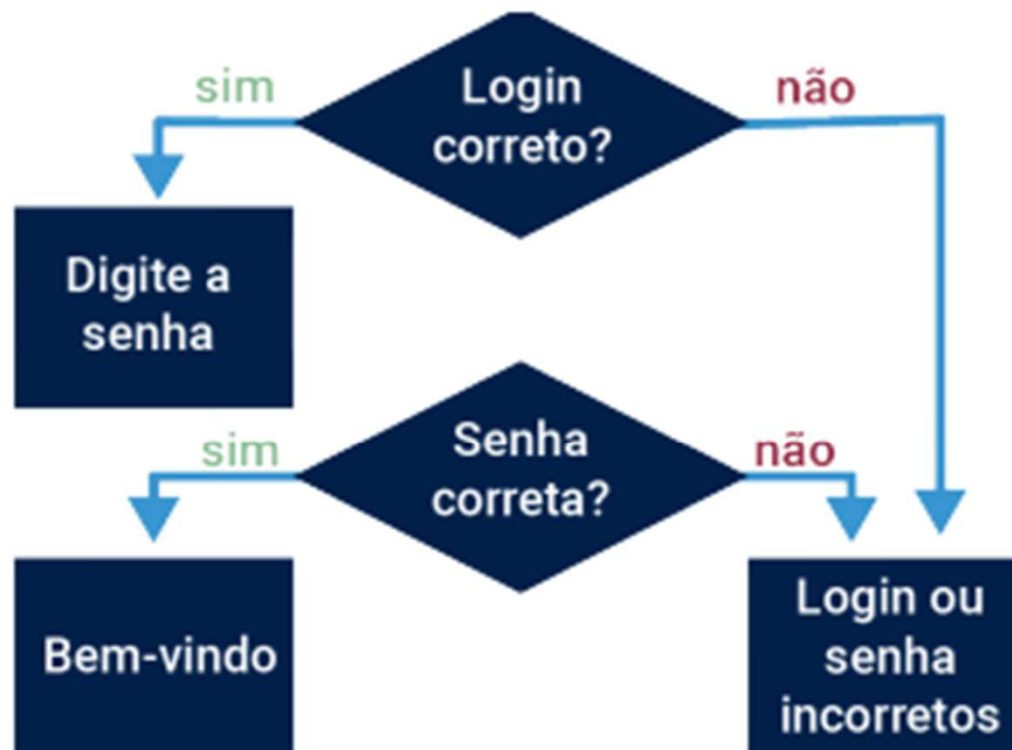
1. Var login = fulano
2. Var senha = a1234
3. 1f (login = fulano) console.log "digite a senha"
4. else 1f (senha = a1234) console. log "bem-vindo"
5. else console.log "login ou senha incorretos"

## Importante!

Note que o código só vai verificar o segundo *if* se o primeiro for atendido, ou seja, só vai verificar se a senha está correta, se o login estiver correto. Se o login estiver incorreto, o código executará o *else*.



Se a estrutura fosse criada com dois *if* e um *else*, cada condição iria gerar uma mensagem na tela, pois a segunda condição seria verificada de qualquer maneira.



## COMANDO CASO (SWITCH)

Este comando é muito utilizado quando uma quantidade maior de condições é necessária para ser utilizada em sua aplicação, quando há uma variedade maior na sua quantidade de alternativas necessárias.

### **Código Genérico**

```
switch (condição)
{
    caso 1:
        comando;
        pare;
    caso 2:
        comando;
        pare;
    caso 3:
        comando;
        pare;
    padrão:
        comando;
}
```

## Texto

verifique (fruta)

caso banana:

mostrar preço da banana;

pare;

caso maçã:

mostrar preço da maçã;

pare;

caso uva:

mostrar preço da uva;

pare;

padrão:

mostrar "Não temos esse produto";



## Linguagem X

```
1.  switch (fruta)
2.  {
3.      case banana:
4.          mostrar "Bananas custam R$10 a dúzia";
5.          break;
6.      case maçã:
7.          mostrar "Maças custam R$15 reais a dúzia";
8.          break;
9.      case uva:
10.         mostrar "Uvas custam R$20 o kilo";
11.         break;
12.     default:
13.         mostrar "Não temos essa fruta";
14. }
```

## Importante!



A instrução opcional *break* associada a cada *case* garante que o programa saia da condicional *switch* e execute a instrução que segue logo após o *switch*. Caso *break* seja omitido, o programa continua a execução para a próxima instrução dentro de *switch*.

No exemplo, "fruta" é avaliado como "Bananas", o programa corresponde o valor com o *case* "Bananas" e executa a instrução associada. Quando *break* for encontrado, o programa para (break), saindo de *switch* e executa a instrução localizada após o *switch*. Se *break* fosse omitido, a instrução para "Maçãs" também seria executada.