Programas, Aplicativos ou Softwares

Vamos pensar um pouco sobre o que costumamos chamar de programas, aplicativos ou softwares. Você sabe o que são os programas? Na realidade, programas são algoritmos que foram codificados em uma determinada linguagem de programação. As linguagens de programação vêm mudando bastante com o tempo. No entanto, a forma de pensar em soluções computadorizadas e de fazer projetos de programas quase não sofreu alterações ao longo dos anos. Você consegue perceber agora a importância de aprender a pensar de forma lógica e a elaborar, interpretar e corrigir algoritmos? Existem várias linguagens sendo utilizadas no mercado atual, como C, C#, C++, Java, JavaScript, PHP, Phyton, Objective-C, Ruby, VB, entre outras. Além disso, novas linguagens surgem de tempos em tempos, fruto do constante desenvolvimento tecnológico, sempre em busca de soluções mais inteligentes e eficientes em

programação. Dessa forma, toda pessoa que pretende atuar na área de programação precisa estar preparada para aprender as linguagens mais modernas que forem surgindo. Nessa área, a atualização é uma constante. Sabemos o quanto um computador é importante, atualmente em nossas vidas, certo? No entanto, o computador só é capaz de facilitar nossas atividades e resolver nossos problemas se for programado. Nesse sentido, existem programas específicos, denominados APLICATIVOS, que buscam resolver um problema específico a partir de dados informados pelos usuários. Cada aplicativo (programa específico) tem sua função!

Nesse sentido, programar é prever situações e tomar decisões sobre como alcançar os resultados esperados. Não podemos esquecer que sempre existe mais de uma forma de resolver um problema. E como podemos prever as situações que um programa precisará resolver? Vamos adiante para descobrir! Hoje em dia, o bom profissional de informática é valorizado por várias competências. No entanto, a capacidade de raciocinar de forma lógica é fundamental para ter reconhecimento no mercado de trabalho. Na prática, isso significa que a Lógica é o pilar de sustentação do profissional da área da Informática.

Essa sequência de frases curtas, com comandos claros e encadeados em ordem lógica, é a forma mais simples de criar algoritmos. Esse jeito de elaborar algoritmos se chama descrição narrativa e são muito usadas em documentos como manuais de equipamentos, receitas culinárias, bulas de remédio, descrições do tipo faça você mesmo etc.

Regras para o Desenvolvimento de Algoritmos

1 - Seja direto: Escreva frases curtas e simples, com apenas um verbo. 2 - Seja detalhista: Descreva, com detalhes, todos os passos necessários. 3 - Seja lógico: Encadeie as ações na ordem em que elas devem acontecer. 4 - Seja objetivo: Procure atingir o objetivo proposto com o menor número de passos, tomando o cuidado de não pular passos importantes. 5 - Seja claro: Evite o uso de palavras e expressões com duplo sentido (termo conhecido tecnicamente como ambiguidade). 6 - Seja compreensível: Redija instruções de forma que qualquer pessoa possa entendê-las, não somente profissionais da área de Informática. 7- Seja crítico: Faça a leitura crítica de seu algoritmo para melhorar a escrita e o encadeamento lógico dele.

Depois de lermos essas orientações, que tal voltarmos ao caso do algoritmo para sacar dinheiro em um caixa eletrônico e vermos esses princípios na prática?

Conceitos Básicos sobre Programação Orientada a Objeto

Na atualidade, programação de softwares, trata como uma frequente evolução, questões relacionadas à segurança e ao reaproveitamento de código, o que é importante no desenvolvimento de softwares, ainda mais quando se visa resultados rápidos e eficientes. A Programação Orientada a Objetos (POO) centra-se em programas estruturados em módulos, os objetos, que devem estar relacionados e agir de forma organizada, sendo essa a questão principal que contribui para reutilização de código, e para entender a POO é essencial entender, pelo menos cinco conceitos básicos: abstração, classe, objeto, encapsulamento, herança e polimorfismo. E, além disso, saber como eles atuam na POO e qual a importância de cada um deles. O padrão da programação orientada a objetos é seguido por

muitas linguagens de programação, por exemplo, o Java, inteiramente baseado nesse padrão e, por isso, utilizado no ensino de programação orientada a objetos, padrão que será estudado no decorrer do aprendizado em lógica, e na programação estrutural. Para obter resultados com a programação orientada a objetos, é necessário abstrair pensamentos e conceitos da realidade vivenciada e tentar aplicá-los no programa, para abranger o que o objeto deve possuir e fazer funcionar dentro do sistema a ser desenvolvido.

Objeto

Os objetos são características definidas pelas classes. Neles é permitido instanciar objetos da classe para inicializar os atributos e invocar os métodos. Veja no exemplo da Figura 1.

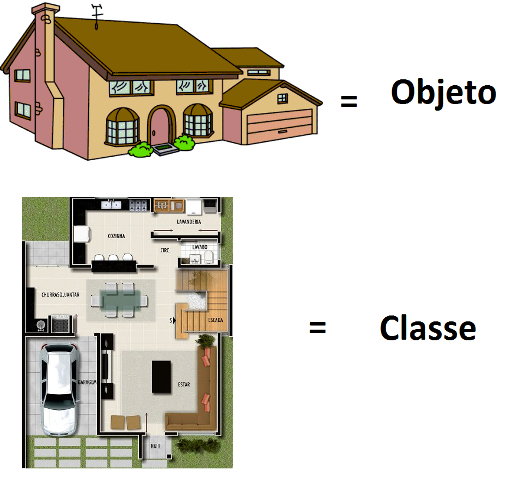


Figura 1: Diferença entre objeto e classe

A Figura 1 mostra que todo objeto é algo que existe, uma coisa concreta, já a classe é considerada como um modelo ou projeto de um objeto, sendo algo que não consegue tocar.

## Atributos

Os atributos são as propriedades de um objeto, também conhecidos como variáveis ou campos. Essas propriedades definem o estado de um objeto, fazendo com que esses valores possam sofrer alterações. O código 1 mostra as características de um cachorro, em linguagem Java, mas os valores que são guardados nas variáveis são diferentes variando para cada cachorro.

**Código 1:** Classe Cachorro

|  |
| --- |
| public class Cachorro {      public String nome;      public int peso;      public String corPelo;      public String raca;  } |

## Métodos

Os métodos são ações ou procedimentos, onde podem interagir e se comunicarem com outros objetos. A execução dessas ações se dá através de mensagens, tendo como função o envio de uma solicitação ao objeto para que seja efetuada a rotina desejada.

Como boas práticas, é indicado sempre usar o nome dos métodos declarados como verbos, para que quando for efetuada alguma manutenção seja de fácil entendimento. Veja algumas nomenclaturas de nomes de métodos:

* acaoVoltar;voltar;
* avançar;
* correr;
* resgatarValor;
* pesquisarNomes;

No Código 5 são declarados as características e o método, nota-se que tem uma condição de acordo com o valor informado na variável “tamanho”.

**Código 5:** Classe Cachorro com método

|  |
| --- |
| class Cachorro{      int tamanho;      String nome;        void latir(){          if(tamanho > 60)              System.out.println("Wooof, Wooof!");          else if(tamanho > 14)              System.out.println("Ruff!, Ruff!");          else              System.out.println("Yip!, Yip!");      }  } |

Com base nas definições dos conceitos atributos e métodos, caracterizamos que a classe Cachorro deve ter:

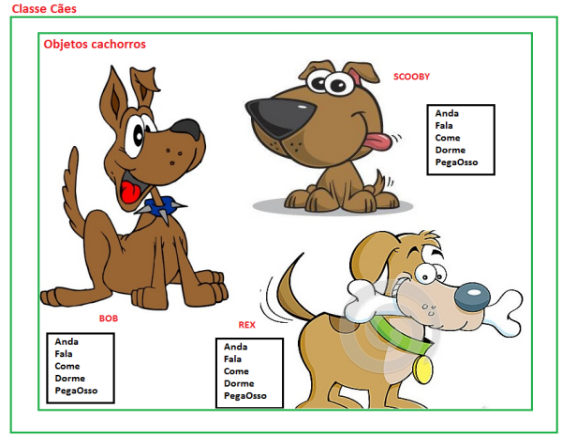
Atributos: Nome, Cor do pelo, Raça, Peso, Tamanho e Idade.

Métodos: Latir, Comer, Correr, Pular e Cavar.

## Classes

**As classes de programação são projetos de um objeto, aonde têm características e comportamentos,** ou seja, permite armazenar propriedades e métodos dentro dela. Para construir uma classe é preciso utilizar o pilar da abstração. Uma classe geralmente representa um substantivo, por exemplo: uma pessoa, um lugar, algo que seja “abstrato”.

### - Características de Classes

* Toda classe possui um nome;
* Possuem visibilidade, exemplo: public, private, protected;
* Possuem membros como: Características e Ações;
* Para criar uma classe basta declarar a visibilidade + digitar a palavra reservada class + NomeDaClasse + abrir e fechar chaves { }.
* 
* **Figura 2:** Demonstração da classe Cães
* No código 2 e na Figura 2, mostra que a classe Cães de um modo genérico, tem os mesmos métodos independente de qualquer cachorro, sendo que a classe é sempre um molde/projeto para o objeto cachorro.

Formas de Representação de Algoritmos

Os algoritmos podem ser representados de várias formas, como por exemplo:

1. Através de uma língua (português, inglês, etc.): forma utilizada nos manuais de instruções, nas receitas culinárias, bulas de medicamentos, etc.
2. Através de uma linguagem de programação (Pascal, C, Delphi, Java.): esta forma é utilizada por alguns programadores experientes, que "pulam" a etapa do projeto do programa (algoritmo) e passam direto para a programação em si.
3. Através de representações gráficas: são bastante recomendáveis, já que um "desenho" (diagrama, fluxograma, etc.) muitas vezes substitui, com vantagem, várias palavras.

Cada uma dessas formas de representar um algoritmo tem suas vantagens e desvantagens, cabe à pessoa escolher a forma que melhor lhe convir. Nesta disciplina serão apresentadas três formas de representação de algoritmos (que são algumas das mais utilizadas), são elas:

- Diagrama de Nassi-Shneiderman (Diagrama de Chapin);

- Fluxograma (Diagrama de Fluxo);

- Português Estruturado (Pseudocódigo, Portugol ou Pseudolinguagem).

Não existe consenso entre os especialistas sobre qual é a melhor maneira de representar um algoritmo.

As linguagens de programação tratam os dados de um computador através do uso de algoritmos. Um algoritmo é uma estruturação passo-a-passo de como um determinado problema deve ser resolvido de forma não ambígua (ou como muitos comparam "uma receita de bolo"). Desta forma, para realizar esta estruturação é necessário o uso de ferramentas e operações oriundas da Lógica, e principalmente da Lógica Matemática.

A elaboração de um algoritmo para criação de um programa requer algumas etapas:

1. Definir o problema;
2. Analisar a situação e interpretar a necessidade de se obter uma forma de resolver o problema;
3. Após a implementação, analisar junto ao usuário se o problema foi resolvido.

O problema que o algoritmo representa é composto por três fases:

PROCESSAMENTO

ENTRADAADA

SAÍDAADA

Onde:

ENTRADA: Dados de inserção de informação;

PROCESSAMENTO: Procedimentos utilizados para chegar ao possível resultado;

SAÍDA: Dados processados e o resultado final.

Em baixo podemos ver um algoritmo implementado num fluxograma, sobre o estado de uma lâmpada:

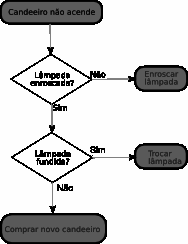


Figura 1 - Algoritmo num fluxograma

## Padrão de Utilização das Principais Formas Geométricas

Vamos conhecer os principais símbolos usados na construção de fluxogramas?

Preste bastante atenção! Cada elemento tem função e significado diferentes.

1. TERMINAL

Terminal. Indica o ponto de início ou término de uma sequência lógica. Deve ser utilizado com a palavra início ou fim escrita dentro da figura.

1. VERIFICAÇÃO OU DECISÃO

Verificação Condição ou decisão. Indica o ponto de teste (verdadeiro ou falso) ou de tomada de decisão (sim ou não). Conta com uma entrada e permite indicar duas saídas diferentes, sendo uma para cada situação/resposta. Exemplo: Está chovendo?

Saída 1 NÃO

Saída 2 SIM

Ver televisão em casa; Passear no parque



1. SETA DE FLUXO

Seta de fluxo. Indica o sentido do fluxo de informações e instruções. Deve ser utilizada para conectar todos os símbolos do diagrama.

1. AÇÃO OU PROCESSAMENTO

Ação ou processamento. Indica uma ação ou um processamento de qualquer tipo. Exemplos: Calcular o preço final do produto. Jogar a embalagem no lixo.

1. ENTRADA DE DADOS

Entrada/saída de dados Símbolo genérico que indica a entrada ou a saída de dados, respectivamente uma leitura/captura de dados ou uma gravação/apresentação da informação após o processamento, independentemente da forma como essas informações serão inseridas ou disponibilizadas aos usuários.

* 1. ENTRADA DE DADOS (via cartão)

Entrada de dados (via cartão). Indica a entrada de dados, através da leitura em cartão perfurado. Este símbolo está em desuso atualmente.

* 1. ENTRADA MANUAL (via teclado)

Entrada manual (via teclado). Indica a entrada de dados feita manualmente pelo usuário, isto é, por meio de digitação. Exemplos: Digitar o preço do produto. Entrar/Digitar a nota da avaliação.

* 1. SAÍDA DE DADOS (via monitor)

Saída de dados (via monitor). Indica que algum resultado, dado ou informação será exibido na tela do dispositivo ao usuário. Exemplos: Mostrar/Exibir/Apresentar a média do aluno. Exibir o valor total do produto.

* 1. SAÍDA DE DADOS (via impressora)

Saída de dados (via impressora). Indica resultados ou informações que serão disponibilizadas ao usuário por meio de impressão – recibos, relatórios, documentos etc. Exemplos: Imprimir relatório de compra ou venda. Imprimir recibo de entrega de documento.

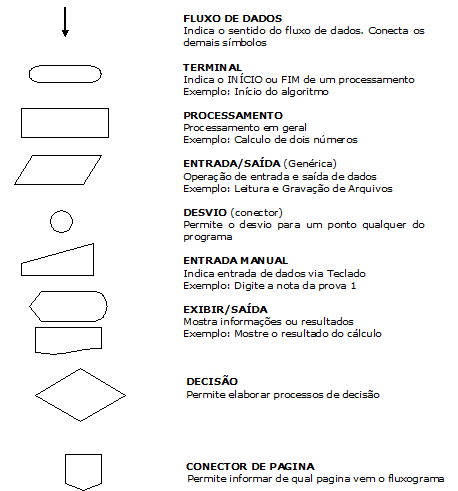
1. CONECTOR

Conector. Indica o ponto a partir do qual o algoritmo ou processa mento irá continuar em outro lugar, repetindo, também, este mesmo símbolo. Deve conter um mesmo número ou uma mesma letra em seu interior, indicando a saída com a sua respectiva entrada no fluxo. Funciona como um conector, que tem por objetivo dar continuidade a duas partes de um mesmo processo que se encontram escritas em locais separados. IMPORTANTE: Se necessário, podemos definir vários conectores de saída no fluxo, mas APENAS UM conector de entrada.

Exemplo 1

Suponha que você precise criar um algoritmo para comer um bombom.

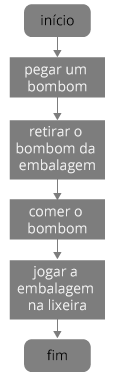
Como esse algoritmo seria apresentado pela descrição narrativa e pelo fluxograma?



#### Descrição Narrativa – Algoritmo para comer um bombom

1. **Pegar** um bombom.
2. **Retirar** o bombom da embalagem.
3. **Comer** o bombom.
4. **Jogar** a embalagem na lixeira.

Fluxograma – Algoritmo para comer um bombom

×

**Simbologia do Diagrama de Chapin**

O Diagrama de Chapin, do criador Ned Chapin, é a substituição do fluxograma tradicional por diagrama que apresenta uma visão hierárquica e estruturada da lógica do programa. Sua maior vantagem é a representação das estruturas que tem um ponto de entrada e um ponto de saída e são compostas pelas estruturas básicas de controle de sequência, seleção e repartição.

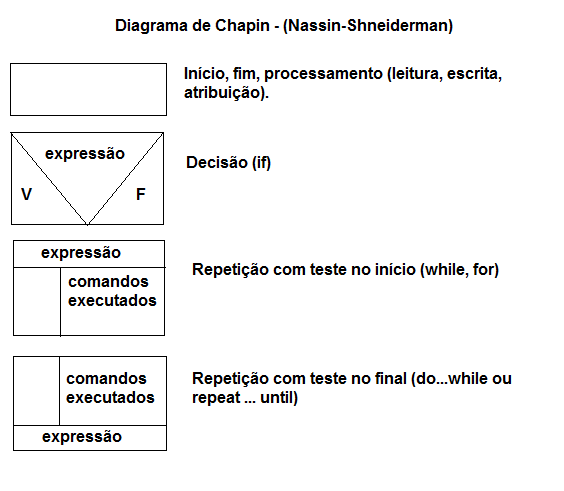
Vantagens e Desvantagens nas formas de representação gráfica (Fluxograma e Chapin)

Vantagens:

* Maior clareza no fluxo de execução;
* Linguagem visualização.

Desvantagens:

* Requer conhecimento de convenções gráficas;
* Mais trabalho em decorrência de seus desenhos;
* Dificuldade para fazer correções.



# Etapas de Desenvolvimento de um Software

Estudo do problema, Análise do problema e de seu contexto, para identificar a situação, refletir sobre suas características e apresentar as possíveis soluções

Estruturação do algoritmo Escolha da melhor solução possível para resolver o problema e estruturação da proposta em forma de “projeto de programa” (algoritmo).

Desenvolvimento do código (programação)

Escrita do algoritmo a partir das regras e dos padrões de uma linguagem de programação específica, tal como C#, Java, C, C++, VB, PHP. É nesta etapa que ocorre a codificação do algoritmo. Implantação da solução Implantação do programa em situação de vida real, para verificar se o problema foi resolvido, e realização de eventuais ajustes (manutenção). Agora, vejamos como utilizar o português estruturado na prática! Para isso, vamos voltar ao algoritmo de conversão de real em dólar. Nesse caso, primeiro relembraremos o algoritmo usando a descrição narrativa e, em seguida, veremos como fica o algoritmo usando o pseudocódigo. Acompanhe a construção do algoritmo em cada caso.

**Descrição Narrativa**

Algoritmo para converter real em dólar:

1. **Informar** o valor disponível em reais para comprar dólares.
2. **Informar** a cotação do dólar do dia.
3. **Dividir** o valor disponível pela cotação do dia.
4. **Exibir** o resultado do cálculo para o usuário.



**Pseudocódigo**

**Algoritmo** Conversao\_real\_dolar

// rotina que converte um determinado valor em reais em dólares

**Var**

VALOR\_REAL, COTACAO\_DOLAR, CONVERSAO: real

VALOR\_REAL = 0

COTACAO\_DOLAR = 0

CONVERSAO = 0

**Início**

**Escreva** “Programa para converter reais em dólares”

**Escreva** “Informe o valor disponível em reais (para comprar dólares): R$”

**Leia** VALOR\_REAL

**Escreva** “Informe o valor do dólar em reais (cotação do dia): R$”

**Leia** COTACAO\_DOLAR

CONVERSAO = VALOR\_REAL / COTACAO\_DOLAR

**Escreva** “Com essa quantia será possível comprar: US$”

**Escreva** CONVERSAO

**Escreva** “Boa viagem!”

**Fim.**

**Descrição Narrativa versus Pseudocódigo**

Algoritmo para converter real em dólar:

1. **Informar** o valor disponível em reais para comprar dólares.
2. **Informar** a cotação do dólar do dia.
3. **Dividir** o valor disponível pela cotação do dia.
4. **Exibir** o resultado do cálculo para o usuário.

**Algoritmo** Conversao\_real\_dolar

// rotina que converte um determinado valor em reais em dólares

**Var**

VALOR\_REAL, COTACAO\_DOLAR, CONVERSAO: real

VALOR\_REAL = 0

COTACAO\_DOLAR = 0

CONVERSAO = 0

**Início**

**Escreva** “Programa para converter reais em dólares”

**Escreva** “Informe o valor disponível em reais (para comprar dólares): R$”

**Leia** VALOR\_REAL

**Escreva** “Informe o valor do dólar em reais (cotação do dia): R$”

**Leia** COTACAO\_DOLAR

CONVERSAO = VALOR\_REAL / COTACAO\_DOLAR

**Escreva** “Com essa quantia será possível comprar: US$”

**Escreva** CONVERSAO

**Escreva** “Boa viagem!”

**Fim.**

As palavras em destaque no texto a seguir para interagir com o pseudocódigo.

Tal como você pôde perceber, o pseudocódigo é uma técnica que organiza os algoritmos em blocos formados por comandos escritos. Para isso, são usados, por exemplo, os **comandos** **Leia** e **Escreva** para inserção (captura) e exibição (apresentação) de dados, respectivamente. Além dos comandos, o pseudocódigo permite a declaração de **variáveis** e **expressões aritméticas** para realizar contas.

**Algoritmo** Conversao\_real\_dolar

// rotina que converte um determinado valor em reais em dólares

**Var**

VALOR\_REAL, COTACAO\_DOLAR, CONVERSAO: real

VALOR\_REAL = 0

COTACAO\_DOLAR = 0

CONVERSAO = 0

**Início**

**Escreva** “Programa para converter reais em dólares”

**Escreva** “Informe o valor disponível em reais (para comprar dólares): R$”

**Leia** VALOR\_REAL

**Escreva** “Informe o valor do dólar em reais (cotação do dia): R$”

**Leia** COTACAO\_DOLAR

CONVERSAO = VALOR\_REAL / COTACAO\_DOLAR

**Escreva** “Com essa quantia será possível comprar: US$”

**Escreva** CONVERSAO

**Escreva** "Boa viagem!"

**Fim.**

Você notou como o pseudocódigo é uma forma bastante detalhada de **escrever sequências lógicas de instruções e processamentos**?

Nesse sentido, a principal característica desse método é ser muito semelhante ao jeito como as linguagens de programação são codificadas. E foi justamente o fato de ser muito parecido com os códigos de programação que fez com que surgisse o nome pseudocódigo, ou seja, “falso” (fictício) código. É importante que você saiba que o pseudocódigo tem grande aceitação entre os programadores. Mas por que será que isso ocorre?

O principal motivo está no fato de o pseudocódigo possibilitar uma tradução praticamente direta para qualquer linguagem de programação específica. Justamente por isso, iremos apresentar a você vários exemplos de algoritmos em pseudocódigo daqui em diante.

**Como programador da área de Informática, você deverá ser capaz de escrever, interpretar, testar, localizar e resolver erros de lógica tanto em fluxogramas como em pseudocódigos. Por isso, mãos ao *mouse* (ou ao teclado)!**

Apresentaremos, a seguir, a forma básica de um algoritmo em português estruturado:

**Algoritmo** <nome\_do\_algoritmo>

// <explicação resumida do que a rotina irá fazer>

**Var**

<declaração das variáveis e do tipo de informação que elas receberão>

**Início**

<instruções e processamentos encadeados de forma lógica>

**Fim.**

Todo algoritmo começa com a definição de um nome de identificação para ele, na primeira linha: <nome\_do\_algoritmo>

Na linha de baixo, deve ser feita a declaração de variáveis, cuja palavra reservada é **Var**. O termo **Var** é uma redução da palavra "variável". Nesse sentido, o termo **Var** é um marcador, assim como as palavras INÍCIO, FIM, SE, SENÃO, ENTÃO, FAÇA CASO etc. Um marcador ou uma palavra reservada são termos interpretados de forma diferenciada pelos programas. Por isso, não devem ter outros usos, como nomear variáveis.

Agora ficou mais fácil identificar um algoritmo estruturado em pseudocódigo, não é mesmo?

Alguns recursos disponíveis na internet podem ser úteis para que você desenvolva a habilidade de construir, interpretar e corrigir pseudocódigos. A seguir, indicamos dois aplicativos que permitem a construção e o teste de algoritmos escritos em pseudocódigo. Eles são gratuitos e fáceis de usar. Nas imagens das logomarcas saberá um pouco mais sobre os recursos indicados e nos *hiperlinks* para conhecer os *sites*.

# Tipos de Dados

Programar computadores requer a definição e o uso de espaços em memória para receber informações inseridas pelos usuários. Os espaços em memória recebem informações que podem variar no decorrer de um processamento. Por isso, é muito importante aprender a criar e definir o tipo, bem como manipular variáveis e constantes.

**Em linhas gerais, variáveis são espaços em memória que servem para guardar um tipo de dado ou informação. Esses espaços em memória devem receber nomes específicos e únicos, para que possam ser referenciados exclusivamente e usados sempre que necessário.**

Ao criar um espaço em memória, com conteúdo variável ou constante, precisamos definir e registrar o tipo de dado que será inserido e manipulado.

Verifique nos itens a seguir para conhecer os três tipos de dados!

# Literais

Dados literais – ou caracteres – são sequências contendo letras, números e outros símbolos especiais. Uma sequência de caracteres deve ser indicada entre aspas (“”). Esse tipo de dado é conhecido também como Alfanumérico, String, Literal ou Cadeia. Como exemplos, temos: “Fundação Bradesco”, “Técnico em Desenvolvimento de Sistemas”, “84”, “843.48”, entre outros numéricos.

# Numéricos Reais

Os dados numéricos reais são números positivos, negativos e fracionários. Como exemplo, temos: 584.87, -848.43, 84, 82, -19.20. Lembrando que, na formatação universal, a parte fracionária é separada por ponto “.” e não por vírgula “,”, como fazemos na formatação brasileira.

Numéricos Inteiros

Os dados numéricos inteiros são definidos como tipos inteiros. Podem ser dados numéricos positivos ou negativos. Nesse tipo, não se encaixam números fracionários. Como exemplo, temos: 10, -10, 5, 85, -33, 88, -67, entre outros. Os dados lógicos são apenas dois valores utilizados por elementos condicionais para tomadas de decisão. Algumas bibliografias escrevem os valores lógicos entre pontos, para dar mais ênfase e diferenciá-los de outros conceitos ou definições (como variáveis ou valores literais). Por exemplo: .V. ou .F. Esses dois valores podem ser: Verdadeiro ou Falso, ou .V. ou .F. Sim ou Não 0 ou 1. São também conhecidos como dados booleanos, por referência a George Boole, matemático que deu nome à álgebra booleana, expressão que trata desses tipos de dado.

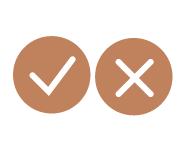
# Uso e Funções de Constantes e Variáveis

Já mencionamos algumas vezes as palavras "constante" e "variável". Você deve estar curioso para saber melhor o que elas significam, não é mesmo? Agora, vamos definir cada uma delas e dar exemplos de suas funções.

**Chamamos de constante todo valor fixo que não será alterado durante a execução de um programa.**

Conforme o contexto de utilização, as constantes podem ser classificadas como:

**Numéricas**  **Literais**

**Lógicas**

Vejamos um exemplo de uso de **CONSTANTE**:

***CONSTANTES***

***CONSTANTES***

***2***

***IMC= peso/altura***

Observe que, na fórmula para calcular o IMC (Índice de Massa Corporal), o número dois é um valor constante, pois não muda. Desse modo, o número dois é diferente dos demais elementos (IMC, peso e altura), que vão variar em cada caso. Em relação ao tipo do dado, o número dois deve ser classificado como uma constante do tipo numérica inteira.

Ficou claro o conceito de constante agora?



Agora, vamos entender melhor as **VARIÁVEIS**.

Uma variável representa um endereço da memória RAM que armazena, temporariamente, valores e informações. Esses endereços recebem um nome e um conteúdo. Cada vez que mencionamos o nome da variável, seu conteúdo é manipulado. Vamos retomar o exemplo da fórmula para calcular o IMC (Índice de Massa Corporal)? Observe a seguir:

IMC = Peso

***VARIÁVEL***

Altura

Simples, não?

É importante sabermos que as variáveis podem armazenar todos os tipos de dados. Além disso, para criar variáveis ou constantes, precisamos definir um nome simbólico, uma identificação a partir da qual elas serão chamadas, o tipo de dado que elas receberão e um valor inicial. A criação de variáveis ocorre no momento de codificação do programa. Desse modo, precisamos atribuir um nome simbólico a determinado endereço da memória RAM. No decorrer do programa, esse nome vai ser utilizado para manipular a informação contida no endereço da memória relacionado à variável. Para nomear uma variável, precisamos seguir algumas regras. Você sabe quais são essas regras? Vamos descobrir!

Os nomes das variáveis:

* Devem sempre começar por uma letra.
* Não devem ter espaços em branco.
* Não devem ter caracteres especiais – no máximo, o símbolo sublinhado.

Por exemplo:

Nome\_cliente

Num\_funcionario

Datanascimento

Além disso, precisamos saber que as variáveis devem ser declaradas sempre no início dos algoritmos e dos programas. Veremos um exemplo mais adiante.

Você sabe até quando os valores permanecem armazenados nas variáveis?

Os valores armazenados nas variáveis permanecem:

* Até que o computador seja desligado pelo usuário ou por falta de energia, já que são armazenados na memória RAM.
* Por término do programa ou da rotina onde foram criados.
* Até que seja atribuído um novo valor para a mesma variável.

Vejamos um exemplo:

Se escrevermos **A = 5**, o valor *cinco* será o conteúdo da variável **A**.

Se, mais adiante, escrevermos **A = 20**, o valor *vinte* passará a ser o novo conteúdo da variável **A**.

Nesse caso, o valor anterior **5** (cinco) será eliminado (substituído).

Depois de tudo o que vimos até aqui, está na hora de interpretarmos um algoritmo mais complexo, escrito em portugol (pseudocódigo). Aceita o desafio?!

Para isso, vamos retomar o fluxograma para sacar dinheiro em um caixa eletrônico. Agora, veja como essa sequência poderia ficar, com alguns ajustes, em portugol. Veja a sequência lógica representada de três formas diferentes:

* Descrição narrativa
* Fluxograma
* Pseudocódigo.

Descrição narrativa *versus* fluxograma *versus* pseudo

#### Portugol

**Algoritmo** Saque\_dinheiro\_caixa\_eletronico

// rotina para sacar dinheiro em um caixa eletrônico

**Var**

VALOR\_SAQUE, SALDO: real

SENHA\_SAQUE, NUM\_AGENCIA, NUN\_CONTA: inteiro

VALOR\_SAQUE = 0

SENHA\_SAQUE = 0

NUN\_CONTA = 0

**Início**

**Escreva** "Seja bem-vindo!"

**Escreva** "Informe o número da sua agência:"

**Leia** NUM\_AGENCIA

**Escreva** "Informe o número da sua conta:"

**Leia** NUN\_CONTA

// após ler os dados, o sistema deverá trazer do banco de dados o valor do saldo do usuário na variável SALDO

**Escreva** "Informe o valor que você do saque:"

**Leia** VALOR\_SAQUE

**Se** VALOR\_SAQUE> 0 e VALOR\_SAQUE<= SALDO **Então**

SALDO = SALDO - VALOR\_SAQUE

**Escreva** "Saque efetuado com sucesso. Seu saldo atual é R$:"

**Escreva** SALDO

// após apresentar o novo saldo o sistema deve fazer a contagem das notas e disponibilizar a quantia sacada ao usuário

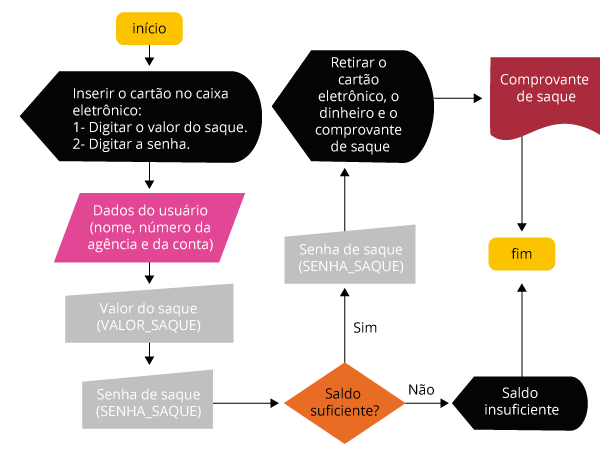
**Escreva** "Retire o cartão, o dinheiro e o comprovante de saque."

**Senão**

**Escreva** "Saldo insuficiente. Não será possível realizar a operação."

**Fim Se**

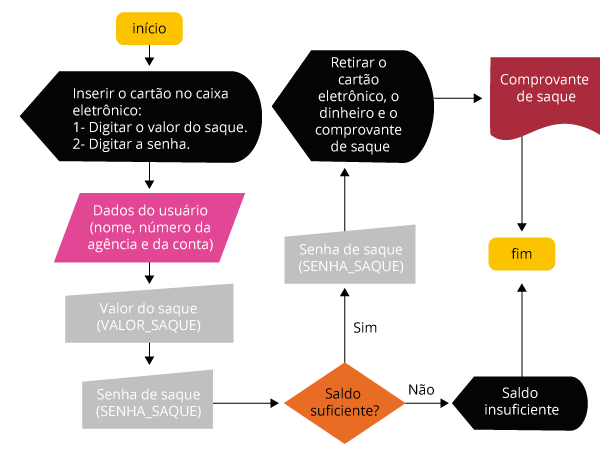
**Fim.**



**Descrição Narrativa**

1. **Entrar** no banco.
2. **Procurar** a fila do caixa eletrônico.
3. **Entrar** na fila do caixa eletrônico.
4. **Aguardar** a vez do atendimento.
5. **Inserir** o cartão eletrônico no caixa.
6. **Informar** o valor a ser retirado.
7. **Digitar** a senha.
8. **Pegar** o comprovante de saque, o cartão eletrônico e o dinheiro.
9. **Conferir** o valor em dinheiro.
10. **Sair** do banco.

**Fluxograma**

[](https://lms.ev.org.br/mpls/Custom/Cds/COURSES/2127-FUND_LOG_PROGR/pag/1_3_15.html#1_3_15_pop)

**Portugol**

**Algoritmo** Saque\_dinheiro\_caixa\_eletronico

// rotina para sacar dinheiro em um caixa eletrônico

**Var**

VALOR\_SAQUE, SALDO: real

SENHA\_SAQUE, NUM\_AGENCIA, NUN\_CONTA: inteiro

VALOR\_SAQUE = 0

SENHA\_SAQUE = 0

NUN\_CONTA = 0

**Início**

**Escreva** "Seja bem-vindo!"

**Escreva** "Informe o número da sua agência:"

**Leia** NUM\_AGENCIA

**Escreva** "Informe o número da sua conta:"

**Leia** NUN\_CONTA

// após ler os dados, o sistema deverá trazer do banco de dados o valor do saldo do usuário na variável SALDO

**Escreva** "Informe o valor que você do saque:"

**Leia** VALOR\_SAQUE

**Se** VALOR\_SAQUE> 0 e VALOR\_SAQUE<= SALDO **Então**

SALDO = SALDO - VALOR\_SAQUE

**Escreva** "Saque efetuado com sucesso. Seu saldo atual é R$:"

**Escreva** SALDO

// após apresentar o novo saldo o sistema deve fazer a contagem das notas e disponi- bilizar a quantia sacada ao usuário

**Escreva** "Retire o cartão, o dinheiro e o comprovante de saque."

**Senão**

**Escreva** "Saldo insuficiente. Não será possível realizar a operação."

**Fim Se**

**Fim.**