



Material do Aluno

Técnico em Desenvolvimento *Web e Cibersegurança*

Tecnologia, Lógica e Sociedade

Lógica de Programação Básica e Algoritmos via no Cod



**Instituto
Telles**

Parcerias:

SEDUC
Secretaria de Estado
da Educação

SECTI
Secretaria de
Estado de Ciência,
Tecnologia e Inovação



Sumário

Capítulo 01: LÓGICA DE PROGRAMAÇÃO BÁSICA E ALGORITMOS VIA NO-CODE	4
1.1 Introdução à lógica de programação	4
1.1.1 Do pensamento à ação: a lógica de programação como linguagem das máquinas	5
1.1.2 Explorando a lógica de programação	7
1.2 Algoritmos de tomada de decisão	7
1.1.1 Explorando um exemplo de algoritmo: como montar um cubo mágico?	8
1.3 Fluxogramas	19
1.4 Pseudo-linguagem	26
1.5 Loops e repetições	27
1.6 Estrutura de dados simples	31
1.7 Resolução de problemas práticos	40
Referências	48

CAPÍTULO 01

LÓGICA DE PROGRAMAÇÃO BÁSICA E ALGORITMOS VIA NO-CODE

O que esperar deste capítulo:

- Apresentação dos conceitos básicos de lógica de programação, como sequências, condições e repetições, e a exploração de como esses mesmos conceitos se aplicam à solução de problemas no contexto de *no-code*;
- Criação de fluxogramas enquanto formas visuais para representar algoritmos. Além disso, será explorado, na prática, como a elaboração de fluxogramas pode resolver problemas simples;
- Discussão sobre como criar algoritmos que tomam decisões com base em condições, usando blocos lógicos “se-então-senão”, bem como exemplos de como implementar essa lógica em ferramentas *no-code*;
- Introdução aos conceitos de *loops* e repetições para a execução de tarefas repetitivas, assim como a criação de algoritmos que utilizam *loops* em ferramentas *no-code*;
- Investigação acerca do armazenamento e manipulação de informações por meio de variáveis e listas simples;
- Aplicação de algoritmos em cenários do mundo real, como planejamento de horários, controle de estoque ou gerenciamento de tarefas, e desenvolvimento de soluções práticas com ferramentas *no-code*.

1.1 Introdução à lógica de programação

Embora se relacione com grandes gênios da história, a lógica, na verdade, está em tudo o que fazemos, seja decidir o que comer no café da manhã ou organizar as nossas tarefas diárias. É como se estivéssemos resolvendo pequenos quebra-cabeças o tempo

todo: cada escolha que fazemos envolve um tipo de raciocínio lógico para determinar a melhor opção.

Vamos observar algumas situações cotidianas nas quais costumamos utilizar a lógica:



Preparar uma comida



A receita funciona como um programa, contendo instruções precisas sobre ingredientes, etapas e tempo de cozimento, o que garante um prato delicioso.



Organizar uma viagem



A logística da viagem envolve planejamento, reservas e roteiros. Tudo isso é baseado em lógica, visando garantir uma experiência tranquila e agradável.



Jogar xadrez



A estratégia do jogo depende da análise do tabuleiro, da previsão de jogadas e da tomada de decisões lógicas para alcançar a vitória.

A lógica formal define, passo a passo, como solucionar um problema de forma clara e sem erros, mas ela também pode ser usada como uma linguagem! Assim como a linguagem nos permite comunicar ideias, a lógica de programação é a base fundamental para a criação das máquinas. Ela possibilita organizar ideias e instruções de modo que um computador possa entendê-las e executá-las corretamente. A seguir, vamos conferir como isso funciona.

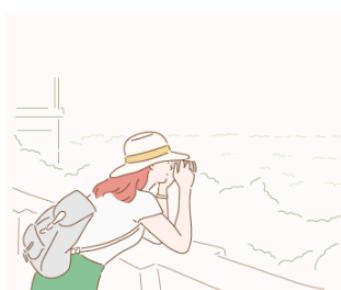
1.1.1 Do pensamento à ação: a lógica de programação como linguagem das máquinas

Reflita:



Hora de imaginar!

Se você tivesse a habilidade de aprender qualquer idioma instantaneamente, qual escolheria? Por quê?



As linguagens de programação são como idiomas que os computadores entendem e seguem para realizar tarefas específicas. Assim como aprendemos inglês, espanhol ou qualquer outro idioma para nos comunicarmos com pessoas, os programadores aprendem linguagens de programação, visando a comunicação com os computadores.

É importante ter em mente que uma linguagem de programação é, antes de tudo, uma linguagem estruturada. Isso significa que, além de uma sintaxe, cada uma delas possui organização lógica e regras específicas que determinam como o código deve ser escrito e executado.

Desvendando a sintaxe

Muitas pessoas possuem dúvidas em relação ao que a sintaxe realmente significa, mas não se preocupe: ela é mais simples do que parece!

As sintaxes são como as palavras e frases são organizadas para formar uma linguagem.

Quer um exemplo?

No português, uma frase precisa começar com uma letra maiúscula e terminar com um ponto final.



Geralmente, a sintaxe da linguagem de programação envolve organização, precisão e generalização. Para entender como todos esses elementos são ordenados, veja o exemplo a seguir, que utiliza uma receita de bolo para ilustrar a aplicação de cada um desses princípios.

	Organização	Precisão	Generalização
Definição	A lógica organiza o processo em etapas sequenciais, definindo um passo a passo claro para alcançar o objetivo final.	As instruções precisam ser precisas para que o computador possa interpretá-las e executá-las sem erros.	A lógica deve ser capaz de lidar com diferentes cenários e entradas de dados, adaptando-se às particularidades de cada situação.
Receita de bolo	Em uma receita de bolo, misturamos os ingredientes secos, depois os líquidos e, finalmente, combinamos tudo antes de assar.	A quantidade exata de cada ingrediente, bem como o tempo e a temperatura de cozimento, são cruciais para o resultado final.	Podemos fazer um bolo de chocolate ou um bolo de baunilha, seguindo a mesma estrutura básica da receita.

1.1.2 Explorando a lógica de programação

Você já deve ter ouvido falar que, para utilizar a lógica de programação, é necessário desenvolver um código em uma linguagem específica, como Python, JavaScript, C++ etc. No entanto, é importante destacar que existem diversas ferramentas e

abordagens que permitem expressar a lógica de programação de maneira mais ampla e acessível. Vamos descobrir quais são elas?

Algoritmos

São sequências finitas de instruções que descrevem como resolver um problema.

Pseudocódigos

É uma linguagem informal que facilita a escrita de algoritmos, utilizando linguagem natural e símbolos matemáticos.

Fluxogramas

São representações gráficas dos algoritmos, utilizando símbolos para visualizar as etapas e o fluxo de execução.

A seguir, exploraremos, de forma mais profunda, como cada uma dessas ferramentas funciona, com o objetivo de compreender melhor os seus mecanismos e aplicações.

1.2 Algoritmos de tomada de decisão



Imagine que você acabou de acordar. O que costuma fazer depois?

Ao responder o questionamento, muitas pessoas mencionariam certas atividades, como alongar o corpo, beber água, usar o banheiro, lavar o rosto e escovar os dentes. Algumas, ainda, poderiam preferir tomar banho e, logo depois, fazer uma refeição.

As respostas variam, pois dependem da rotina diária de cada um. É comum, porém, que essas etapas sigam uma ordem lógica. Por exemplo, é improvável que alguém acabe de acordar e saia imediatamente de casa.

Todos esses passos seguem uma ordem estruturada e que conduzem a determinado objetivo, assim como os algoritmos. Da mesma maneira que eles, a sua rotina diária apresenta uma abordagem sistemática para realizar tarefas. Nesse mesmo

sentido, os algoritmos são conjuntos de instruções ordenadas e finitas, projetadas para resolver um problema ou executar uma tarefa específica.

1.1.1 Explorando um exemplo de algoritmo: como montar um cubo mágico?

Muitas pessoas ficam impressionadas com a facilidade com a qual alguns indivíduos montam o cubo mágico. O que muitos não sabem, entretanto, é que isso é realizado seguindo um algoritmo específico! A seguir, observe como isso é feito:



BLOG ONCUBE. *Como resolver o cubo mágico: método básico - camadas.* [S.d.]. Disponível em: <<https://www.blog.oncube.com.br/tutoriais/tutorial-3x3x3/como-resolver-o-cubo-magico-metodo-basico-camadas-intro/>>.

Por que o algoritmo ficou associado ao computador

Os algoritmos são fundamentais para a programação de computadores, porque eles fornecem uma maneira clara e lógica de definir instruções para essas máquinas, permitindo que elas executem tarefas complexas de forma precisa e eficiente.

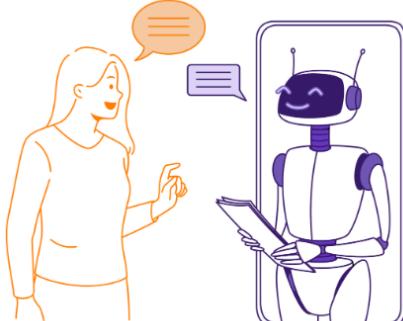
A associação entre algoritmos e computadores se deu por diversos motivos:

Precisão e execução automática	Permite a automação de tarefas complexas, liberando o ser humano de atividades repetitivas e propensas a falhas.
Eficiência e velocidade	Computadores processam informações a velocidades muito superiores às capacidades humanas. Por isso, ao executar algoritmos em computadores, obtemos soluções de forma mais rápida e mais eficiente.
Escalabilidade	Algoritmos podem ser facilmente adaptados para lidar com grandes volumes de dados e situações complexas. Isso torna essas máquinas ferramentas ideais quando se trata da resolução de problemas em escala superior, o que seria inviável para o ser humano.
Confiabilidade e reproduzibilidade	Algoritmos garantem resultados consistentes e previsíveis, independentemente de quem os executa.
Universalidade	A linguagem algorítmica é universal, podendo ser utilizada em diferentes áreas do conhecimento. Isso facilita a comunicação entre diferentes áreas e permite o compartilhamento de soluções.

Outro fator importante e que contribui para o uso de algoritmo no ambiente computacional é a crescente popularidade da inteligência artificial (IA) e do aprendizado de máquina (ML), o que reforça a importância dos algoritmos na era digital.

Algoritmo e inteligência artificial

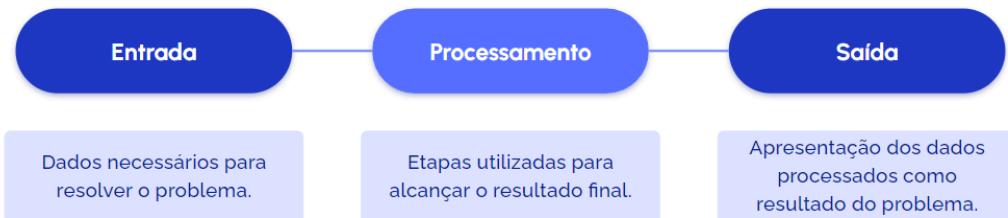
A inteligência artificial se refere à habilidade que os computadores têm de realizar atividades que, geralmente, só humanos conseguem, como **reconhecer padrões** ou **entender linguagem natural**. É daí que vêm os **algoritmos**: em redes sociais, por exemplo, eles são usados para analisar o que as pessoas fizeram antes, sugerindo conteúdo semelhante e que elas provavelmente gostarão.



Em resumo, algoritmos funcionam como instruções que os computadores seguem. Eles se beneficiam mutuamente: os algoritmos fornecem direções precisas para os computadores, que as executam de forma eficaz. Por sua vez, os computadores proporcionam um ambiente ideal para a execução de algoritmos complexos em grande escala. Essa relação é fundamental para o avanço da tecnologia e da sociedade.

Etapas de um algoritmo

Um algoritmo é uma sequência lógica de instruções executáveis. Sendo assim, qualquer tarefa seguindo um padrão pode ser descrita por um algoritmo, o que inclui a elaboração de uma receita de bolo. Ao criar um algoritmo, é essencial dividir as informações em três fases principais:



Como é possível ver, um algoritmo começa com a entrada de dados, etapa na qual recebe as informações necessárias para realizar sua tarefa. Após receber esses dados, o algoritmo os processa, geralmente realizando operações ou cálculos a partir deles. Finalmente, o algoritmo gera uma saída, que é a resposta ou resultado do processamento realizado com os dados de entrada. Observe o exemplo. Nele, um algoritmo verifica se determinado número é par:



Vamos explorar a história do algoritmo?

A máquina de Turing é considerada um modelo teórico fundamental para entender o que é computável e como funciona a computação. Foi idealizada por Alan Turing, que, posteriormente, ficou responsável por introduzir a ideia de algoritmos. Acompanhe esse processo na linha do tempo a seguir.

Invenção da máquina em 1936

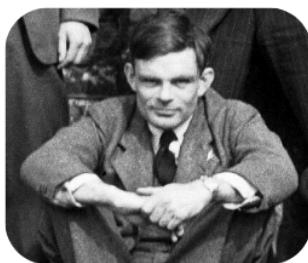
Desenvolvimento de linguagens de programação (a partir de 1950)

Em 1936, Alan Turing inventou uma máquina imaginária que poderia seguir regras simples para resolver problemas matemáticos.

Isso ajudou as pessoas a entenderem melhor como os algoritmos funcionam, mostrando que problemas podem ser resolvidos de forma sistemática e mecânica.

A partir dos anos 1950, as linguagens de programação foram criadas, visando tornar a escrita de instruções para computadores mais fácil.

Elas permitem que os programadores escrevam instruções de forma parecida com a linguagem humana, facilitando a criação de algoritmos em computadores.



Alan Turing, em 1950.



Vista do Bletchley Park, complexo de edifícios onde os decifradores de códigos trabalharam com Alan Turing durante a Segunda Guerra Mundial. Aqui, desenvolveram a máquina Enigma e os primeiros computadores.

Algoritmos de tomada de decisão

Suponha que você esteja decidindo se deve ou não sair de casa. No fim das contas, a sua decisão depende do clima. Provavelmente, alguns pensamentos e considerações passarão pela sua cabeça.

Devo sair de casa?



Se estiver ensolarado

Verificar se você tem tempo livre.

Se tiver tempo livre, sair de casa.

Se não tiver tempo livre, ficar em casa.



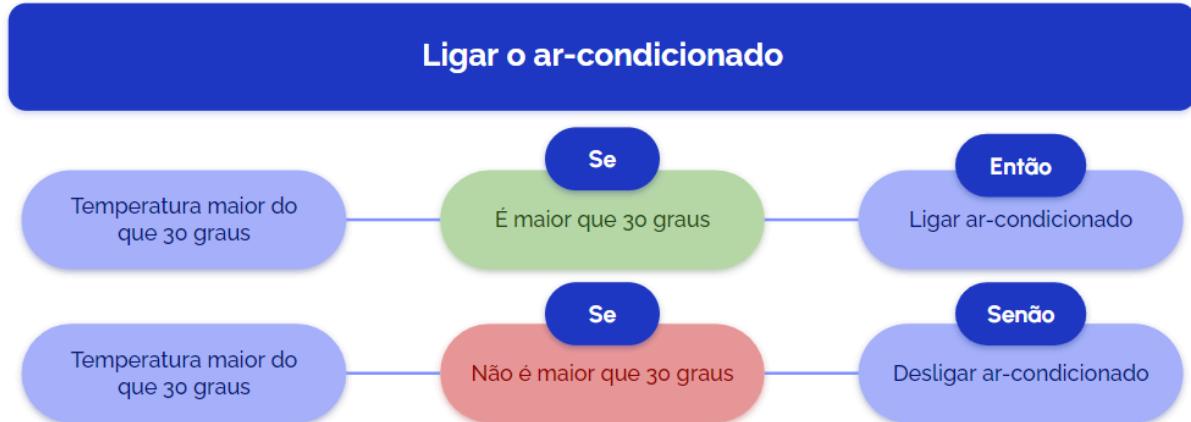
Se estiver chovendo

Verificar se você tem um guarda-chuva.

Se tiver um guarda-chuva, sair de casa.

Se não tiver um guarda-chuva, ficar em casa.

É dessa forma que os algoritmos de tomada de decisão funcionam. Eles são usados para tomar decisões com base em condições específicas, usando blocos lógicos de “se-então-senão” para avaliar as condições e tomar decisões com base nos resultados. Por exemplo, considere o seguinte algoritmo simples:



No exemplo, a entrada de dados é a temperatura, o que significa que a saída é a ação de ligar ou desligar o ar-condicionado. O algoritmo avalia a temperatura e toma uma decisão com base na condição especificada. Os algoritmos de tomada de decisão podem ser usados em uma variedade de aplicações, desde sistemas de controle de processos industriais até sistemas de recomendação de produtos *on-line*. Eles também podem ser usados para tomar decisões complexas, como diagnósticos médicos ou análises de risco financeiro.

Para criar um algoritmo de tomada de decisão, é necessário definir as condições que serão avaliadas e as ações que serão tomadas com base nos resultados. É importante, também, considerar todas as possíveis condições e ações, garantindo que o algoritmo seja preciso, generalizado e finito.

Exemplos de algoritmos de tomada de decisão

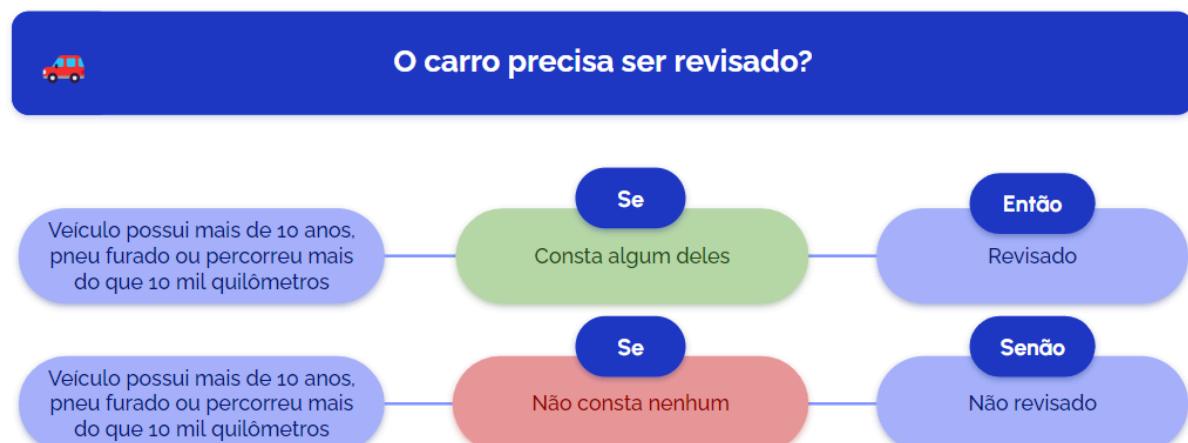
1. Algoritmo para decidir se uma pessoa pode doar sangue:



2. Algoritmo para decidir se um produto deve ser recomendado a um cliente:



3. Algoritmo para decidir se um veículo deve ir para a revisão:



É importante notar que esses algoritmos são apenas exemplos e podem ser adaptados de acordo com as necessidades específicas de cada situação.

Variáveis

Em programação, uma variável é um espaço reservado em memória que armazena um valor alterado durante a execução do programa. É daí que vem o nome “variável”: pode variar de propriedades a depender do contexto. Por exemplo:

1

Em um programa que calcula a média de idade de um grupo de pessoas, você pode ter uma variável chamada `"idade_da_pessoa"` para armazenar a idade de cada indivíduo.

2

Em um jogo, você pode ter uma variável chamada `"pontuação"` para rastrear quantos pontos o jogador acumulou.

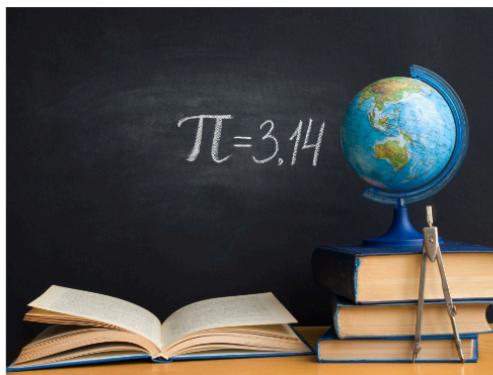
Cada variável deve ter um nome único, chamado de identificador, e pode ter um tipo específico de dado, como números inteiros, reais, caracteres ou um conjunto de caracteres (*string*), entre outros. Para entender melhor, imagine uma caixa.



Assim como uma caixa física pode conter algo dentro dela, em programação, uma variável pode conter um valor específico. A analogia com uma caixa é útil para entendermos que podemos armazenar e manipular diferentes tipos de informações em uma variável, assim como podemos adicionar e retirar objetos de uma caixa física.

Constantes

Diferente das variáveis, as constantes são utilizadas para armazenar valores fixos durante a execução do algoritmo. No entanto, assim como as variáveis, as constantes devem ter um nome único e significativo, além de um tipo de dado associado. Não podemos nos esquecer, ainda, que o valor de uma constante não pode ser alterado durante a execução do algoritmo.



Um exemplo de constante é o número irracional pi (π), que possui um valor aproximado de 3.1415 e não pode ser alterado.

É importante utilizar o tipo de dado apropriado para cada variável ou constante, para garantir a integridade e precisão dos cálculos, bem como operações, realizados no programa. Além disso, os tipos de dados podem variar de acordo com a linguagem de programação utilizada.

Elementos utilizados nos algoritmos

Os algoritmos são compostos por diferentes elementos que ajudam a definir as etapas necessárias para resolver um problema específico, definindo qual será o tipo de dado: constante ou variável. Os principais tipos de dados são:

	Definição	Exemplos
Numéricos	São utilizados para representar valores numéricos, podendo ser inteiros ou reais.	<ul style="list-style-type: none">• Inteiros: 42, -7, 0, 1000;• Reais: 3.14159.
Caracteres	São utilizados para representar letras, símbolos e outros caracteres especiais.	<ul style="list-style-type: none">• "A", "a", "#", " " (espaço em branco).
Lógicos	São utilizados para representar valores booleanos, ou seja, verdadeiro ou falso .	<ul style="list-style-type: none">• Verdadeiro - true;• Falso - false.
Texto	São utilizados para representar sequências de caracteres, como palavras ou frases.	<ul style="list-style-type: none">• "Olá, mundo!";• "12345" (sequência de caracteres, não um número);• "Isso é um texto, ou seja, uma string com múltiplas palavras".

Vale relembrar!

Como visto em capítulos anteriores, para criar bons algoritmos, é essencial entender os diferentes tipos de operadores na programação. Esses operadores são usados para realizar ações com valores e variáveis e são classificados em grupos, como aritméticos, relacionais, lógicos, de atribuição, entre outros. Cada grupo de operadores têm suas próprias regras e é importante usá-los corretamente, obtendo, assim, os resultados desejados.

Veja os principais tipos de operadores usados na programação:

- Operadores aritméticos:** são utilizados para realizar operações matemáticas básicas, como adição, subtração, multiplicação, divisão e resto da divisão.

Por exemplo:

Adição (+)	$5 + 3 = 8$
Subtração (-)	$5 - 3 = 2$
Multiplicação (x)	$5 \times 3 = 15$
Divisão (÷)	$5 \div 3 = 1.6666666666666667$
Resto da divisão (%)	$5 \% 3 = 2$

- 2. Operadores relacionais:** são utilizados para comparar valores e retornar um valor booleano (verdadeiro ou falso), indicando se a comparação é verdadeira ou falsa.

Por exemplo:

Igualdade (==)	$5 == 3$ (falso)
Desigualdade (!=)	$5 != 3$ (verdadeiro)
Maior que (>)	$5 > 3$ (verdadeiro)
Menor que (<)	$5 < 3$ (falso)
Maior ou igual a (>=)	$5 \geq 3$ (verdadeiro)
Menor ou igual a (<=)	$5 \leq 3$ (falso)

- 3. Operadores lógicos:** são utilizados para combinar valores booleanos e realizar operações lógicas, como AND (e), OR (ou) e NOT (não).

Por exemplo:

AND verdadeiro E verdadeiro	Se estiver chovendo E estiver frio, então levo um guarda-chuva.
OR verdadeiro OU falso	Se estiver chovendo OU estiver ventando, então levo um guarda-chuva.
NOT não verdadeiro	Se não estiver chovendo, então NÃO levo um guarda-chuva.

4. Operadores de atribuição: são utilizados para atribuir valores às variáveis.

Por exemplo:

Atribuição (<code>=</code>)	<code>X = 5</code>
Adição e atribuição (<code>+=</code>)	<code>X+ = 3</code> (equivalente a <code>X = X + 3</code>)
Subtração e atribuição (<code>-=</code>)	<code>X- = 3</code> (equivalente a <code>X = X - 3</code>)
Multiplicação e atribuição (<code>*=</code>)	<code>X*= 3</code> (equivalente a <code>X = X \times 3</code>)
Divisão e atribuição (<code>/=</code>)	<code>X÷ = 3</code> (equivalente a <code>X = X ÷ 3</code>)
Resto da divisão e atribuição (<code>%=</code>)	<code>X %= 3</code> (equivalente a <code>X = X % 3</code>)

5. Operadores unários: são utilizados para realizar operações em um único valor.

Por exemplo:

Incremento	<code>x++</code> (equivalente a <code>X = X + 1</code>)
Decremento	<code>x--</code> (equivalente a <code>X = X - 1</code>)
Negação	<code>-X</code> (equivalente a <code>X = -X</code>)

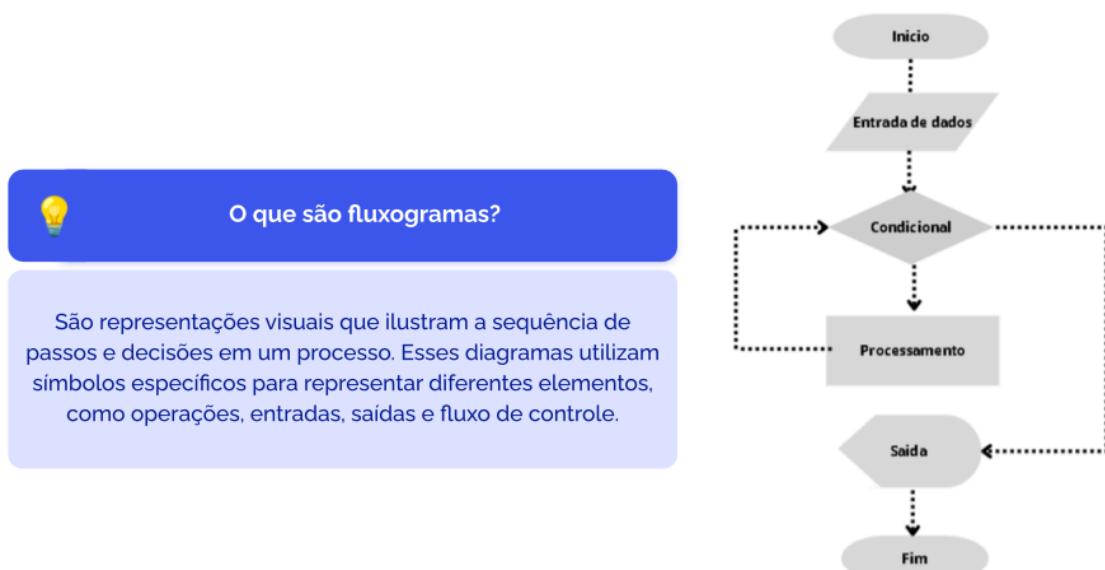
Cada linguagem de programação pode ter seus próprios operadores e sintaxe específica, mas esses são os principais tipos de operadores utilizados. Para saber mais sobre Lógica de Programação e Algoritmo, assista o vídeo:



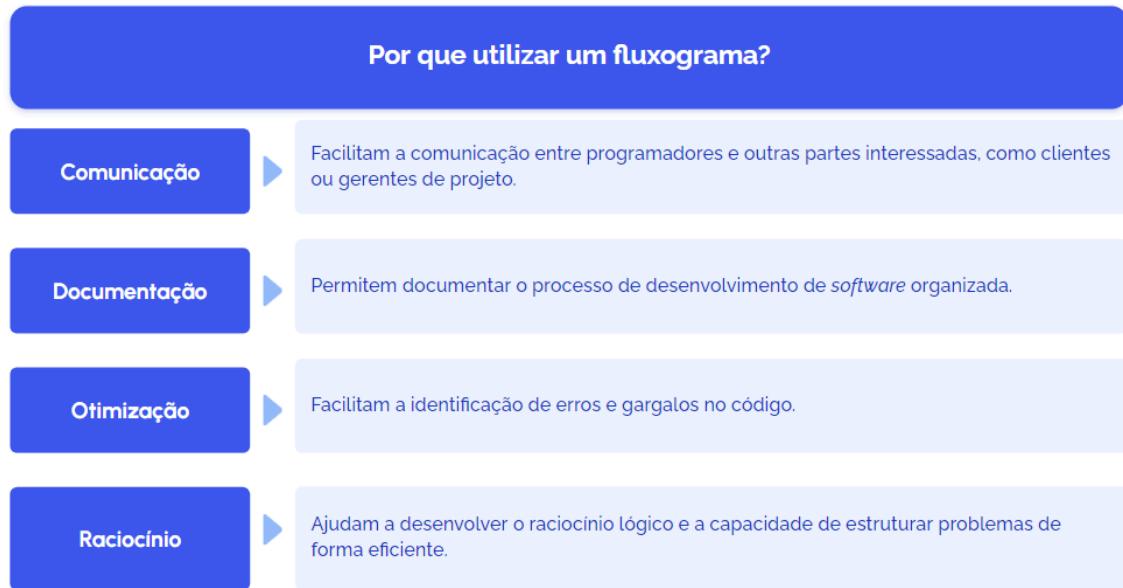
Vídeo no YouTube *Entenda LÓGICA DE PROGRAMAÇÃO e ALGORITMOS - Aula 01*, do canal Sharpax (22 maio 2019). Disponível em: <<https://youtu.be/JaTf3dhx464?si=2WhmVX9572qF7eGF>>. Acesso em: 12 mar. 2024.

1.3 Fluxogramas

Durante o estudo da lógica de programação, podemos utilizar uma ferramenta que ajuda na representação visual dos algoritmos: os fluxogramas.



Em essência, eles funcionam como um mapa visual, utilizando símbolos para representar diferentes etapas e decisões, facilitando a compreensão da lógica por trás do código.



Para muitos especialistas, o fluxograma é considerado a forma universal de representação, pois emprega figuras geométricas padronizadas para ilustrar os passos necessários à resolução de problemas. De forma geral, um algoritmo deve ser compreendido da mesma maneira por diferentes indivíduos que o utilizam, e é nesse sentido que o fluxograma é mais útil. Entretanto, para uma interpretação precisa de um fluxograma, é essencial compreender a sua sintaxe e a sua semântica.

VS

Sintaxe

Refere-se ao uso correto dos símbolos gráficos e das expressões dentro dele. Isso significa que cada símbolo tem uma maneira específica de ser usado, havendo regras sobre como eles podem ser combinados.

Semântica

Refere-se ao significado de cada símbolo. É como interpretamos cada elemento do fluxograma para entender o que ele representa no contexto do algoritmo.

Além de entender como cada um desses tópicos é organizado em um fluxograma, é importante ficar atento a algumas regras específicas:

1

A interpretação de um fluxograma ocorre de **cima para baixo** e da **esquerda para a direita**.

2

Pode não ser adequado para representar algoritmos maiores e/ou mais complexos. Nessas situações, é comum recorrer ao pseudocódigo como forma alternativa de representação.

Agora, vamos conhecer a representação gráfica do fluxograma:



Figura para definir inicio e final do algoritmo.



Figura utilizada na representação de entrada de dados inserida.



Figura utilizada na representação de saída de dados inserida.



Figura utilizada no processamento de cálculo, atribuições e processamento de dados no geral.



Figura que indica o processo seletivo ou condicional, possibilitando o desvio do caminho no processo.



Símbolo que identifica o sentido no fluxo de dados, permitindo a conexão entre as outras figuras existentes.

Exemplo de fluxograma

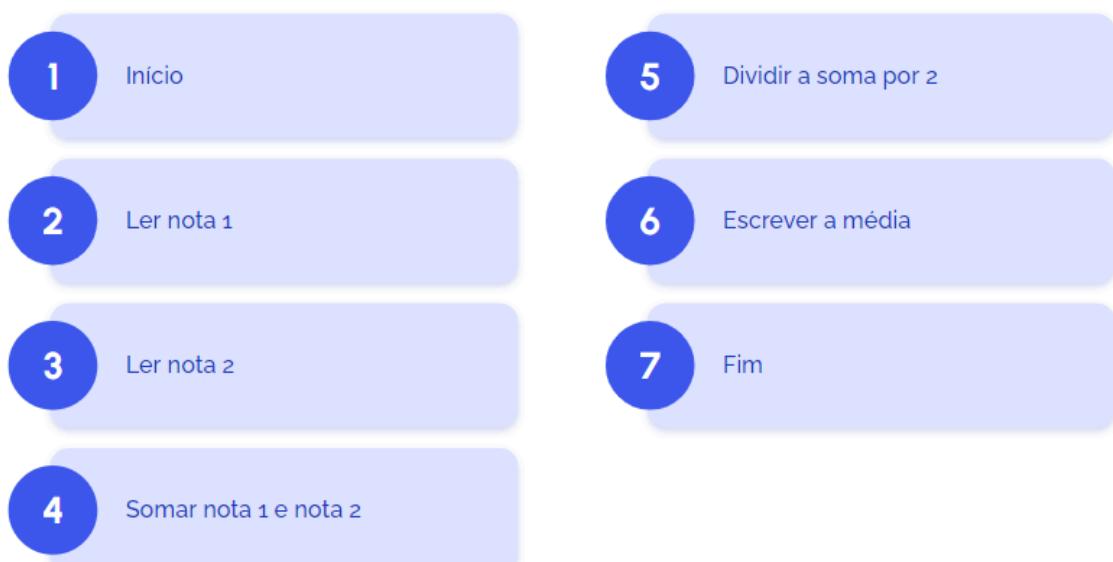
Observe a situação a seguir:

Fim de semestre

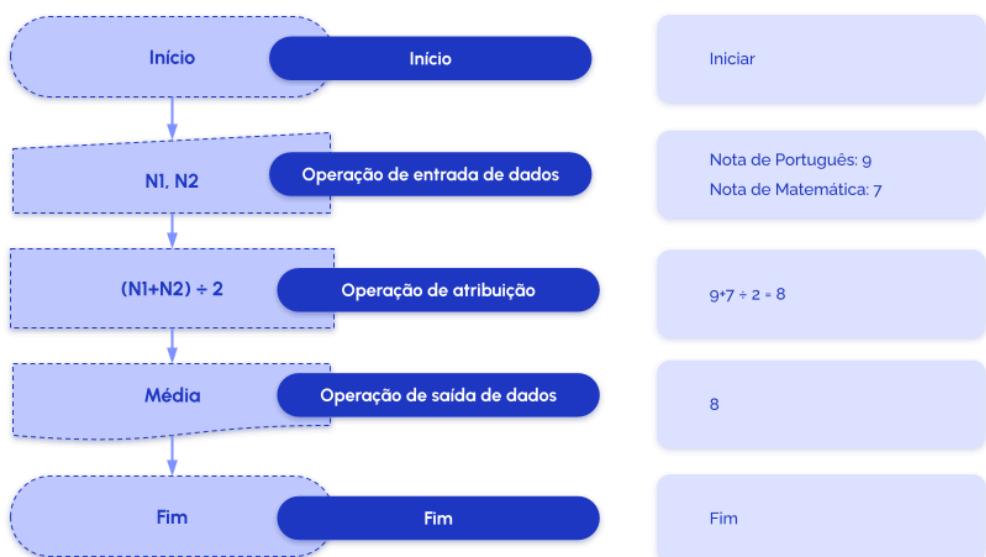
Filipe está indeciso sobre suas férias. Sua mãe estabeleceu que, se ele conseguir uma média de notas **maior do que 8** em Português e Matemática, eles viajarão para a casa da avó. Ansioso, Filipe optou por utilizar um fluxograma, visando determinar se alcançará ou não esse objetivo.



Para calcular a soma de suas notas, Filipe decidiu desenvolver o seguinte algoritmo:



Por meio desse passo a passo, ele chegou ao seguinte fluxograma:

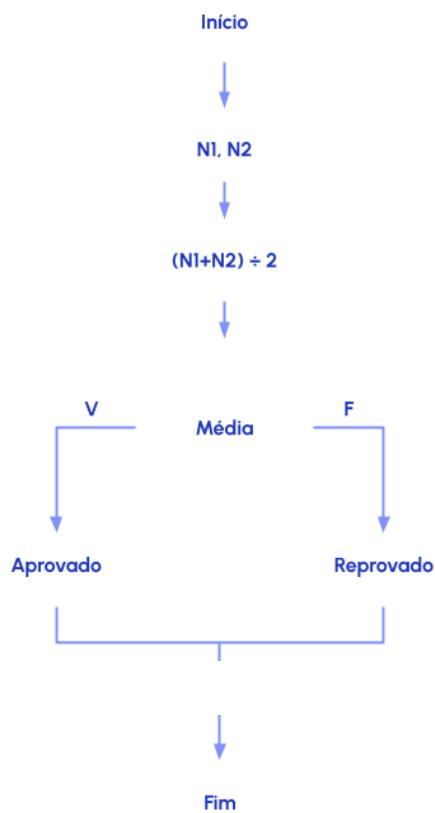


Usando o raciocínio lógico do fluxograma, Filipe percebeu que é possível alcançar pontuação suficiente para poder viajar para a casa de sua avó. Com isso, ele vai poder aproveitar as suas férias e se divertir ao máximo!

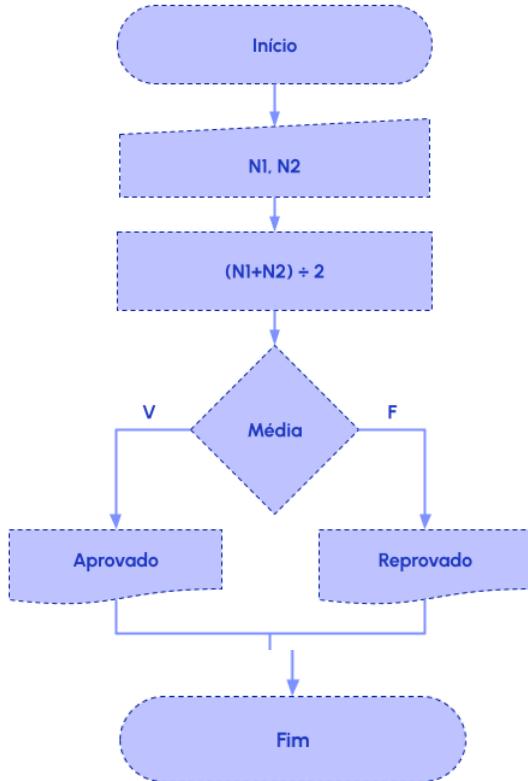
E aí, gostou do exemplo? Que tal praticar um pouco?

Imagine que você é um professor e precisa determinar se um aluno foi aprovado ou reprovado, tomando como base a média das suas notas. A média mínima para aprovação é **7** e, sabendo disso, você já calculou as notas de duas provas que o aluno realizou: **8** e **9**.

A seguir, veja um fluxograma que descreve as etapas para essa decisão. Considere, porém, que os blocos do fluxograma não estão configurados corretamente para representar cada fase da decisão, uma vez que sua tarefa é identificar o formato correto de cada um deles.



E aí, conseguiu resolver? Veja o gabarito:



Continue aplicando o seu conhecimento e resolvendo questões de maneira prática! Para exercitar ainda mais o que você aprendeu até aqui, observe o próximo desafio.

DESAFIO PRÁTICO

Planejando um piquenique

Descrição

Vocês estão organizando um piquenique e querem garantir que tudo seja perfeito. O desafio é criar um fluxograma que represente as etapas simples para planejar o evento, desde a escolha do local até a hora de recolher a sujeira. Cada decisão deve levar a um próximo passo lógico no planejamento.



Objetivos

Usar conceitos básicos de lógica de programação por meio do fluxograma.

Orientações

- Revisem os conceitos básicos de fluxograma como símbolos de processo de decisão;
 - O grupo deve identificar as etapas-chave do planejamento de um piquenique, como a escolha do local e a preparação de alimentos. Além disso, a equipe deve considerar se o dia estará chuvoso ou não, se haverá atividades, entre outras informações e decisões que o time, com a criatividade de seus integrantes, irá informar;
 - Desenhem o fluxograma em papel ou cartaz, utilizando símbolos e setas para indicar as etapas do planejamento;
 - Certifiquem-se de que o fluxograma seja claro e fácil de entender;
 - Apresentem o que foi feito para a turma, explicando como as decisões permitiram alcançar as diferentes etapas do planejamento.
-

1.4 Pseudo-linguagem

O pseudocódigo é uma maneira de representar algoritmos de forma mais próxima à linguagem humana, o que facilita a compreensão e a comunicação entre programadores. Ele é menos formal e mais flexível do que uma linguagem de programação real, permitindo descrever a lógica e a sequência de passos de um algoritmo de forma mais fácil de entender.

É importante ressaltar que um pseudo-código utiliza frases que correspondem às estruturas utilizadas em uma linguagem de programação. Veja como esse processo ocorre no exemplo a seguir.

Consegue perceber a sua semelhança com o fluxograma?

```

Algoritmo CalcularMediaAlunos
  LER Nota1
  LER Nota2
  LER Nota3
  media = (Nota1+Nota2+Nota3)/3
  SE MEDIA >= 6 ENTÃO
    IMPRIME "Aprovado"
  SENÃO
    IMPRIME "Reprovado"
  FIM-SE
  IMPRIME o valor da média
Fim-Algoritmo CalcularMediaAlunos

```

Neste exemplo, as palavras "LER", "IMPRIME" e os símbolos "=" e "+" correspondem às estruturas utilizadas em uma linguagem de programação. As palavras "SE", "ENTÃO", "SENÃO" e "FIM-SE" são utilizadas para representar a estrutura condicional "**if-else**" em português estruturado. Já as palavras "ALGORITMO", "FIM-ALGORITMO" e "CALCULARMEDIAALUNOS" são utilizadas para definir o início e o fim do algoritmo, assim como dar um nome a ele, respectivamente.

Para se aprofundar um pouco mais no assunto, explore a plataforma Portugol Studio, uma linguagem de programação educativa utilizada para ensinar os conceitos básicos de programação. Ela possui uma sintaxe simples e fácil de entender, o que a torna uma boa opção para quem está começando a aprender a programar. Saiba mais no vídeo "Programação para Iniciantes usando Portugol Studio":



Vídeo no YouTube *Programação para Iniciantes usando Portugol Studio*, do canal Portugol Studio (31 mar. 2019). Disponível em: <<https://encurtador.com.br/gvBWq>>. Acesso em: 15 mar. 2024.

1.5 Loops e repetições

Simplificando tarefas repetitivas

Veja o exemplo.

Você precisa contar quantas pessoas estão na fila de um *show*. Por isso, começa a contar pela primeira pessoa e, em seguida, verifica se há alguém atrás dela. Se houver, você adiciona mais um à contagem e passa para o próximo. Essa ação se repete até que você chegue ao final da fila, quando não há mais ninguém para contar.

Na sua opinião, em uma escala de 0 a 10, o quanto isso seria cansativo?

Contar a quantidade de pessoas em um *show* nem sempre é uma tarefa fácil. Isso porque a multidão pode estar em constante movimento, entrando e saindo, o que dificulta uma contagem precisa e rápida. Além disso, em locais muito lotados, como festivais ou grandes eventos, a densidade populacional pode ser alta, tornando ainda mais desafiador estimar o número exato de presentes. Considerando esse exemplo, um *loop* seria útil para percorrer cada indivíduo da fila e realizar a contagem automaticamente.

O que são loops?

Em termos gerais, um *loop* é uma sequência de ações ou passos que são repetidos diante de uma propriedade específica. Isso pode ser comparado a uma roda que gira repetidamente até que seja interrompida ou a um ciclo que se repete a cada dia, como acordar, tomar café, trabalhar e dormir.

Por exemplo:



Tente imaginar

Em uma tarefa diária, como lavar pratos, você pode utilizar um *loop* para repetir a mesma sequência de ações (enxaguar, esfregar, enxaguar novamente e colocar para secar), fazendo isso até que todos os pratos estejam limpos. Depois, **quando não há mais pratos sujos para lavar**, o *loop* é interrompido.

Já imaginou como, nesse caso, um *loop* seria útil? Em programação, eles podem ser usados para processar os dados em lotes, evitando a sobrecarga de memória e mantendo a eficiência do programa.

Os diferentes tipos de loops e repetições

Existem diversos loops e repetições, mas alguns são mais comuns, como os apresentados a seguir.

- *Loop “repita”*: executa uma sequência de instruções até que uma determinada condição seja atendida.

Imagine só: você está tentando ligar para um amigo. Se a ligação não for bem-sucedida (por exemplo, se o telefone estiver ocupado ou for desligado), é possível esperar um pouco e tentar novamente. Você continua repetindo essa ação até que a ligação seja bem-sucedida e você consiga falar com o seu colega.



Aqui, o *loop “repita”* é usado para executar a mesma ação (tentar ligar para o amigo) até que uma condição específica seja atendida (a ligação ser bem-sucedida).

Em programação, o *loop “repita”* é usado para executar um bloco de código até que uma condição seja atendida. Por exemplo, você pode usar um *loop “repita”* para solicitar uma senha do usuário e verificar se ela está correta. Se a senha estiver incorreta, há a possibilidade de solicitar que o usuário digite novamente, então a ação continua se repetindo até que a senha esteja certa. Nesse caso, a condição *loop* seria “repita até que a senha esteja correta”.

- *Loop “para” ou “for”*: executa uma sequência de instruções um número fixo de vezes, com base em um contador.

Por exemplo:



Você tem uma lista para uma festa e, nela, há 10 convidados. É possível começar com o primeiro convidado e entregar um convite para ele. Depois, você avança para o segundo convidado e faz o mesmo. Desse modo, você continua repetindo a ação até chegar ao décimo convidado, quando o *loop* é interrompido e, por isso, há a certeza de que todos os convites foram entregues.

Nesse caso, o *loop* “para” é usado para executar a mesma ação (entregar um convite) um número fixo de vezes (10 vezes), com base em um contador (a lista de convidados).

Em programação, o *loop* “para” é usado para realizar um bloco de código um determinado número fixo de vezes, tendo um contador como base para isso. Por exemplo, você pode usar um *loop* “para” com o objetivo de transmitir números de 1 a 10 em uma tela, com base em um contador que começa em 1 e vai até 10.

- *Loop “enquanto” ou “while”* (em inglês): executa uma sequência de instruções enquanto uma determinada condição for verdadeira.

Por exemplo:

Imagine que você esteja jogando um jogo de tabuleiro com amigos. Você pode começar a partida com um determinado número de rodadas. Em cada rodada, cada jogador faz uma jogada e ganha ou perde pontos com base nas regras do jogo. No final da rodada, você verifica se o jogo acabou (por exemplo, se alguém atingiu um determinado número de pontos ou se todas as rodadas foram jogadas). Se o jogo ainda não tiver acabado, a sequência de jogadas é repetida na próxima rodada. Assim, você continua repetindo essa ação até que o jogo acabe.



Nesse caso, o *loop* “enquanto” é usado para executar a mesma sequência de ações (jogar uma rodada do jogo) enquanto uma determinada condição for verdadeira (o jogo ainda não acabou).

Em programação, o *loop* “enquanto” é usado para executar um bloco de código enquanto a condição específica for verdade. Por exemplo, você pode usar um *loop* “enquanto” para ler uma lista de números e parar quando chegar ao final da lista. Nesse caso, a condição seria “enquanto houver mais números na lista”.

Para criar um algoritmo com *loops*, é necessário seguir os seguintes passos:

1

Definir a condição de parada do *loop*: antes de criar o *loop*, é necessário definir a condição que fará com que ele pare de executar as instruções;

2

Selecionar o tipo de *loop*: de acordo com as necessidades do algoritmo, é necessário escolher o tipo de *loop* mais adequado (“para”, “enquanto” ou “repita”);

3

Definir as instruções do *loop*: dentro do *loop*, é necessário definir as instruções que serão executadas repetidamente;

4

Testar o *loop*: após criar o *loop*, é necessário testá-lo com diferentes entradas, a fim de garantir que ele esteja funcionando corretamente.

1.6 Estrutura de dados simples

As estruturas de dados são fundamentais na programação, visto que permitem armazenar e manipular informações de maneira sistemática. Neste tópico, exploraremos como utilizar variáveis e listas simples para armazenar e manipular dados.

O que são dados?

Um dado é uma unidade de informação que pode ser identificada e acessada por meio de um nome ou identificador, como uma variável em linguagens de programação.

A maioria das linguagens de programação utiliza variações de quatro tipos:

Inteiro (INT)	Representa valores numéricos inteiros, positivos ou negativos.
Ponto flutuante (FLOAT)	Representa valores numéricos com casas decimais, positivos ou negativos.
Booleano (BOOLEAN)	Representa apenas dois valores, verdadeiro ou falso, também conhecidos como operadores lógicos.
Texto (TEXT)	Representa sequências ou cadeias de caracteres, utilizados para manipular textos e outros tipos de dados não numéricos ou booleanos.

Na computação, lidar com conjuntos de dados é uma tarefa comum. No entanto, o desafio é **como** organizar e manipular esses dados de maneira eficiente. É aí que entram em cena as estruturas de dados, pois elas funcionam como ferramentas poderosas para gerenciar e acessar informações com velocidade e precisão.

O que são estruturas de dados?

Em termos simples, as estruturas de dados são diferentes maneiras de organizar e armazenar dados em um computador. Imagine uma estante de livros, sendo que, nela, cada livro está em um local específico, facilitando a busca e a organização. As estruturas de dados funcionam da mesma forma, mas digitalmente.

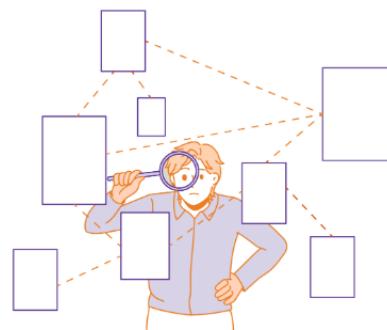
Tipos de estruturas de dados

- **Array:** é uma estrutura de dados que armazena uma coleção ordenada de elementos do mesmo tipo (como números ou *strings*) em posições sequenciais na memória do computador. Ele também pode ser conhecido como vetor, matriz ou arranjo. A fim de facilitar a compreensão, veja o exemplo a seguir:

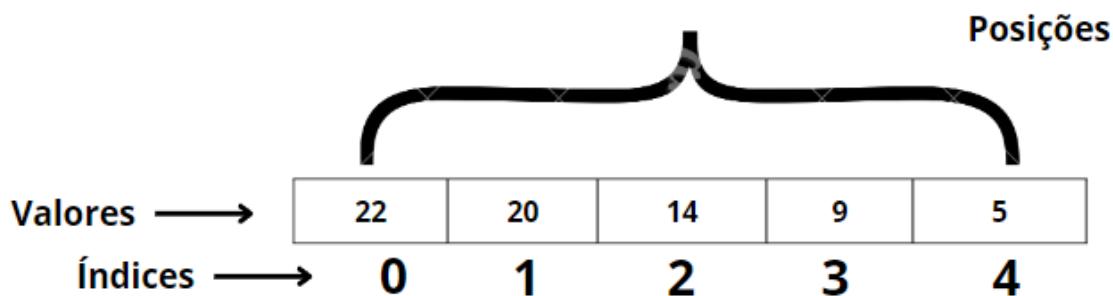
Imagine que você tem uma caixa grande e, dentro dela, há várias gavetas numeradas de 0 a N, sendo N o número total de gavetas **menos um**.

Cada gaveta, além disso, pode conter um objeto. Por exemplo, na gaveta 0 pode ter uma maçã, na gaveta 1 pode haver uma banana, na gaveta 2 pode estar uma laranja e assim por diante.

Um array é, basicamente, como essa caixa cheia de gavetas.



Um *array* contém valores que, por sua vez, possuem um índice que determina a posição que o elemento está nessa estrutura. A maioria das linguagens de programação define o índice inicial do *array* como **0**.



Para manipular um *array*, podemos realizar as seguintes operações:

Inserir (INSERT)

Adicionar um novo elemento no *array* em uma posição específica, manipulando o seu índice.

Receber (GET)

Obter o valor de um elemento do *array* a partir de seu índice.

Excluir (DELETE)

Remover um elemento do *array* a partir de seu índice e realizar os ajustes necessários nos demais elementos.

Saber o tamanho (SIZE)

Obter o número de elementos presentes no *array*.

Exemplos de arrays

Imagine uma lista de notas de alunos em uma turma. Nela, cada nota é um elemento do *array* e todas elas são do mesmo tipo (números). Assim, o *array* pode ser ordenado de forma crescente ou decrescente, permitindo que sejam realizadas buscas e operações em suas posições de forma eficaz.

```
lista_notas = [9.50, 8.20, 8.00, 7.98, 7.50, 6.60, 6.00, 5.90, 5.50, 4.00]
```

Um outro exemplo é um *array* de nomes de clientes de uma empresa, no qual cada nome é uma *string* (ou conjunto de caracteres). Esses nomes podem ser ordenados alfabeticamente para facilitar a busca por um em específico.

```
lista_clientes = ["André", "Amanda", "Carlos", "Diana", "Gabriel", "Hugo"]
```

- **Pilhas:** funcionam como um “empilhamento” de dados. Esse tipo de estrutura é semelhante a uma lista, mas segue o paradigma **LIFO** (*Last In, First Out*), o que significa que o último elemento a ser adicionado é o primeiro a ser removido.

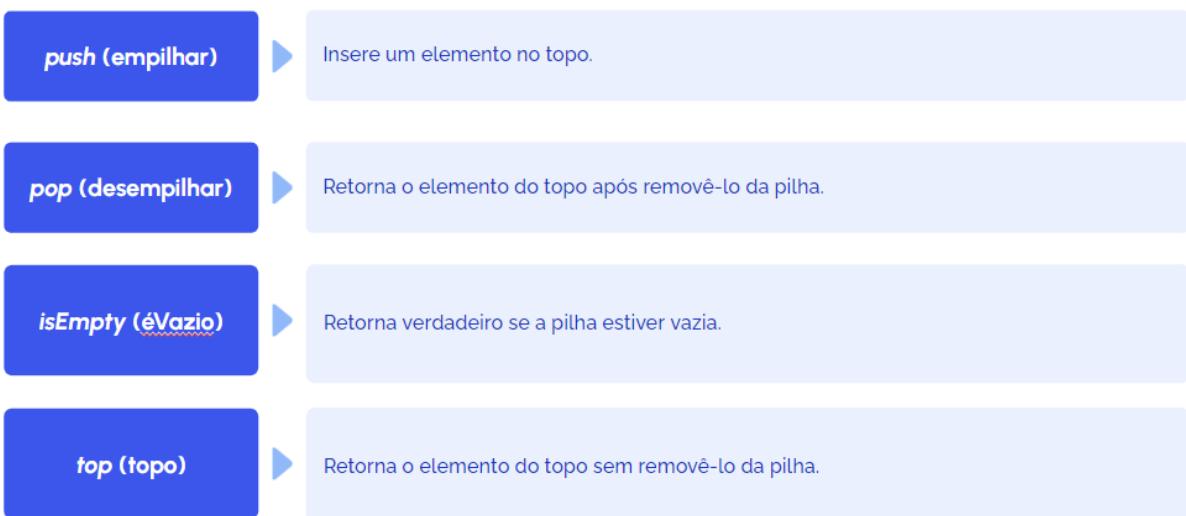
Podemos imaginar uma pilha de pratos. Ao empilhar pratos, o primeiro a ser retirado é sempre o último que foi colocado. O último prato adicionado é o primeiro a ser retirado.



No contexto de programação, existem apenas dois métodos para manipular dados em uma pilha:

- a) inserir um elemento no topo;
- b) remover um elemento do topo.

Diferente dos *arrays*, geralmente, as linguagens de programação não oferecem métodos nativos para criação e manipulação de pilhas. No entanto, é possível utilizar métodos de *array* para implementar pilhas.



Um exemplo de pilha é a implementação do botão “**voltar**” em um navegador *web*.

Implementação do botão “voltar”



Cada página visitada é adicionada à pilha. Quando o usuário clica no botão “voltar”, a última página adicionada à pilha é removida e exibida novamente.

- **Filas:** ao contrário de pilhas, funcionam como uma “fila de espera”. Aqui, o primeiro elemento adicionado é o primeiro a ser removido. A lógica segue o

paradigma **FIFO** (*First In, First Out*), o que significa que o primeiro elemento a entrar é o primeiro a sair.

Imagine uma fila de bilheteria como exemplo. A pessoa que chegou primeiro vai ser atendida antes de todos e, consequentemente, vai adquirir seu ingresso antes daqueles que chegaram depois e estão atrás dela. A fila, como estrutura de dados, adere rigorosamente a esse mesmo princípio.



Portanto, assim como na pilha, existem apenas duas operações fundamentais para manipular uma fila:

- a) adicionar um elemento ao final da fila;
- b) remover um elemento do início da fila.

Para manipular uma fila, podemos realizar as seguintes operações:

enqueue (enfileirar)

Insere um elemento no final da fila.

dequeue (desenfileirar)

Retorna o elemento do início após removê-lo da fila.

isEmpty (éVazia)

Retorna verdadeiro se a fila estiver vazia.

front (frente)

Retorna o elemento do início sem removê-lo da fila.

rear (traseira)

Retorna o elemento do final sem removê-lo da fila.

Um exemplo de uso de filas em programação é a implementação de uma fila de impressão. Veja o exemplo a seguir:

Suponha que vários documentos sejam enviados para impressão em uma rede de computadores. Cada documento é adicionado à fila de impressão e aguarda sua vez para ser impresso. Quando o primeiro documento da fila é impresso, ele é removido da fila, e o próximo documento na fila passa a ser o primeiro. Dessa forma, os documentos são impressos na ordem em que foram adicionados à fila.



- **Listas encadeadas:** estrutura de dados responsável por armazenar sequências ordenadas de elementos, especialmente quando é necessário inserir ou remover elementos com frequência. No entanto, é importante considerar as vantagens e desvantagens das listas encadeadas em relação às outras estruturas de dados disponíveis, a fim de escolher a melhor opção para cada situação.

As operações básicas com listas encadeadas incluem:

InsertAtEnd (inserirAoFim)	Adiciona um elemento ao final da lista vinculada.
InsertAtHead (inserirAoNício)	Adiciona um elemento no inicio da lista vinculada.
Delete (excluir)	Remove um elemento específico da lista vinculada.
DeleteAtHead (excluirAoNício)	Remove o primeiro elemento da lista vinculada.
Search (busca)	Procura e retorna um elemento específico da lista vinculada.
isEmpty (éVazio)	Verifica se a lista vinculada está vazia ou não, retornando verdadeiro ou falso.

Um exemplo prático de uso de listas encadeadas é a implementação de uma *playlist* de música em uma plataforma *on-line*.

Cada música da *playlist* pode ser representada por um nó da lista encadeada que contém informações, como o título, artista, duração e um ponteiro para a próxima música da *playlist*. Dessa forma, é possível inserir novas músicas na *playlist* e remover outras, sem haver a necessidade de deslocar todos os elementos da lista, como ocorreria em um *array*.



- **Árvore:** é uma estrutura de dados hierárquica e não linear que organiza os dados para facilitar a busca e a classificação. Devido à hierarquia, as árvores permitem que os dados sejam organizados em níveis, cada um representando uma categoria ou subcategoria diferente. Isso torna esse tipo de estrutura uma opção popular para armazenar dados em uma variedade de aplicações, como bancos de dados, sistemas de arquivos e motores de busca. Um exemplo é a árvore que modela os relacionamentos em uma rede social ou uma árvore genealógica.

Em ciência da computação, as árvores são amplamente utilizadas em algoritmos de busca e ordenação, como a **árvore binária de busca**, a **árvore AVL** e a **árvore B**.



Em resumo, as estruturas de dados são uma ferramenta essencial na programação, permitindo a organização eficiente e a manipulação de dados, o que facilita a criação de algoritmos e aplicações mais eficientes e escaláveis.

E aí, conseguiu compreender a importância da estrutura de dados para a lógica de programação? Que tal colocar em prática os seus conhecimentos?

Observe o desafio prático a seguir!



Criando uma lista de compras para uma festa de aniversário

Descrição

O seu grupo possui um grande amigo que vai fazer aniversário em breve e, por isso, vocês vão ajudá-lo a organizar uma festa. Para tanto, façam uma lista do que deve ser comprado e organizem os itens da lista pelo tipo de item. O desafio é considerar que vocês possuem um orçamento fixo de R\$ 500,00.

Objetivos

Utilizar os conceitos de estrutura de dados simples, como listas e tuplas, para criar uma lista de compras para uma festa de aniversário de um amigo de idade entre 15 e 19 anos.

Orientações

- Formem grupos de três a cinco alunos;
- Definam o orçamento total da festa, que é de R\$500,00;
- Cada aluno do grupo deve pesquisar e sugerir, ao menos, um item para a lista de compras, considerando o orçamento total;
- Criem uma lista com os itens sugeridos pelos seus colegas. Ela deverá conter o nome do produto, a quantidade necessária e o preço unitário;
- Organizem a lista de compras de forma clara e objetiva, utilizando uma ferramenta colaborativa, como o Google Docs ou Google Sheets;

- Verifiquem se o orçamento foi respeitado e se todos os itens necessários para a festa foram incluídos;
 - Apresentem a lista de compras final para a classe.
-

1.7 Resolução de problemas práticos

Agora que aprendemos a definição de algoritmos e de estrutura de dados, é necessário compreender como aplicar os conhecimentos para solucionar problemas do mundo real. Confira alguns exemplos a seguir:

1

Planejamento de horários: podemos utilizar algoritmos de otimização para encontrar a melhor solução possível, levando em consideração as restrições e preferências de cada participante.

2

Controle de estoque: é possível utilizar estruturas de dados como listas encadeadas ou árvores para armazenar as informações dos produtos e realizar certas operações, como busca, inserção e remoção de itens, de forma eficiente.

3

Gerenciamento de tarefas: podemos utilizar algoritmos de ordenação para priorizar as atividades com base em alguns critérios, como prazo de entrega, importância ou dependência de outras tarefas.

Planejamento de horários

A fim de entender como colocar em prática o uso de algoritmos e estrutura de dados para solucionar um problema, confira o exemplo que vamos explorar juntos a seguir.

Desafio

Organizar horários de forma eficiente, visando otimizar a produtividade.

Solução

Desenvolvimento de algoritmos que consideram prioridades, bem como disponibilidade e prazos, permitindo a criação de agendas personalizadas.



Maria está organizando um evento e precisa planejar os horários das atividades, mas considerando as restrições e preferências de cada participante. Ela precisa criar um algoritmo que otimize o cronograma do evento, garantindo que todos os participantes possam participar de suas atividades preferidas sem conflitos de horário.

Passo 1: coletar informações

Primeiro, Maria precisa coletar informações sobre as restrições e preferências de cada participante, como horários disponíveis, fuso-horário, localização e função.

Aqui, são definidas as variáveis e seus tipos. Exemplo:

- **var;**
- **participantes**: tipo lista de *strings*;
- **horário_disponíveis**: tipo lista de *strings*;
- **restricoes**: tipo lista de *strings*;
- **preferencias**: tipo lista de *strings*.

Passo 2: definir restrições

Em seguida, Maria precisa definir as restrições para o planejamento de horários.

Passo 3: criar lista

Criar uma lista de participantes com suas respectivas informações e restrições.

Passo 4: gerar horários

Com base nas informações e restrições coletadas, Maria deve gerar uma lista de horários possíveis para as reuniões.

Passo 5: selecionar horário

Selecionar o melhor horário possível, levando em consideração as preferências e restrições de cada participante.

Passo 6: enviar convite

Por fim, Maria deve enviar um convite para todos os participantes com as informações da reunião, incluindo data, hora, local e demais informações.

Esse algoritmo pode ser implementado em uma ferramenta *no-code*, como o Google Sheets ou o Microsoft Excel. Para isso, as planilhas são utilizadas para armazenar as informações dos participantes e os horários possíveis e as fórmulas são usadas para realizar os cálculos e as orientações necessárias. Com isso, Maria poderá organizar as reuniões da empresa de forma eficiente e otimizada, levando em consideração as restrições e as preferências de cada pessoa.

Tabela de horários

Aa Nome	Local	Restrições	Preferências
Maria	Virtual	12:00 às 13:00	09:00 - 12:00
João	Virtual	13:00 ás 14:00	09:00 - 13:00 e 14:00 - 17:00
Pedro	Híbrido	12:00 às 14:00	16:00 ás 17:00
Ana	Híbrido	12:00 ás 13:00	14:00 - 17:00
Carla	Virtual	11:00 ás 13:00	09:00 - 11:00 e 13:00 - 17:00

E aí, gostou da apresentação prática? Com ela, fica mais fácil compreender como aplicar o conteúdo visto em um problema do cotidiano.

Que tal solucionar dois desafios práticos para exercitar o que você viu até aqui?

Confira os dois desafios práticos a seguir.



DESAFIO PRÁTICO

Calculando tempo de viagem e consumo de combustível



Descrição

Você faz parte de um grupo de amigos que está planejando viajar de carro. O que vocês precisam é calcular o tempo do percurso e o consumo de combustível para estimar o custo total da viagem. Para isso, vocês decidem criar um algoritmo simples que considera a distância percorrida, a velocidade média e o consumo médio de combustível do veículo.



Objetivos

- Aplicar os conceitos aprendidos no capítulo em situações práticas;
- Desenvolver habilidades lógicas e de raciocínio.

Orientações

- Escolham uma viagem de um ponto inicial A até um ponto inicial B;
- Definam as variáveis necessárias para armazenar a distância percorrida, a velocidade média, o consumo médio de combustível do veículo e o tempo de viagem;
- Criem uma estrutura de tomada de decisão para verificar se a velocidade média inserida pelo usuário é válida (ou seja, maior que zero);
- Criem um fluxograma para calcular o tempo de viagem baseado na distância percorrida, na velocidade média, no consumo de combustível com base na distância percorrida e no consumo médio de combustível do veículo;

- O grupo pode apresentar utilizando diversas ferramentas ou softwares, Google Docs, planilhas, entre outras.
-



DESAFIO PRÁTICO II

Calculando o índice de massa corporal (IMC)



Descrição

Neste desafio, a equipe deverá utilizar os conceitos de variáveis, tipos de dados, operadores e expressões para criar um algoritmo que calcule o IMC de uma pessoa, que deve ser imaginária e não deve ser baseada em nenhuma da sua turma. Usando o algoritmo, o usuário será capaz de inserir seus dados e receber a classificação do IMC de acordo com a tabela da OMS.



Objetivos

- Aplicar os conceitos aprendidos no capítulo em situações práticas;
 - Desenvolver habilidades lógicas e de raciocínio.
-

Orientações

- Pesquisem sobre o IMC e a tabela de classificação da OMS;
- Criem um fluxograma que represente o algoritmo de cálculo do IMC e a classificação de acordo com a tabela da OMS;
- Definam as variáveis necessárias para o cálculo do IMC, como a altura e o peso;

- Criem uma fórmula para calcular o IMC com base nas variáveis definidas;
 - Utilizem uma estrutura de tomada de decisão para classificar o IMC de acordo com a tabela da OMS;
 - Apresentem utilizando diversas ferramentas ou *softwares*, como o Google Docs, Google Sheets etc.
-

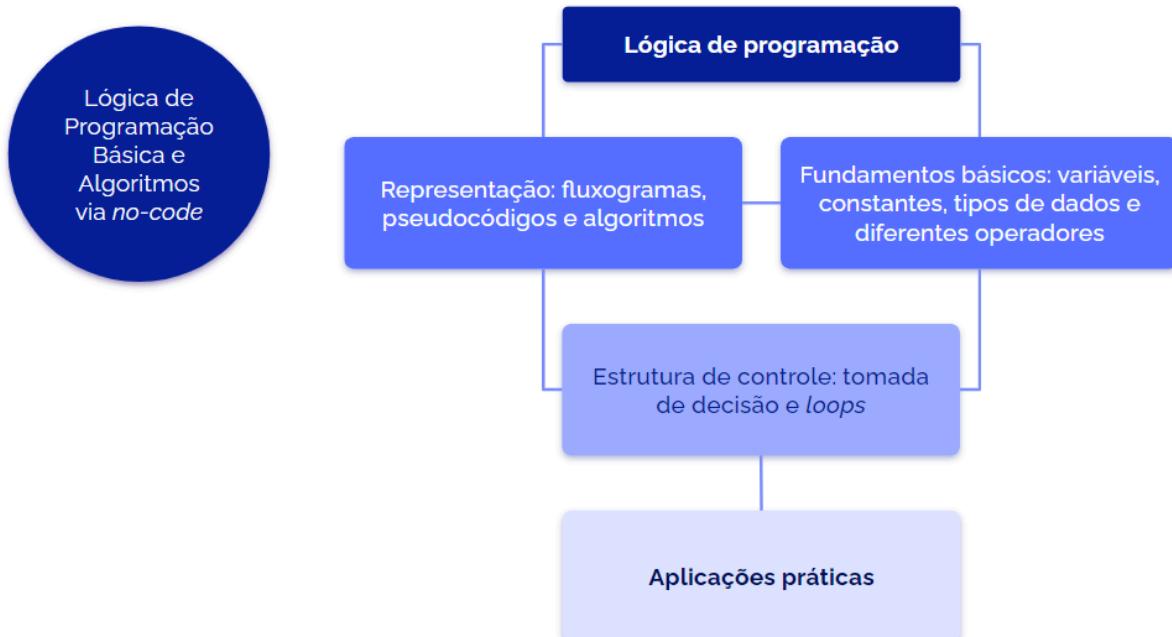


Neste capítulo introdutório sobre a lógica de programação, foi possível entender alguns fundamentos básicos, como variáveis, constantes, tipos de dados, operadores e expressões. Depois, exploramos a representação visual dos algoritmos por meio de fluxogramas, utilizando símbolos e conectores para tornar o fluxo do processo mais compreensível e comunicável.

Em seguida, estudamos a tomada de decisão com o uso de blocos lógicos, como "se-então-senão" e tabelas verdade. Também estudamos os *loops* e as repetições para executar instruções repetidamente até uma condição ser atingida, com "para", "enquanto" e "repita".

Por fim, exploramos estruturas de dados simples, como variáveis e listas, para armazenar e manipular informações e aplicamos o que foi aprendido em problemas práticos, como um planejamento de horários entre diferentes funcionários de uma mesma empresa.

Veja, a seguir, como todos esses conceitos se relacionam:





ATIVIDADE DE FIXAÇÃO

- 1.** O que é lógica de programação? Para o que ela serve?
- 2.** O que é um fluxograma? Construa um que represente o seu dia, desde o momento em que você acorda até o que vai dormir. Além disso, inclua as suas decisões, como "se estiver chovendo, pegue um guarda-chuva".
- 3.** Construa um algoritmo, mas utilizando estruturas de decisão e condições.
- 4.** O que é um *loop*? Cite dois exemplos de *loops* no seu dia a dia.
- 5.** O que é uma variável? Cite os seus tipos e exemplifique.
- 6.** O que é uma constante? Em que a aplicamos?
- 7.** O que é uma lista? Faça uma que seja simples e que contenha cinco itens, indicando o índice de cada um.
- 8.** Como podemos construir um algoritmo para resolver corretamente um problema de gerenciamento de tarefas?
- 9.** Considerando a lógica de programação, como devemos começar a estudar programação? O que devemos aprender para começar a programar?
- 10.** Disserte sobre a sua dificuldade até agora.

Referências

ALBINO, Carla. *Tecnologia ambiental: como a tecnologia ajuda o planeta?*. Ingram Micro Blog, 15 out. 2020. Disponível em: <https://blog.ingrammicro.com.br/tecnologia-e-sustentabilidade/tecnologia-ambiental/>. Acesso em: 1 fev. 2024.

E-TEC BRASIL. *Segurança do trabalho: Aula 5 | Ética e novas tecnologias*. Disponível em: https://proedu.rnp.br/bitstream/handle/123456789/582/Aula_05.pdf?sequence=5&isAllo wed=y. Acesso em: 1 fev. 2024.

GOV.BR. *Ministério da Saúde incorpora vacina contra a dengue no SUS*. Disponível em: <https://www.gov.br/saude/pt-br/assuntos/noticias/2023/dezembro/ministerio-da-saud e-incorpora-vacina-contra-a-dengue-no-sus>. Acesso em: 1 fev. 2024.

PRESIDÊNCIA DA REPÚBLICA. *Lei nº 13.709, de 14 de agosto de 2018*. Disponível em: https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.html. Acesso em: 1 fev. 2024.

CETIC.BR. *TIC domicílios*. [S.d.]. Disponível em: <https://cetic.br/pt/pesquisa/domiciliostic/>. Acesso em: 1 fev. 2024.

DOMINGUES, Ivan. *Ética, ciência e tecnologia*. Kriterion, Belo Horizonte, v. 45, n. 109, jun. 2004. Disponível em: <https://www.scielo.br/j/kr/a/3TrN3nmtqxkmwp3BZ588snH/>. Acesso em: 1 fev. 2024.

ELIAS, Adonai; LEÃO, Luiza. *State of Search Brasil 4: a influência da IA no comportamento de busca e a percepção dos brasileiros*. Hedgehog, 7 dez. 2023. Disponível em: <https://br.hedgehogdigital.co.uk/blog/state-of-search-brasil-4/>. Acesso em: 1 fev. 2024.

ENGEL, Vonia; AREND, Silvio C. *A inovação tecnológica no contexto do desenvolvimento regional endógeno*. VI Seminário Internacional sobre desenvolvimento regional, Rio Grande do Sul, 4-6 set. 2013. Disponível em: <https://www.unisc.br/site/sidr/2013/Textos/302.pdf>. Acesso em: 1 fev. 2024.

FRAISSAT, Nilceia. *Tecnologia e meio ambiente*: impactos, importância e pontos positivos. Mundo Inovação, 5 jun. 2023. Disponível em: <https://mundoinovacao.com.br/tecnologia-e-meio-ambiente/>. Acesso em: 1 fev. 2024.

HAYNE, Luiz A.; WYSE, Angela T. *Análise da evolução da tecnologia*: uma contribuição para o ensino da ciência e tecnologia. Revista Brasileira de Ensino de Ciência e Tecnologia, Curitiba, v. 11, n. 3, 2018. Disponível em: <https://periodicos.utfpr.edu.br/rbect/article/view/5947>. Acesso em: 1 fev. 2024.

INEP. *Censo da educação básica | 2020*. Brasília: Ministério da Educação, 2021. Disponível em: https://download.inep.gov.br/publicacoes/institucionais/estatisticas_e_indicadores/notas_estatisticas_censo_escolar_2020.pdf. Acesso em: 1 fev. 2024.

LARAIA, Roque de Barros. *Cultura*: um conceito antropológico. 14. ed. Rio de Janeiro: Zahar, 2001.

PRESIDÊNCIA DA REPÚBLICA. *Lei nº 10.973, de 2 de dezembro de 2004*. Disponível em: <https://encurtador.com.br/dtPR7>. Acesso em: 1 fev. 2024.

LÉVY, Pierre. *Cibercultura*. 2. ed. Rio de Janeiro: Editora 34, 2000.

LORENZETTI, J.; TRINDADE, L. L.; PIRES, D. E. P.; RAMOS, F. R. S. *Tecnologia, inovação tecnológica e saúde*: uma reflexão necessária. Texto & contexto – enferm, v. 21, n. 2, jun. 2012. Disponível em: <https://www.scielo.br/j/tce/a/63hZ64xJVrMf5fwkBh7dnnq/?lang=pt>. Acesso em: 1 fev. 2024.

FINEP. *Manual de Oslo*: proposta e diretrizes para coleta e interpretação de dados sobre Inovação Tecnológica. 3. ed. 2006. Disponível em: <https://encurtador.com.br/epqxQ>. Acesso em: 1 fev. 2024.

MEDEIROS, Z.; VENTURA, P. C. S. *O conceito Cultura Tecnológica e um estudo no meio educacional*. Rev. Ensaio, v. 09, n. 02, jul.-dez. 2007. Disponível em: <https://www.scielo.br/j/epec/a/m6bqLmLjNwP3SX5KCG8p6NQ/?format=pdf>. Acesso em: 1 fev. 2024.

NEGRI, Paulo. *Cultura e tecnologia: estabelecendo uma rede*. Curitiba: UFPR, 2008. Disponível em: https://www.academia.edu/4309337/CULTURA_E_TECNOLOGIA_ESTABELECENDO_UM_A_REDE. Acesso em: 1 fev. 2024.

DE PAULA, Julia; PESSOA, Luan K.; NEVES, João E. D. A. *Os impactos da tecnologia na educação*. RBTI, Campinas, v. 5, n. 1, 2023. Disponível em: <https://encurtador.com.br/kHMV4>. Acesso em: 1 fev. 2024.

STAHLER, Gabriela. *Ranking de vazamento de dados: Brasil é o 12º colocado*. Privacy Tech, 26 abr. 2022. Disponível em: <https://privacytech.com.br/vazamentos/ranking-de-vazamento-de-dados-brasil-e-o-12-colocado.413580.jhtml>. Acesso em: 1 fev. 2024.

REIS, Dálcio R. *Gestão da inovação tecnológica*. São Paulo: Manole, 2004.

RODRIGUES, P. dos Santos, et al (orgs.). *A universidade e a pesquisa: o público e o privado*. UFRJ/ICB. Sonda; IURME/CASPAM. Rio de Janeiro, 1997.

SANTOS, Zélia Maria de Sousa Araújo et al. *Tecnologias em saúde: da abordagem teórica à construção e aplicação no cenário do cuidado*. Fortaleza: EdUECE, 2016. Disponível em: <https://encurtador.com.br/zz247>. Acesso em: 1 fev. 2024.

SARAIVA EDUCAÇÃO. *Tecnologia e meio ambiente: como as inovações podem ajudar o planeta?*. Saraiva Educação, 2 jun. 2023. Disponível em: <https://conteudo.saraivaeducacao.com.br/meio-ambiente/tecnologia-e-meio-ambiente/>. Acesso em: 1 fev. 2024.

SCHUMPETER, J. Alois. *História da análise econômica*. Rio de Janeiro: Fundo de Cultura, 1964.

SERASA EXPERIAN. *O que é cidadania digital? Fique por dentro desse conceito*. 7 dez. 2022. Disponível em: <https://serasa.certificadodigital.com.br/blog/e-conectividade-social/cidadania-digital/>. Acesso em: 1 fev. 2024.

TOTVS. *Cidadania digital*: o que é, elementos, desafios e mais!. 17 maio 2023. Disponível em: <https://encurtador.com.br/jvPW2>. Acesso em: 1 fev. 2024.

TRAUTMANN, Dagmar Aparecida. *Educação, ética e tecnologia*: impressões e reflexões. 2002. Dissertação (Mestrado). Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis. Disponível em: <https://repositorio.ufsc.br/xmlui/bitstream/handle/123456789/84016/188872.pdf?sequence=1&isAllowed=y>. Acesso em: 1 fev. 2024.

DA SILVA, Eduardo A. *Ciberespaço e cibercultura*: definições e realidades virtuais inseridas na práxis do homem moderno. Só pedagogia, 15 abr. 2014. Disponível em: http://www.pedagogia.com.br/artigos/ciberespaco_cibercultura/index.php?pagina=0. Acesso em: 1 fev. 2024.