# Lógica e Pensamento Computacional

*Tércio Cavalcanti da Silva Santos*

Técnico em

**Desenvolvimento de Sistemas**

Educação a Distância

# *Lógica e Pensamento Computacional*

*Tércio Cavalcanti da Silva Santos*

## Curso Técnico em Desenvolvimento de Sistemas

Escola Técnica Estadual Professor Antônio Carlos Gomes da Costa

Educação a Distância

Recife

1.ed. | Agosto 2023

Licença Pública Creative Commons
Atribuição-NãoComercial-Compartilhalgual 4.0 Internacional

**Professor Autor**
Tércio Cavalcanti da Silva Santos

**Revisão**
Tércio Cavalcanti da Silva Santos

**Coordenação de Curso**
José Américo Teixeira de Barros

**Coordenação Design Educacional**
Deisiane Gomes Bazante

**Design Educacional**
Ana Cristina do Amaral e Silva Jaeger
Helisangela Maria Andrade Ferreira
Jailson Miranda
Roberto de Freitas Morais Sobrinho

**Diagramação**
Jailson Miranda

**Catalogação e Normalização**
Hugo Cavalcanti (Crb-4 2129)

**Coordenação Executiva (Secretaria de Educação e Esportes de Pernambuco | SEE)**
Ana Cristina Cerqueira Dias
Ana Pernambuco de Souza

**Coordenação Geral (ETEPAC)**
Arnaldo Luiz da Silva Junior
Gustavo Henrique Tavares
Maria do Rosário Costa Cordouro de Vasconcelos
Paulo Euzébio Bispo

**Secretaria Executiva de Educação Integral e Profissional**

**Escola Técnica Estadual Professor Antônio Carlos Gomes da Costa**

**Gerência de Educação a distância**

# Sumário

## Introdução

Olá estudante, seja bem-vindo(a) a disciplina de lógica e pensamento computacional, essas duas habilidades são fundamentais para você que deseja se tornar um programador(a) de sucesso ou mesmo para entender melhor o funcionamento dos computadores e sistemas. Essas habilidades estão relacionadas à forma como você pensa, organiza e resolve problemas. A lógica de programação é a capacidade de pensar de forma lógica para solucionar problemas. É uma habilidade essencial, pois permite a criação de algoritmos e estruturas de dados que possam ser executados pelos computadores.

Já o pensamento computacional é uma habilidade que envolve a capacidade de resolver problemas complexos, não se assuste com a complexidade de alguns projetos, você vai aprender a dividir os problemas em partes menores e mais simples, você vai identificar padrões e utilizar algoritmos para encontrar as soluções mais adequadas. É muito importante que você estudante desenvolva essa habilidade para se adaptar ao mundo digital e compreender como as tecnologias funcionam.

Além disso, **é essencial que você pratique, erre e tente novamente, não desista, assim você aprenderá a resolver problemas diferentes em diversas áreas**. A prática vai ajudar a consolidar o conhecimento que você está adquirindo como também a desenvolver habilidades de solução de problemas, além de estimular a sua criatividade e a sua curiosidade.

A disciplina de lógica e pensamento computacional está dividida em quatro Unidades como descrito abaixo:

Na Unidade 1 você vai aprender a reconhecer os princípios de lógica de programação algorítmica, esses princípios irão ajudar a construir a base necessária para você avançar no campo do desenvolvimento de sistemas. Em seguida, na Unidade 2 você vai aprender a desenvolver um algoritmo para resolução de um problema, você verá na prática como aplicar um conjunto de regras seguindo uma sequência lógica de passos para resolver um problema específico.

Já na Unidade 3 você passará a reconhecer o que é pensamento computacional e como o processo de reconhecer aspectos da computação no seu ambiente irá ajudá-lo a entender e raciocinar sobre como os computadores e sistemas funcionam. Por fim, na Unidade 4, você vai passar a identificar como desenvolver o pensamento computacional elaborando, resolvendo problemas e apresentando soluções de forma que possam ser executadas pelos computadores.

É isso estudante, espero que você esteja animado(a) para essa grande aventura. Gostaria de lembrar que esse é o primeiro passo e ele é muito importante para que você forme uma base consistente e possa aplicar o conhecimento que adquiriu utilizando qualquer linguagem de programação e/ou ambiente computacional. **Estaremos juntos e você tem um grande desafio e uma meta, chegar juntamente com os outros estudantes no final da Unidade 4, não desista,** quando surgir alguma questão estaremos a sua disposição nos fóruns para tirar dúvidas e esclarecer qualquer ponto. Seja bem-vindo(a) mais uma vez! Aperte o cinto e vamos lá!

## As Ferramentas

Para começar nossa aventura vou apresentar algumas ferramentas que iremos utilizar nesta disciplina. A principal ferramenta que iremos utilizar é o Portugol Studio, desenvolvida pelo LITE - laboratório de inovação tecnológica na educação da Universidade do Vale do Itajaí (Univali).

Portugol foi desenvolvida com o objetivo de facilitar o processo de aprendizado de programação. O nome "Portugol" vem da junção das palavras "Português" e "Algoritmo", já que essa linguagem utiliza palavras em português para descrever as operações e comandos de programação.

O Portugol é uma linguagem de programação estruturada, ou seja, utiliza sequências de instruções lógicas para resolver problemas. É possível desenvolver algoritmos completos com esta linguagem. Uma das principais vantagens do Portugol é sua simplicidade, que permite a você estudante aprender os conceitos fundamentais da programação sem se preocupar com detalhes complexos. Além disso, como a linguagem é em português, será mais fácil entender os comandos e estruturas utilizadas.

É importante destacar que o Portugol não é uma linguagem de programação utilizada no mercado de trabalho. Ela é utilizada exclusivamente para fins educacionais, para que você aprenda os conceitos e estruturas básicas da programação e possa evoluir para outras linguagens mais complexas.

A versão da ferramenta adotada para os exemplos e videoaulas nesta disciplina é a desktop, ou seja, a instalada no computador e que pode ser utilizada sem internet (nada impede que você utilize a versão para a Web ou a versão para dispositivos móveis). A figura a seguir apresenta a tela inicial da interface do Portugol Studio.

Acesse: http://lite.acad.univali.br/portugol/
E faça o download do Portugol Studio e instale o programa em seu computador.

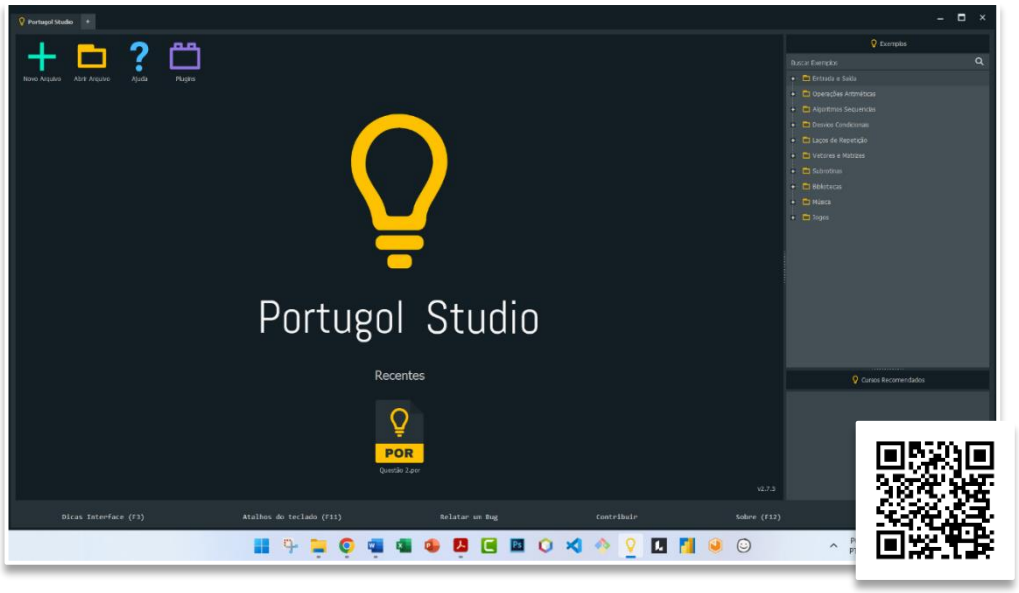



**Figura 1 - Tela inicial do Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol
**Descrição da figura:** Tela inicial do Portugol Studio com os botões de Novo Arquivo, Abrir Arquivo, Ajuda, Plugins, Dicas Interface (F3), Atalhos do Teclado (F11), Relatar um Bug, Contribuir e Sobre (F12), além de uma barra lateral de exemplos.

Além da figura anterior, as figuras a seguir apresentam as bibliotecas fornecidas pela ferramenta.


**Figura 2 - Exemplos oferecidos pelo Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** Barra lateral de exemplos oferecidos pela ferramenta.

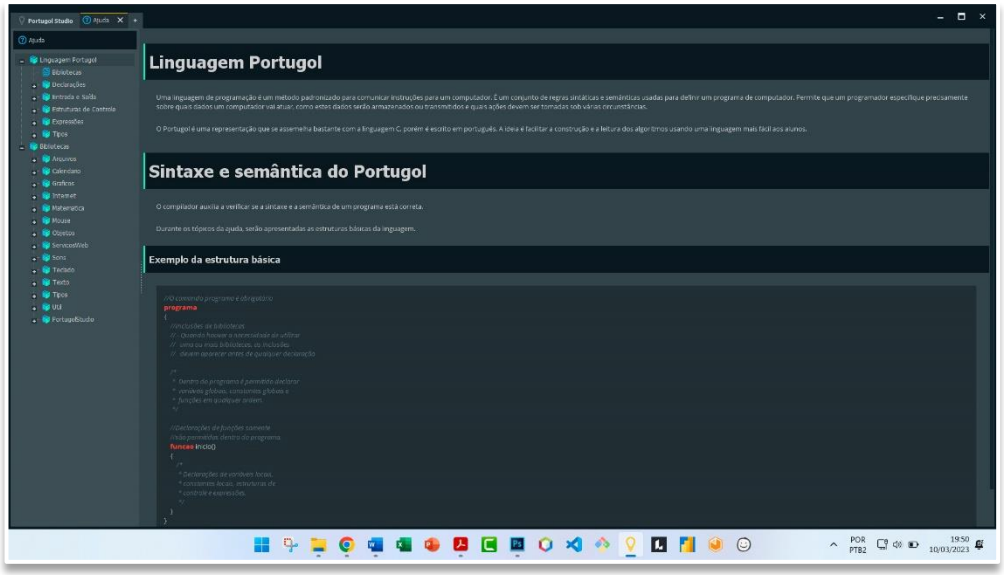



**Figura 3 - Biblioteca de funções fornecida pelo Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** Aba de Ajuda oferecida pela ferramenta com barra lateral contendo particularidades da linguagem e bibliotecas, no lado direito é mostrado o conteúdo do item selecionado.

ASSISTA O VÍDEO PARA AMPLIAR O CONCEITO DE PORTUGOL STUDIO.
https://www.youtube.com/watch?v=6OIADpFImtc

Existe a versão para a Web do Portugol Studio (executada no navegador) e outra para dispositivos móveis (celulares e tablets). A versão Web é mostrada na figura abaixo.

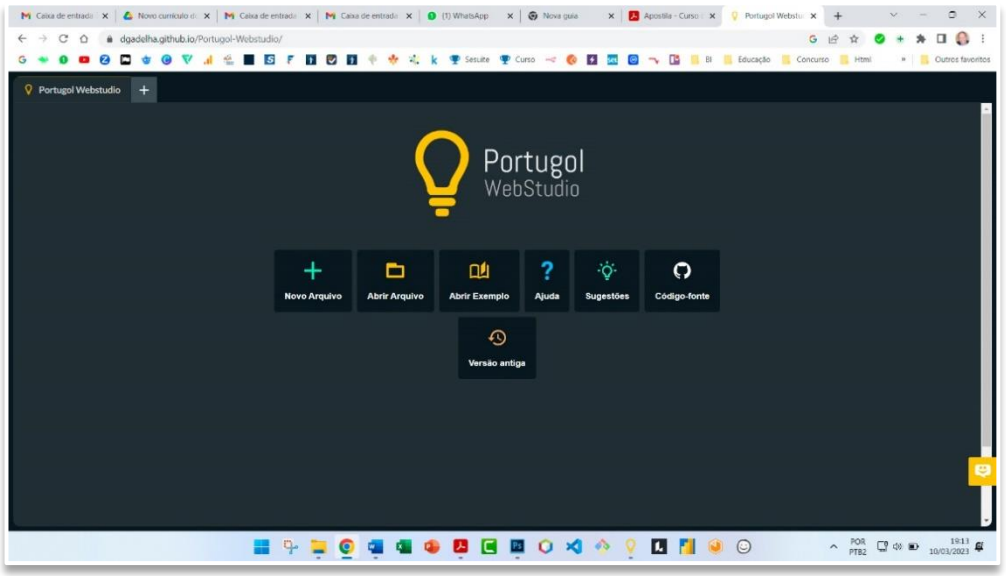



**Figura 4 - Interface do Portugol Studio para Web.**
**Fonte:** https://portugol-webstudio.cubos.io/ide
**Descrição da figura:** Interface do Portugol Studio para a Web.

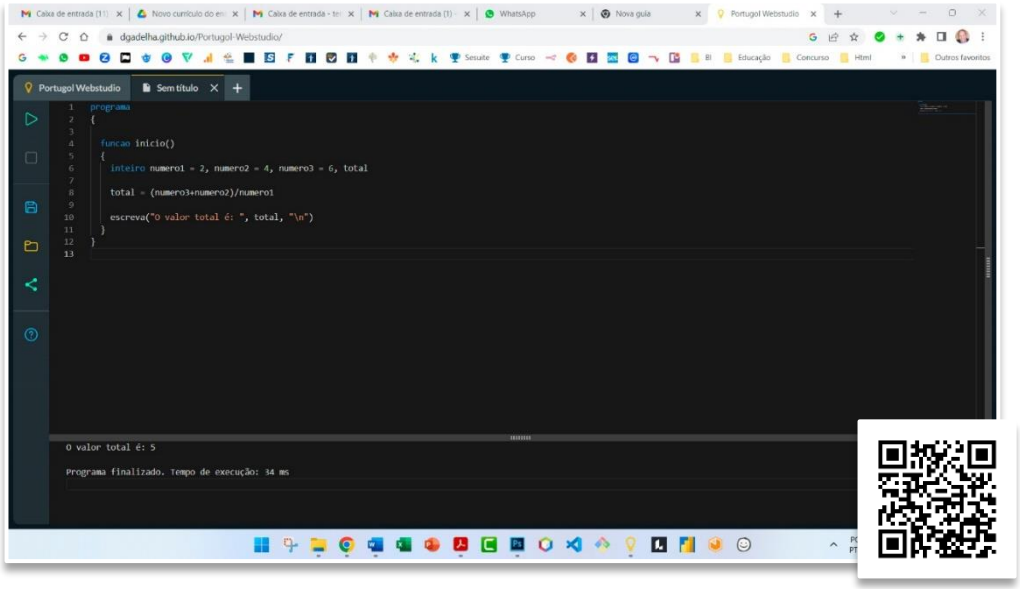



**Figura 5 - Algoritmo executado no Portugol Studio para Web.**
**Fonte:** https://portugol-webstudio.cubos.io/ide
**Descrição da figura:** Tela do Portugol Studio para a Web. A tela apresenta um algoritmo simples sendo executado e o resultado mostrado na barra inferior.

**PARA CONHECER O PORTUGOL STUDIO PARA A WEB ACESSE:**
https://portugol-webstudio.cubos.io/ide

**CONFIRA TAMBÉM O PORTUGOL STUDIO PARA DISPOSITIVOS MÓVEIS, ACESSE:**
https://play.google.com/store/apps/details?id=br.erickweil.portugolweb&hl=p

Existem outras ferramentas além do Portugol Studio para criação e execução de algoritmos, uma das mais usadas é o Visualg, um software livre de programação em português que permite ao usuário criar e executar algoritmos de forma simples e intuitiva. O conceito do Visualg está relacionado à sua facilidade de uso e sua abordagem didática para ensinar programação.

O Visualg oferece uma interface gráfica amigável, que inclui recursos como editor de código, depurador, ferramentas de entrada e saída, além de recursos para ajudar os usuários a entender a estrutura básica dos algoritmos.

Uma das principais características do Visualg é o seu ambiente de aprendizado, que permite que os usuários aprendam a programar por meio de exemplos práticos e exercícios.

O software também possui uma ampla biblioteca de exemplos e modelos, tornando-o uma ferramenta muito boa para o desenvolvimento de algoritmos.

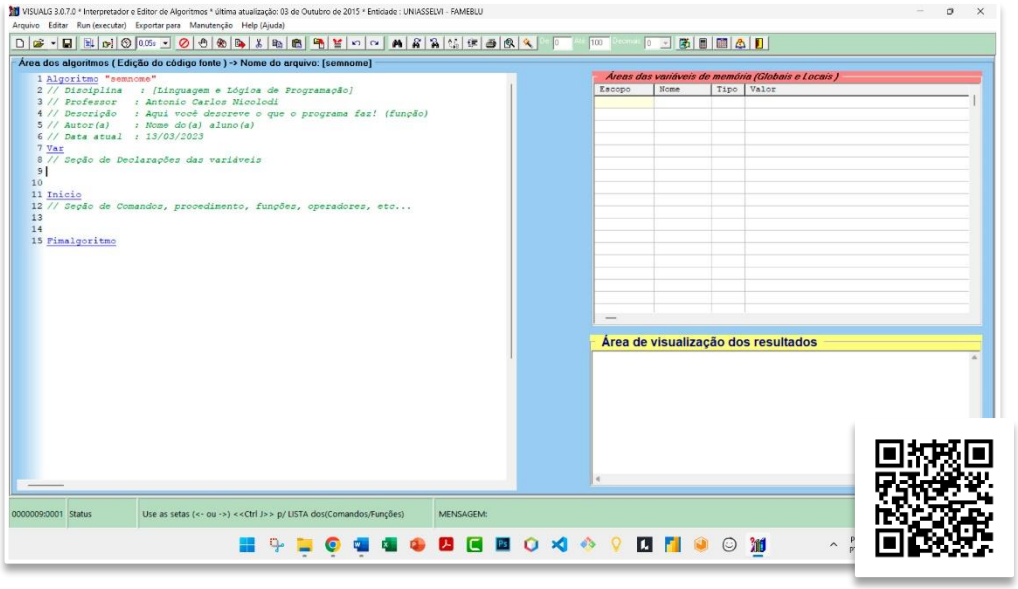



**Figura 6 - Interface do VisuAlg.**
**Fonte:** https://visualg3.com.br/baixe-o-visualg-3-0-7
**Descrição da figura:** Interface do VisuAlg.

Além das ferramentas para execução e criação de algoritmos, existem outras ferramentas que são muito importantes para o desenvolvimento do raciocínio lógico no decorrer da disciplina, segue abaixo alguns exemplos.

**PARA CONHECER O LUCIDCHART ACESSE:**
https://www.lucidchart.com/pages/pt

**PARA CONHECER O CANVA ACESSE:**
https://www.canva.com/pt_br/graficos/fluxograma/

# Unidade 01 | Reconhecer princípios de lógica de programação algorítmica

## 1.1 Sequência Lógica

Você sabia que sequência lógica, nada mais é do que executar as instruções em ordem para chegar no resultado que queremos? É importante que você pense nessa sequência de forma organizada e clara para garantir que tudo dê certo.

Começamos definindo o problema e as etapas necessárias para resolvê-lo. Daí em diante, traduzimos cada passo em instruções que serão executadas uma após a outra. É importante que essas instruções sejam objetivas e claras para não bagunçar tudo. Ah, e precisamos usar instruções de controle de fluxo, como if's e loop's, para garantir que o programa funcione bem. Vamos aprender mais sobre isso depois.

## 1.2 Instrução

Nesse tópico você vai aprender que instrução é uma sequência de símbolos e sinais transformados em código para que o computador realize uma operação. Cada instrução consiste em campos que indicam o tipo de operação, os operandos apropriados e outras informações relevantes para a execução da operação.

As instruções são a base de trabalho de um computador e são interpretadas e executadas sequencialmente, uma a uma para executar tarefas complexas a partir de operações simples. As instruções podem ser criadas por programadores ou geradas automaticamente por outras ferramentas.

## 1.3 Algoritmo

Algoritmo é um passo a passo de instruções para realizar uma tarefa específica, ele pode ser escrito em várias linguagens de programação e é muito útil para achar a solução de algum problema. Mas para que um algoritmo funcione bem, você tem que seguir algumas etapas:

1. Preste atenção na ordem lógica de execução de cada passo;
2. O algoritmo deve ter início e fim;

3. Deve ser claro e completo;
4. Precisa ser escrito com riqueza de detalhes;
5. Cada tarefa será uma instrução para o computador e precisa ser bem clara.

## 1.4 Fases de um Algoritmo

As fases de um algoritmo são os passos que precisamos seguir para resolver um problema ou realizar uma tarefa. Elas incluem:

- **Entrada:** Nessa fase, o algoritmo recebe as informações ou dados que precisa para fazer a tarefa. Se você está usando um computador, pode digitar essas informações no teclado ou em algum outro dispositivo.
- **Processamento:** Nessa fase, o algoritmo faz as operações necessárias para resolver o problema ou realizar a tarefa. Isso pode incluir cálculos matemáticos, manipulação de dados, execução de comandos ou comunicação com outros sistemas.
- **Saída:** Depois que termina de processar, o algoritmo mostra um resultado ou uma saída para o usuário ver. Por exemplo, você pode ver essas informações na tela do computador ou em algum outro dispositivo.

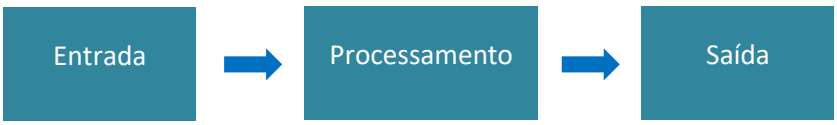



**Figura 7 - Fases de um algoritmo.**
**Fonte:** O autor
**Descrição da figura:** Sequência conectada, entrada, processamento e saída.

## 1.5 Desenvolvimento Estruturado

Essa é uma forma de programar que ajuda a organizar e dividir o código em partes menores, para que fique mais fácil de entender e manter. Além disso, você vai aprender que essa forma de programação divide o trabalho em fases bem definidas para evitar erros. Para fazer isso, você pode usar algumas técnicas de programação, como dito anteriormente, podemos dividir o código em partes menores e usar estruturas como loop's e if's.

## 1.6 Programas

Um programa de computador é um conjunto de instruções escritas em linguagem de programação, como C, Java ou Python por exemplo, que o computador usa para executar uma tarefa específica. As instruções são organizadas em uma sequência lógica, passo a passo, o computador segue essa sequência para executá-las sozinho.

Os programas de computador podem ser usados para muitas coisas diferentes, e o mais legal é que eles podem ser mudados e atualizados para resolver problemas ainda maiores.

## 1.7 Formas de Representação de Algoritmos

Você sabia que existem diferentes formas de apresentar um algoritmo? Elas são chamadas de representações. Essas representações ajudam você a expressar de maneira clara, a lógica e o passo a passo de um algoritmo, para que outras pessoas possam entender.

Algumas dessas formas incluem descrições narrativas, fluxogramas, pseudocódigos e outras. Cada forma tem suas próprias características, por isso é importante que você conheça várias formas para escolher a melhor.

### 1.7.1 Descrição Narrativa

Essa forma de representação explica o problema a ser resolvido usando palavras em um idioma que as pessoas entendem, como português, inglês, espanhol e outros.

- **Vantagem:** Não é necessário aprender nenhum conceito novo, pois a língua natural, o português no nosso caso, já é conhecida.
- **Desvantagem:** A língua natural pode ter diferentes interpretações e possíveis indecisões, o que dificulta a passagem do algoritmo para uma linguagem de programação.

Veja abaixo dois exemplos de algoritmos representados por meio da descrição narrativa.

Descrição narrativa da troca de uma lâmpada.

- Desligue a energia da lâmpada que será trocada.

- Suba a escada com segurança até alcançar a lâmpada queimada.
- Se a lâmpada estiver quente, aguarde alguns minutos até esfriar.
- Remova a lâmpada queimada do bocal e descarte em local seguro.
- Pegue a lâmpada nova, coloque no bocal rosqueando até que fique firme.
- Desça da escada com segurança.
- Ligue a energia novamente e teste a nova lâmpada.

Descrição narrativa para calcular a média final, sabendo-se que o estudante fez três trabalhos e uma prova, para aprovação a nota deve ser maior ou igual a sete.

- Some as notas dos três trabalhos com a nota da prova.
- Divida o resultado dessa soma por quatro.
- Se a nota for maior ou igual a sete, o estudante foi aprovado, senão, o estudante foi reprovado.

### 1.7.2 Fluxograma

O fluxograma é uma representação gráfica que mostra o passo a passo das atividades de um sistema. Ele é uma ferramenta visual que ajuda a entender as etapas de forma clara e objetiva, para encontrar problemas e melhorar a solução.

O fluxograma usa símbolos gráficos para representar cada passo e setas para mostrar a direção da atividade. Cada símbolo tem um significado e representa uma ação específica. Confira abaixo o significado de cada símbolo e a ação que ele representa:

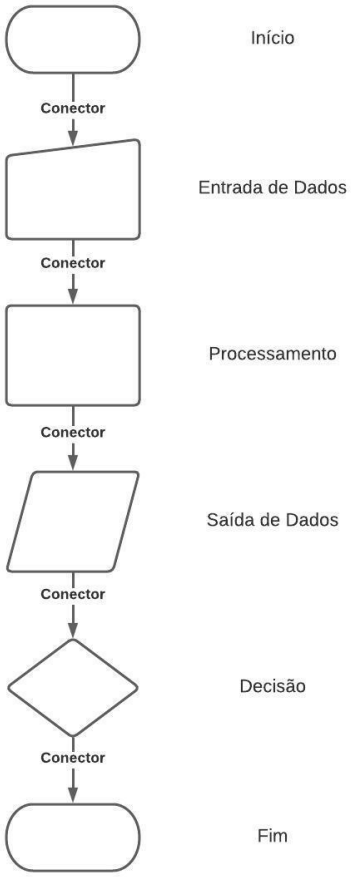



**Figura 8 - Símbolos que representam uma ação em um fluxograma.**
**Fonte:** O autor
**Descrição da figura:** Imagem com os símbolos e sua descrição. Início, um retângulo com as arestas abauladas. Conector, seta indicando o fluxo da atividade. Entrada de dados, um retângulo com a aresta superior esquerda rebaixada. Processamento, um retângulo. Saída de dados, uma fita de papel. Decisão, um losango. Fim, um retângulo com as arestas abauladas.

- **Vantagem**: Os símbolos gráficos são mais fáceis de compreender.
- **Desvantagem**: É preciso conhecer os símbolos dos fluxogramas. A falta de detalhes também pode comprometer o algoritmo e a passagem para uma linguagem.

A figura abaixo mostra um fluxo simples com uma decisão onde são usadas as palavras “Leia” e “Mostra” para representar as ações de entrada e saída.

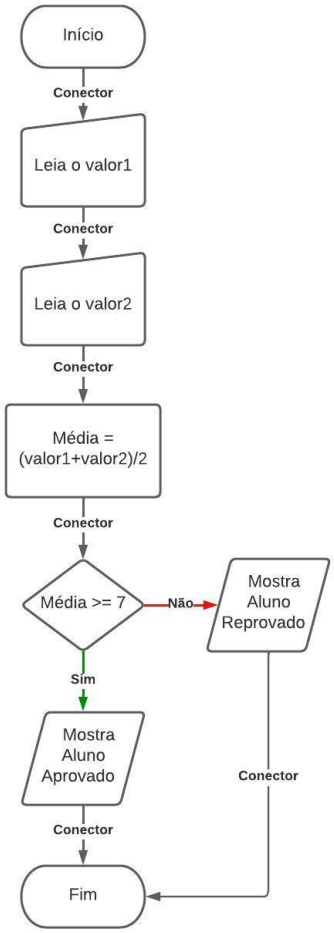



**Figura 9 - Símbolos que representam uma ação em um fluxograma.**
**Fonte:** O autor
**Descrição da figura:** Fluxograma com um fluxo simples, com os passos Início, Leia valor1, Leia valor2, Média é igual a valor1 mais valor2 dividido por 2, pergunta condicional, se a Média for maior ou igual a 7 mostra **Aluno Aprovado** e Fim, caso a média seja menor que 7 mostra **Aluno Reprovado** e Fim.

ASSISTA O VÍDEO ABAIXO PARA APROFUNDAR O SEU CONHECIMENTO
https://www.youtube.com/watch?v=jbRzQVzH7ss

No exemplo abaixo foi utilizada uma estrutura condicional, senha é igual a 123456, “se sim” mostra acesso permitido, “se não” mostra senha incorreta e retorna ao primeiro passo pedindo que a senha seja digitada novamente.

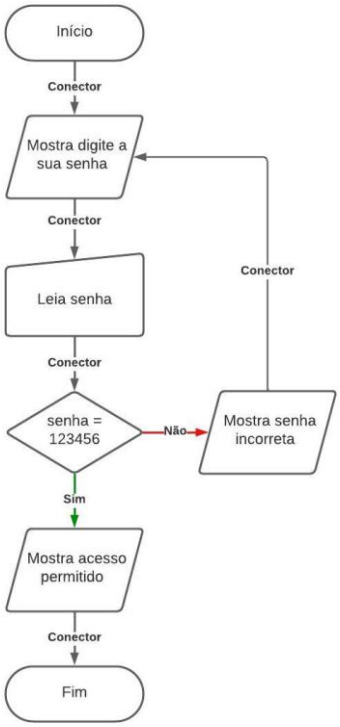



**Figura 10 - Fluxograma bifurcado com repetição.**
**Fonte:** O autor
**Descrição da figura:** Fluxograma bifurcado com repetição onde os passos são: Início, Mostra “Digite a sua senha.”, Leia senha, então vem a condição “SE” a senha é igual a “123456”, se sim Mostra “Acesso permitido” e vai para o passo Fim, se não Mostra “Senha incorreta” e retorna ao passo Mostra “Digite a sua senha”.

**OLHO NESSA DICA!**
Existem várias ferramentas para ajudar na construção de fluxogramas, escolha a que for mais confortável para o seu aprendizado.

**PARA CONHECER O LUCIDCHART ACESSE:**
https://www.lucidchart.com/pages/pt

**PARA CONHECER O CANVA ACESSE:**
https://www.canva.com/pt_br/graficos/fluxograma/

### 1.7.3 Pseudocódigo

Pseudocódigo é uma linguagem de programação informal utilizada para descrever a lógica e o passo a passo de um algoritmo, sem se preocupar com os detalhes de uma linguagem de programação real.

É uma ferramenta muito legal para planejar soluções para um problema antes de começar a escrever o código real numa linguagem de programação. Com o Pseudocódigo, dá para mostrar o passo a passo de um algoritmo de um jeito fácil de entender, e ele sempre pode ser melhorado para facilitar o entendimento e a resolução do problema.

- **Vantagem:** A passagem do algoritmo para uma linguagem de programação é praticamente imediata.
- **Desvantagem:** É necessário aprender as regras do pseudocódigo que serão apresentadas nos exemplos a seguir.

Nos países onde o idioma é o português, o pseudocódigo é conhecido como Portugol. Ao longo deste ebook teremos todos os exemplos de pseudocódigo apresentados em Portugol.

## 1.8 Comandos de Entrada e Saída

Os comandos de entrada e saída são aqueles que permitem que você se comunique com um programa de computador usando dispositivos como teclados, mouses, telas sensíveis ao toque, impressoras e outros.

Quando você usa comandos de entrada, você fornece dados ou informações para o programa. Quando você usa comandos de saída, o programa exibe informações para você ou envia informações para outros dispositivos.

### 1.8.1 Comando de entrada de dados (leia)

O comando “leia” recebe os valores digitados por você passando esses valores para variáveis que são passadas como parâmetro. Não se preocupe, vamos ver isso detalhadamente por meio de exemplos.

### 1.8.2 Comando de saída de dados (escreva)

O conteúdo das variáveis é impresso usando o comando “escreva”, ele é enviado para os dispositivos de saída (normalmente a tela do computador).

No Portugol e em outras linguagens de programação as variáveis devem ser separadas por vírgula e às vezes também por vírgulas combinadas com aspas, veja o exemplo abaixo.

```
programa
{
	funcao inicio()
	{
		cadeia nome

		escreva("Digite o seu nome: ")
		leia(nome)
		escreva("\n", "O nome digitado foi: ", nome, "\n")
	}
}
```

**Quadro 1 - Código com exemplo de entrada e saída de dados no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada “inicio” e nela exibe uma mensagem com o comando “escreva”, solicitando que o usuário digite o seu nome, o comando "leia", lê o valor fornecido pelo usuário, e por fim exibe o conteúdo da variável “nome” que armazenou o nome digitado pelo usuário.

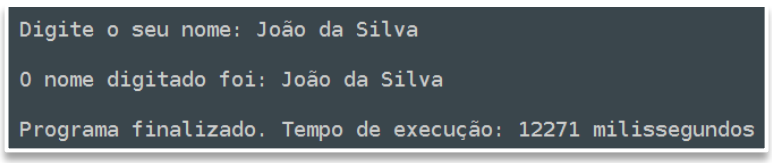



**Figura 11 - Exemplo de entrada e saída de dados no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** Exibição da saída do programa de exemplo de entrada e saída de dados, apresentando o nome digitado, no caso é exibida a frase “O nome digitado foi: João da Silva”.

**OLHO NESSA DICA!**

A indentação ou recuo (do inglês, indentation) é um recurso para organizar a estrutura de um algoritmo aumentando sua legibilidade. Adote essa boa prática em todos os seus códigos.

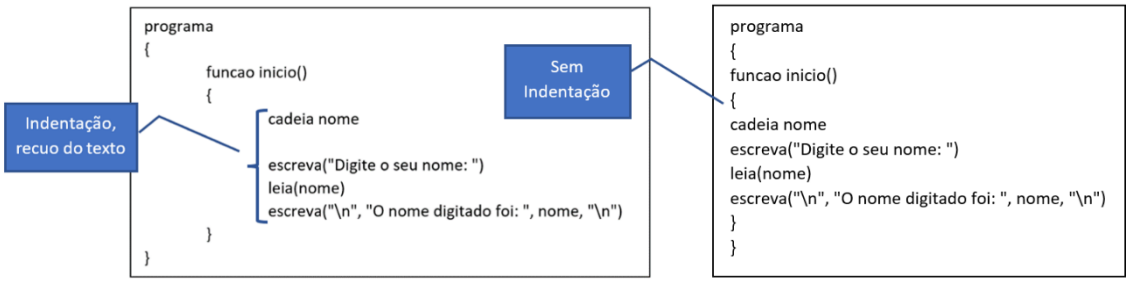



**Figura 12 - Exemplo de indentação no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Indentação de um programa no Portugol Studio.

### 1.8.3 Representação e Armazenamento de Dados

As variáveis e constantes são as formas de representação e armazenamento dos dados. Segue abaixo um exemplo de representação e armazenamento de dados.

```
programa
{
        funcao inicio()
        {
                real media, nota1, nota2

                escreva("Digite a primeira nota: ")
                leia(nota1)

                escreva("\n", "Digite a segunda nota: ")
                leia(nota2)

                media = (nota1+nota2)/2

                escreva("\n", "A Média é: ", media, "\n")
        }
}
```

**Quadro 2 - Código com exemplo de uso de variáveis no Portugol Studio.**

**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função “inicio” e nela declara três variáveis do tipo real, “media”, “nota1” e “nota2”, lê as duas variáveis “nota1” e “nota2” com o comando “leia”, e atribui a variável “media” o valor da soma das duas variáveis dividido por dois, depois apresenta na tela, com o comando “escreva”, o valor calculado da variável “media”.

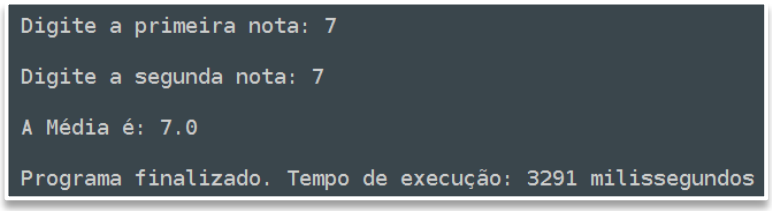



**Figura 13 - Saída do exemplo de uso de variáveis no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do quadro 2 de representação e armazenamento de dados. O resultado exibido é a frase “A Média é: 7.0”.

## 1.9 Variáveis, Constantes e Tipos de Dados

### 1.9.1 Variáveis

Você sabia que as variáveis guardam valores que podem ser modificados ao longo da execução do programa? Elas são identificadas por um nome e podem guardar diferentes tipos de dados, como números, texto, valores booleanos etc.

### 1.9.2 Declaração de Variáveis

Quando você cria uma variável, basicamente está dando um nome e escolhendo o tipo de informação que ela vai guardar. O objetivo é reservar um lugar na memória do computador para que o programa possa usar essa variável quando precisar.

É importante lembrar de criar a variável antes de usá-la no código, para garantir que o nome e o tipo estejam certos e que tenha espaço suficiente na memória para guardar as informações.

```
programa
{
	funcao inicio()
	{
		real soma, numero1, numero2

		escreva("Digite o primeiro número: ")
		leia(numero1)

		escreva("\n", "Digite o segundo número: ")
		leia(numero2)

		soma = numero1+numero2

		escreva("\n", "A Soma é: ", soma, "\n")
	}
}
```

**Quadros 3 - Código com exemplo de declaração de variáveis no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada “inicio” e nela declara três variáveis, “soma”, “numero1” e “numero2”, lê as duas variáveis “numero1” e “numero2” com o comando “leia”, e atribui à terceira variável “soma” a soma das variáveis “numero1” e “numero2”.

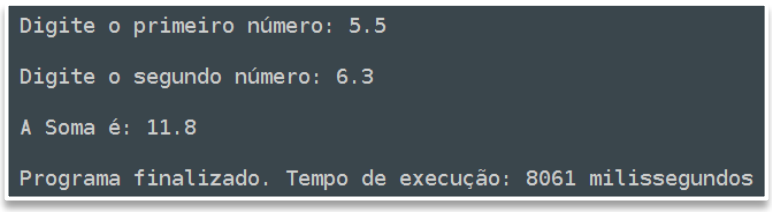



**Figura 14 - Saída do exemplo de declaração de variáveis no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do quadro 3 por meio do valor armazenado na variável “soma”. O resultado exibido é a frase “A Soma é: 11.8”.

### 1.9.3 Constantes

Constantes são valores que ficam guardados e não mudam enquanto o programa está rodando. Elas também têm nomes como as variáveis, mas ao contrário delas, não podem ser alteradas depois de serem definidas.

As constantes são úteis quando você tem valores que não mudam e serão usados várias vezes no seu programa.

```
programa
{
funcao inicio()
	{
		real raio, area
		const real pi = 3.1415

		escreva("Digite o raio: ")
		leia(raio)

		area = pi * raio * raio
		escreva("\n", "A área de um círculo de raio ", raio," é ", area, "\n")
	}
}
```

**Quadros 4 - Código com exemplo de uso de constantes no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada “inicio” e nela declara três variáveis do tipo real, “raio”, “área” e “pi”. A variável “pi” é declarada como uma constante com o valor pré-definido 3.1415. Em seguida o programa, lê a variável “raio” através do comando "leia" e atribui à variável “area” que recebe a multiplicação do valor de “raio” por ela mesma
e pela constante “pi”, depois escreve na saída o resultado de “area”.

```
Digite o raio: 2

A área de um círculo de raio 2.0 é 12.566

Programa finalizado. Tempo de execução: 2222 milissegundos
```

**Figura 15 - Saída do uso de constantes no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do quadro 4 de uso de constantes. O resultado exibido é a frase “A área de um círculo de raio 2.0 é 12.566”.

### 1.9.4 Tipos de Dados

Os tipos de dados são valores que podem ser armazenados em uma variável ou constante. Os tipos de dados mais comuns incluem:

- **Números inteiros (tipo inteiro):** Utilizados para armazenar valores inteiros positivos ou negativos.
- **Números de ponto flutuante (tipo real):** Utilizados para armazenar números decimais.

- **Texto (tipo cadeia):** Utilizado para armazenar sequências de caracteres, como palavras ou frases.
- **Caracter (tipo caracter):** São valores contidos dentro de um conjunto de caracteres alfanuméricos. Atenção, quando números são declarados como caracteres eles se tornam representativos e perdem a atribuição de valor. É composto de apenas um carácter alfanumérico ou especial.
- **Valores lógicos (tipo logico):** Utilizados para armazenar valores lógicos, verdadeiro ou falso.

**ASSISTA O VÍDEO ABAIXO PARA APROFUNDAR O SEU CONHECIMENTO**
https://www.youtube.com/watch?v=9ybMQQZtOys

**VIDEOAULA!**
Não deixe de assistir à videoaula, lá vou explicar os princípios ensinados nessa Unidade.

Agora que você assistiu à videoaula e estudou esta Unidade, acesse o **Fórum de Dúvidas**: Atividade Prática Semanal da Unidade 1 e siga as orientações para a realização da atividade prática.

Para entender quais tópicos foram apresentados na Unidade, **acesse o podcast** disponível no Ambiente Virtual de Aprendizagem (AVA).

# Unidade 02 | Desenvolver um algoritmo para resolução de um problema

## 2.1 Manipulação de Dados

### 2.1.1 Operadores de Atribuição

Os operadores de atribuição são símbolos que você usa para guardar um valor dentro de uma variável. Eles funcionam junto com algumas operações matemáticas, como somar, subtrair, multiplicar, dividir, encontrar o resto da divisão, adicionar 1 ou subtrair 1, permitindo que você atualize o valor da variável. Na tabela abaixo você pode ver exemplos de operadores de atribuição.

| Operador | Descrição | Exemplo |
|---|---|---|
| = | Atribuição simples de um valor ou expressão a uma variável. | nota1 = 0<br>escreva(nota1) → 0 |
| += | Adição do valor de uma variável ao valor de outra variável ou constante. Equivale a: variavel1 = variavel1 + variavel2. | nota1 = 5, nota2 = 4<br>nota1 += nota2<br>escreva(nota1) → 9 |
| -= | Subtração do valor de uma variável do valor de outra variável ou constante. Equivale a: variavel1 = variavel1 - variavel2. | nota1 = 9, penalidade = 1<br>nota1 -= penalidade<br>escreva(nota1) → 8 |
| *= | Multiplicação do valor de uma variável pelo valor de outra variável ou constante. Equivale a: variavel1 = variavel1 * variavel2. | nota1 = 9, peso = 3<br>nota1 *= peso<br>escreva(nota1) → 27 |
| /= | Divisão do valor de uma variável pelo valor de outra variável ou constante. Equivale a: variavel1 = variavel1 / variavel2. | media = 28, nProvas = 4<br>media /= nProvas<br>escreva(media) → 7 |
| %= | Resto da divisão do valor de uma variável pelo valor de outra variável ou constante. Equivale a: variavel1 = variavel1 % variavel2. | resto = 7<br>resto %= 2<br>escreva(resto) → 1 |
| ++ | Adição de 1 ao valor de uma variável. Equivale a: variavel1 = variavel1+1. | contador = 2<br>contador++<br>escreva(contador) → 3 |
| -- | Subtração de 1 ao valor de uma variável. Equivale a: variavel1 = variavel1-1. | contador = 2<br>contador--<br>escreva(contador) → 1 |

**Tabela 1 - Operadores de atribuição no Portugol Studio.**
**Fonte:** O autor

## 2.1.2 Operadores Aritméticos Básicos

Os operadores aritméticos básicos são símbolos matemáticos que representam operações fundamentais. Segue abaixo a lista de operadores referentes às quatro operações básicas:

- **Adição (+):** A operação de adição é usada para somar dois números e produzir um resultado que é a soma dos valores.
- **Subtração (-):** A operação de subtração é usada para subtrair um número de outro e produzir um resultado que é a diferença entre os valores.
- **Multiplicação (*):** A operação de multiplicação é usada para multiplicar dois números e produzir um resultado que é o produto dos valores.
- **Divisão (/):** A operação de divisão é usada para dividir um número por outro e produzir um resultado que é o quociente dos valores.

```
programa
{ funcao inicio()
        {
                real a, b, soma, sub, mult, div

                escreva("Digite o primeiro número: ")
                leia(a)

                escreva("\n", "Digite o segundo número: ")
                leia(b)

                soma = a + b
                sub = a - b
                mult = a * b
                div = a / b

                escreva("\n", "A soma dos números é igual a: ", soma, "\n")
                escreva("\n", "A subtração dos números é igual a: ", sub, "\n")
                escreva("\n", "A multiplicação dos números é igual a: ", mult, "\n")
                escreva("\n", "A divisão dos números é igual a: ", div, "\n")
        }
}
```

**Quadros 5 - Código com exemplo de uso de operadores aritméticos no Portugol Studio.**
**Fonte:** O autor

**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada “inicio” e nela declara seis variáveis do tipo real, "a", "b", "soma", "sub", "mult" e "div". Em seguida solicita ao usuário que forneça os valores de

dois números, recebendo os valores após cada mensagem através do comando "leia". Por fim, calcula o resultado das quatro operações matemáticas básicas, soma, subtração, multiplicação e divisão atribuindo os valores às variáveis, em seguida exibe os resultados através de mensagens realizadas com o comando "escreva".

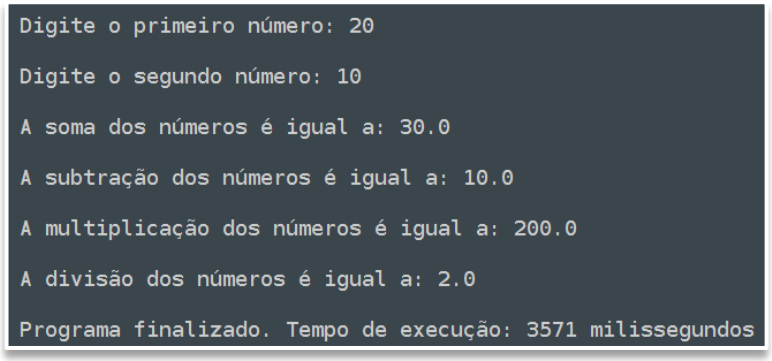



**Figura 16 - Saída do uso de operadores aritméticos no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de operadores aritméticos no Portugol Studio. O resultado exibido, baseado no fornecimento dos números 20 e 10, são as mensagens “A soma dos números é igual a 30.0”, “A subtração dos números é igual a 10.0”, “A multiplicação dos números é igual a 200.0” e “A divisão dos números é igual a 2.0”.

### 2.1.3 Prioridade dos Operadores Aritméticos Básicos

Os operadores de multiplicação, divisão, potenciação e de módulo têm prioridade em relação aos operadores de adição e subtração, e acima de todos, a utilização de parênteses pode alterar a prioridade de qualquer operador numa expressão.

A tabela abaixo mostra os principais símbolos matemáticos e a ordem em que devem ser usados, do que tem mais prioridade ao que tem menos prioridade.

| Operador | Descrição | Prioridade |
|---|---|---|
| ( ) | Parênteses, Operador de Prioridade. | 1ª |
| * | Operador de Multiplicação. | 2ª |
| / | Operador de Divisão. | 2ª |
| % | Operador de Módulo ou Resto. | 2ª |
| + | Operador de Soma. | 3ª |
| - | Operador de Subtração. | 3ª |

**Tabela 2 - Operadores aritméticos básicos.**
**Fonte:** O autor

**Descrição da figura:** Tabela com Tipo, Descrição e Prioridade dos operadores aritméticos básicos, sendo eles: adição, “+”, subtração, “-“, multiplicação, “*”, divisão, “/”, módulo ou resto, “%” e parênteses, “( )”.

ASSISTA O VÍDEO ABAIXO PARA APROFUNDAR O SEU CONHECIMENTO SOBRE O CONCEITO DE OPERADORES ARITMÉTICOS PARTE 1 e 2.
https://www.youtube.com/watch?v=Zax00WPcbec
https://www.youtube.com/watch?v=SRiP2R8Ue_o

O exemplo dos quadros a seguir demonstram a priorização dos operadores aritméticos quando usados em expressões aritméticas.

```
programa
{
	funcao inicio()
	{
		real resultado

		resultado = 5.0 + 4.0 * 2.0
		escreva("Operação: 5 + 4 * 2 = ", resultado, "\n")

		resultado = (5.0 + 4.0) * 2.0
		escreva("\n", "Operação: (5 + 4) * 2 = ", resultado, "\n")

		resultado = 1.0 + 6.0 / 3.0 * 4.0
		escreva("\n", "Operação: 1 + 6 / 3 * 4 = ", resultado, "\n")

		resultado = (4.0 + 20.0) / (3.0 * 4.0)
		escreva("\n", "Operação: (4 + 20) / (3 * 4) = ", resultado, "\n")
	}
}
```

**Quadro 6 - Código com exemplo de prioridade dos operadores aritméticos no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada “inicio” e nela declara uma variável do tipo real chamada "resultado", em seguida atribui a esta variável uma expressão matemática com os operadores matemáticos básicos e exibe os resultados nas mensagens através do comando "escreva".

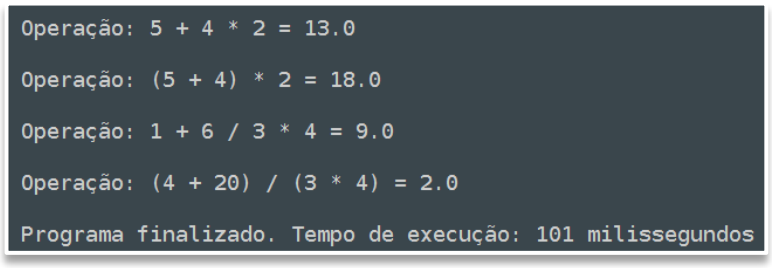



**Figura 17 - Saída de prioridade dos operadores aritméticos no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** São exibidas as saídas que foram calculadas no programa do exemplo de prioridade dos operadores aritméticos no Portugol Studio, sendo elas, "Operação: 5 + 4 * 2 = 13.0", "Operação: (5 + 4) * 2 = 18.0", "Operação: 1 + 6 / 3 * 4 = 9.0" e "(4 + 20) / (3 * 4) = 2.0".

O operador aritmético módulo "%", é um operador binário que retorna o resto da divisão inteira. Retorna o valor que sobra quando o primeiro número é dividido pelo segundo.

Por exemplo, na expressão "17 % 5", o operador módulo retorna o valor 2, que é o resto da divisão inteira de 17 por 5.

```
programa
{
	funcao inicio()
	{
		inteiro resultado

		resultado = 17 % 5
		escreva("O resto da divisão de 17 por 5 é: ", resultado, "\n")
	}
}
```

**Quadro 7 - Código com exemplo de uso do operador aritmético módulo no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada “inicio” e nela declara uma variável do tipo inteiro chamada "resultado", atribui a ela o cálculo do resto da divisão de 17 por 5 e exibe o resultado numa mensagem na tela através do comando "escreva".

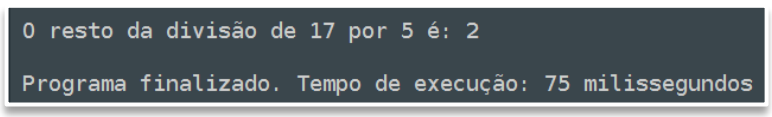



**Figura 18 - Saída do exemplo de uso do operador aritmético módulo no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de uso do operador aritmético módulo no Portugol Studio. Retorna a mensagem " O resto da divisão de 17 por 5 é: 2".

### 2.1.4 Operadores Relacionais

Os operadores relacionais são símbolos que servem para comparar dois valores. Eles são importantes na programação, pois permitem que o programa tome decisões baseadas nessas comparações. Quando você usa um operador relacional, ele retorna um resultado lógico, que pode ser Verdadeiro ou Falso.

Os principais operadores relacionais são: “igual a”, “diferente de”, “maior que”, “menor que”, “maior ou igual a” e “menor ou igual a”. Confira o quadro abaixo com exemplos.

**DICAS IMPORTANTE!**

Qualquer tipo primitivo de dado pode ser comparado através de um operador relacional.

Os dados comparados por operadores relacionais devem ser sempre do mesmo tipo.

A tabela abaixo mostra exemplos dos principais operadores relacionais.

| Operador | Símbolo | Exemplo | Resultado |
|---|---|---|---|
| Igual a | == | 20 == 5*4 | Verdadeiro |
| Diferente de | != | 10 != 20/2 | Falso |
| Maior que | > | 50/2*3 > 100 | Falso |
| Menor que | < | 9-6 < 123%7 | Verdadeiro |
| Maior ou igual a | >= | 25%5 >= 40%3 | Falso |
| Menor ou igual a | <= | 3*2-2 <= 40*2/5 | Verdadeiro |

**Tabela 3 - Operadores relacionais básicos.**
**Fonte:** o autor
**Descrição da figura:** Tabela com Operador, Símbolo, Exemplo e Resultado dos operadores relacionais básicos, sendo: “igual a”, “diferente de”, “maior que”, “menor que”, “maior ou igual a” e “menor ou igual a”.

Segue abaixo um exemplo que demonstra como os operadores relacionais realizam comparações entre valores numéricos.

```
programa
{
	funcao inicio()
	{
		escreva("O resultado da comparação se 20 == 5*4 é: ", 20 == 5*4, "\n\n")
		escreva("O resultado da comparação se 10 != 20/2 é: ", 10 != 20/2, "\n\n")
		escreva("O resultado da comparação se 50/2*3 > 100 é: ", 50/2*3 > 100, "\n\n")
		escreva("O resultado da comparação se 9-6 < 123%7 é: ", 9-6 < 123%7, "\n\n")
		escreva("O resultado da comparação se 25%5 >= 40%3 é: ", 25%5 >= 40%3, "\n\n")
		escreva("O resultado da comparação se 3*2-2 <= 40*2/5 é: ", 3*2-2 <= 40*2/5, "\n")

	}
}
```

**Quadro 8 - Código com exemplo de uso de operadores relacionais no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada “inicio” e por meio do comando “escreva” mostra na tela o resultado de todas as comparações entre os valores: “20 == 5*4”, “10 != 20/2”, “50/2*3 > 100”, “9-6 < 123%7”, “25%5 >= 40%3”, “3*2-2 <= 40*2/5”, usando os operadores relacionais básicos.

```
O resultado da comparação se 20 == 5*4 é: verdadeiro

O resultado da comparação se 10 != 20/2 é: falso

O resultado da comparação se 50/2*3 > 100 é: falso

O resultado da comparação se 9-6 < 123%7 é: verdadeiro

O resultado da comparação se 25%5 >= 40%3 é: falso

O resultado da comparação se 3*2-2 <= 40*2/5 é: verdadeiro

Programa finalizado. Tempo de execução: 107 milissegundos
```

**Figura 19 - Saída do exemplo de uso de operadores relacionais no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de uso de operadores relacionais no Portugol Studio. Retorna as mensagens as quais as expressões relacionais sejam verdadeiras ou falsas em cada operação de comparação.

### 2.1.5 Expressões Aritméticas

Em programação, uma expressão aritmética pode ser uma instrução que realiza uma operação matemática e retorna um resultado.

As operações aritméticas mais comuns são: adição (+), subtração (-), multiplicação (*) e divisão (/). As expressões aritméticas podem incluir ainda parênteses para definir a ordem de prioridade das operações.

**DICA IMPORTANTE!**

Divisões de números pelo valor 0 (zero) ou de raízes quadradas de números negativos não possuem um valor definido matematicamente para computadores. A execução dessas expressões gera erro. Evite estas situações.

**DICA IMPORTANTE!**

Deixar parênteses não pareados (não fechados) nas expressões aritméticas é um erro difícil de se localizar depois de finalizado o código. Evite esta situação.

```
programa
{
	funcao inicio()
	{
		real numero1 = 24, numero2 = 30, numero3 = 10, resultado

		resultado = numero1*2+numero2/numero3-1
		escreva("numero1*2+numero2/numero3-1 = ", resultado,"\n")
	}
}
```

**Quadro 9 - Código com exemplo de uso de expressões aritméticas no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada “inicio” e nela declara quatro variáveis do tipo real "resultado" e, com valor pré-determinado, "numero1" = 24, "numero2" = 30, "numero3" = 10. Em seguida é atribuída à variável "resultado" a expressão numero1*2+numero2/numero3-1. Em seguida exibe o resultado numa mensagem na tela através do comando "escreva".

```
numero1*2+numero2/numero3-1 = 50.0

Programa finalizado. Tempo de execução: 80 milissegundos
```

**Figura 20 - Código com exemplo de uso de expressões aritméticas no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de uso de expressões aritméticas no Portugol Studio. Retorna o resultado da expressão aritmética na mensagem “numero1*2+numero2/numero3-1 = 50.0".

### 2.1.6 Funções Primitivas

As funções primitivas são como ferramentas básicas que a linguagem de programação oferece para serem usadas no código. Elas já vêm prontas, não precisando ser criadas pelo programador. Essas funções podem incluir cálculos matemáticos complicados que seriam difíceis de fazer e programar manualmente.

Por isso, muitas linguagens de programação fornecem uma biblioteca com essas funções prontas para serem usadas nos programas. Na tabela abaixo, estão algumas das funções primitivas mais comuns. Cada linguagem de programação tem seu próprio conjunto de funções.

| Função | Descrição |
|---|---|
| **Raiz(x) ou SQRT(x)** | Raiz quadrada |
| **ABS(x)** | Valor absoluto |
| **ROUND(x)** | Valor arredondado |
| **SIN(x)** | Função seno |
| **COS()** | Função cosseno |
| **TAN()** | Função tangente |
| **LOG()** | Função logarítmica |

**Tabela 4 - Funções primitivas.**
**Fonte:** O autor
**Descrição da figura:** Tabela com Função e Descrição das funções primitivas, sendo elas: “Raiz” ou “SQRT” (função raiz quadrada), “ABS” (função valor absoluto), “ROUND” (função valor arredondado), “SIN” (função seno), “COS” (função cosseno), “TAN” (função tangente) e “LOG” (função logaritmo).

### 2.1.7 Operadores Lógicos

Os operadores lógicos são utilizados para criar expressões condicionais que serão avaliadas como verdadeiras ou falsas. Os operadores lógicos são usados para comparar valores booleanos (verdadeiro ou falso) ou para combinar múltiplas expressões condicionais.

Existem três operadores lógicos principais em lógica de programação: AND (e), OR (ou) e NOT (nao). O operador AND (e) retorna verdadeiro apenas se ambas as expressões condicionais forem verdadeiras. Caso contrário, ele retorna falso.

O operador OR (ou), retorna verdadeiro se pelo menos uma das expressões condicionais for verdadeira. Caso contrário, ele retorna falso.

O operador NOT (nao) é usado para negar uma expressão condicional. Ele inverte o valor da expressão, tornando uma condição verdadeira em falsa, e uma condição falsa em verdadeira. A tabela abaixo apresenta os operadores lógicos e suas características.

| Operador | Tipo | Operação | Prioridade |
|---|---|---|---|
| **NAO (NOT)** | Unário | Negação | 1ª |
| **E (AND)** | Binário | Conjunção | 2ª |
| **OR (OU)** | Binário | Disjunção | 3ª |

**Tabela 5 - Operadores lógicos.**
**Fonte:** O autor
**Descrição da figura:** Tabela com Operador, Tipo, Operação e Prioridade dos operadores lógicos, sendo eles: “OU”, “E” e “NAO”.

### 2.1.8 Expressões Lógicas

Você sabia que as expressões lógicas são superimportantes porque elas nos ajudam a descobrir se algo é verdadeiro ou falso? Você pode usar operadores lógicos (como "e", "ou", "nao") e os operadores relacionais (como "=", "<", ">") para combinar e comparar os valores de entrada. Então, basicamente, eu crio uma pergunta e uso os operadores para ver se a resposta é verdadeira ou falsa. Vou te mostrar um exemplo disso no Portugol Studio:

```
programa
{
	funcao inicio()
	{
		real variavel1 = 2, variavel2 = 5, variavel3 = 7
		logico resultado, A, B, C, D
		A = (variavel1<variavel2)
		B = (variavel3>variavel2)
		C = (variavel1>variavel2)
		D = (variavel1>variavel3)
		resultado = (A) e (B)
		escreva("( ",variavel1," < ",variavel2," ) e ( ",variavel3," > ",variavel2," ) = ",resultado)
		resultado = (A) e (C)
		escreva("\n\n( ",variavel1," < ",variavel2," ) e ( ",variavel1," > ",variavel2," ) = ",resultado)
		resultado = (C) e (D)
		escreva("\n\n( ",variavel1," > ",variavel2," ) e ( ",variavel1," > ",variavel3," ) = ",resultado)
		resultado = (A) ou (B)
		escreva("\n\n( ",variavel1," < ",variavel2," ) ou ( ",variavel3," > ",variavel2," ) = ",resultado)
		resultado = (A) ou (D)
		escreva("\n\n( ",variavel1," < ",variavel2," ) ou ( ",variavel1," > ",variavel3," ) = ",resultado)
		resultado = (C) ou (D)
		escreva("\n\n( ",variavel1," > ",variavel2," ) ou ( ",variavel1," > ",variavel3," ) = ",resultado)
		resultado = nao(A) e (B)
		escreva("\n\n não( ",variavel1," < ",variavel2," ) e ( ",variavel3," > ",variavel2," ) = ",resultado)
		resultado = (A) e nao(C)
		escreva("\n\n( ",variavel1," < ",variavel2," ) e não( ",variavel1," > ",variavel2," ) = ",resultado)
		resultado = nao(A) ou (D)
		escreva("\n\n não( ",variavel1," < ",variavel2," ) ou ( ",variavel1," > ",variavel3," ) = ",resultado)
		resultado = nao(C) ou (D)
		escreva("\n\n não ( ",variavel1," > ",variavel2," ) ou ( ",variavel1," > ",variavel3," ) = ",resultado)
	}
}
```

**Quadros 10 - Código exemplo de uso de expressões lógicas no Portugol Studio.**

**Fonte:** O autor

**Descrição da figura:** Este exemplo apresenta um programa de computador que possui uma função chamada “inicio” e nela declara três variáveis do tipo real com valores pré-determinados “variavel1” = 2, “variavel2” = 5, “variavel3” = 7, e cinco variáveis do tipo logico, “A“ = (variavel1 < variavel2), “B” = (variavel3 > variavel2), “C” = (variavel1 > variavel2), “D” = (variavel1 > variavel3) e "resultado", sendo este último sem valor pré-definido. Em seguida é atribuída uma expressão lógica à variável "resultado" e é exibida na tela uma mensagem com o seu valor através do comando "escreva", sendo que esta operação de atribuição é repetida dez vezes com expressões lógicas diferentes. "resultado" = ((A) e (B)), "resultado" = ((C) e (D)), "resultado" = ((A) ou (B)), "resultado" = ((A) ou (D)), "resultado" = ((C) ou (D)), "resultado" = (nao(A) e (B)), "resultado" = ((A) e nao(C)), "resultado" = (nao(A) ou (D), "resultado" = (nao(C) ou (D))).

```
( 2.0 < 5.0 ) e ( 7.0 > 5.0 ) = verdadeiro
( 2.0 < 5.0 ) e ( 2.0 > 5.0 ) = falso
( 2.0 > 5.0 ) e ( 2.0 > 7.0 ) = falso
( 2.0 < 5.0 ) ou ( 7.0 > 5.0 ) = verdadeiro
( 2.0 < 5.0 ) ou ( 2.0 > 7.0 ) = verdadeiro
( 2.0 > 5.0 ) ou ( 2.0 > 7.0 ) = falso
 não( 2.0 < 5.0 ) e ( 7.0 > 5.0 ) = falso
( 2.0 < 5.0 ) e não( 2.0 > 5.0 ) = verdadeiro
 não( 2.0 < 5.0 ) ou ( 2.0 > 7.0 ) = falso
 não ( 2.0 > 5.0 ) ou ( 2.0 > 7.0 ) = verdadeiro

Programa finalizado. Tempo de execução: 631 milissegundos
```

**Figura 21 - Saída do exemplo de uso de expressões lógicas no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa de computador do exemplo de expressões lógicas no Portugol Studio. Retorna as mensagens "( 2.0 < 5.0 ) e ( 7.0 > 5.0 ) = verdadeiro", "( 2.0 < 5.0 ) e ( 2.0 > 5.0 ) = falso", "( 2.0 > 5.0 ) e ( 2.0 > 7.0 ) = falso", "( 2.0 < 5.0 ) ou ( 7.0 > 5.0 ) = verdadeiro", "( 2.0 < 5.0 ) ou ( 2.0 > 7.0 ) = verdadeiro", "( 2.0 > 5.0 ) ou ( 2.0 > 7.0 ) = falso", "nao( 2.0 < 5.0 ) e ( 7.0 > 5.0 ) = falso", "( 2.0 < 5.0 ) e nao( 2.0 > 5.0 ) = verdadeiro", "nao( 2.0 < 5.0 ) ou ( 2.0 > 7.0 ) = falso" e "nao( 2.0 > 5.0 ) ou ( 2.0 > 7.0 ) = verdadeiro".

## 2.1.9 Tabela Verdade

Você sabe o que é uma tabela verdade? É uma tabela que mostra todas as combinações possíveis de valores verdadeiros ou falsos. Isso é útil para verificar expressões lógicas e mostrar o valor booleano resultante para cada combinação.

Se você está trabalhando com algoritmos, essa tabela pode ser uma ferramenta muito útil para visualizar o resultado de expressões e fluxos de execução. Confira abaixo alguns exemplos de tabela verdade.

| Operação 1 | Operação 2 | Operação 1 e Operação 2 | Operação 1 ou Operação 2 |
|---|---|---|---|
| **Verdadeiro** | Verdadeiro | Verdadeiro | Falso |
| **Verdadeiro** | Falso | Falso | Falso |
| **Falso** | Verdadeiro | Falso | Falso |
| **Falso** | Falso | Falso | Verdadeiro |

**Tabela 6 - Tabela Verdade, operadores lógicos E e OU.**
**Fonte:** O autor
**Descrição da figura:** Tabela Verdade com as colunas "Operação 1", "Operação 2", "Operação 1 e Operação 2" e "Operação 1 ou Operação 2". O conteúdo das linhas são valores lógicos, VERDADEIRO ou FALSO.

| Operação 1 | Operação 2 | Operação 1 e Operação 2 | Operação 1 ou Operação 2 |
|---|---|---|---|
| **30 >= 11** | 3 < 7 | Verdadeiro | Verdadeiro |
| **5 <= 5** | 3 > 12 | Falso | Verdadeiro |
| **5 < 2** | 9 > 1 | Falso | Verdadeiro |
| **3 + 5 > 15** | 4/2 < 1 | Falso | Falso |

**Tabela 7 - Tabela Verdade, exemplo dos operadores lógicos E e OU.**
**Fonte:** O autor
**Descrição da figura:** Tabela Verdade com as colunas "Operação 1", "Operação 2", "Operação 1 e Operação 2" e "Operação 1 ou Operação 2". O conteúdo das linhas são expressões lógicas de exemplo e valores lógicos correspondentes.

| Operação 1 | Não Operação 1 | Exemplo Não Operação 1 | Resultado |
|---|---|---|---|
| **Verdadeiro** | Falso | Nao(10 <= 15) | Falso |
| **Falso** | Verdadeiro | Nao(4 > 10) | Verdadeiro |

**Tabela 8 - Tabela Verdade, exemplo do operador lógico NAO.**
**Fonte:** O autor
**Descrição da figura:** Tabela Verdade com as colunas “Operação 1”, “Não Operação 1”, “Exemplo Não Operação 1” e “Resultado”.

## 2.2 Estrutura de controle se então / if - then

Você pode usar uma estrutura condicional para fazer com que o seu programa avalie uma ou mais condições lógicas e execute um bloco de código específico dependendo do resultado.

Existem várias maneiras de implementar essa estrutura, você pode usar o "if/else" ou "select/case", ou ainda loop’s como o "while" e "for", que verificam uma condição a cada iteração. Com isso, você pode controlar o fluxo de execução do seu código de maneira eficiente.

Os quadros a seguir apresentam um exemplo da sintaxe (palavras permitidas) básica de uma estrutura “se então” no Portugol Studio e um exemplo de utilização da estrutura “se então” na mesma ferramenta.

```
programa
{
	funcao inicio()
	{
		real media, nota1, nota2

		escreva("Digite a primeira nota: ")
		leia(nota1)

		escreva("\n", "Digite a segunda nota: ")
		leia(nota2)

		media=(nota1+nota2)/2

		se(media >= 7)
			escreva("\n", "Aluno Aprovado com média: ", media, "\n")
		se(media < 7)
			escreva("\n", "Aluno Reprovado com média: ", media, "\n")

	}
}
```

**Quadro 11 - Código exemplo de uso de estrutura “SE ENTÃO” no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada “inicio” e nela declara três variáveis do tipo real “media”, “nota1”, “nota2”, recebe do usuário duas notas através do comando "leia". Em seguida

são construídos dois blocos de estrutura condicional "SE ENTÃO", um em seguida do outro, que aplicam, cada um, um teste lógico através de operadores relacionais e caso seus testes lógicos sejam verdadeiros exibem uma mensagem através do comando "escreva".

```
Digite a primeira nota: 8

Digite a segunda nota: 5

Aluno Reprovado com média: 6.5

Programa finalizado. Tempo de execução: 4203 milissegundos
```

**Figura 22 - Saída do exemplo de estrutura "SE ENTÃO" no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de uso de estrutura "SE ENTÃO" no Portugol Studio. No caso do exemplo utilizado o aluno recebe a mensagem enviada por meio do comando "escreva".

**DICA IMPORTANTE!**
Muita atenção em todos os sinais de todas as expressões das estruturas condicionais.

### 2.2.1 Estrutura de controle se então senão / if - then - else

Quando você está programando, pode usar a estrutura de controle "se-então-senão" para executar diferentes blocos de código com base em uma condição. Essa estrutura é comumente implementada usando o comando "if/else".

Se a condição for verdadeira, o bloco de código associado ao "if" será executado, caso contrário, o bloco de código associado ao "else" será executado. É uma maneira fácil de controlar o fluxo do seu programa e tomar decisões com base nas condições.

```
programa
{
	funcao inicio ()
	{
		caracter letra

		escreva("Digite uma letra: ")
		leia(letra)

		se (letra == 'A' ou letra == 'E' ou letra == 'I' ou letra == 'O' ou
		    letra == 'U' ou letra == 'a' ou letra == 'e' ou letra == 'i' ou
		    letra == 'o' ou letra == 'u')
		{
			escreva("\n", "A letra ", letra, " é uma vogal", "\n")
		}
		senao
		{
			escreva("\n", "A letra ", letra, " é uma consoante", "\n")
		}
	}
}
```

**Quadro 12 - Código exemplo de uso de estrutura “SE ENTÃO SENÃO” no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada “inicio” e nela declara uma variável do tipo caracter chamada "letra", solicita que o usuário forneça uma letra através de uma mensagem na tela usando o comando "escreva" e recebe o valor através do comando "leia". Em seguida usa uma estrutura condicional "SE ENTÃO" com um teste lógico verificando se a letra fornecida é uma vogal, caso o resultado do teste seja verdadeiro, exibe uma mensagem apresentando a letra fornecida classificando-a como vogal, caso o resultado do teste não seja verdadeiro, executa o bloco de código da estrutura SENÃO e exibe uma mensagem apresentando a letra fornecida classificando-a como consoante.

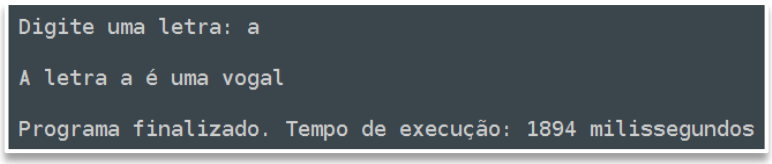



**Figura 23 - Saída do exemplo de uso da estrutura “SE ENTÃO SENÃO” no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de uso de estrutura condicional “SE ENTÃO SENÃO” no Portugol Studio. No caso a letra digitada pelo usuário, “a”, é identificada como uma vogal e é exibida a mensagem "A letra “a” é uma vogal.

## 2.2.2 Estrutura de controle se então senão aninhada / if - then - else

Quando você está programando, pode precisar usar a estrutura de controle "se-então-senão" aninhada para executar diferentes blocos de código com base em várias condições. Essa estrutura permite que você avalie várias condições em ordem e execute um bloco de código diferente para cada condição que for verdadeira.

Para fazer isso, você usa várias instruções "if/else" aninhadas, onde cada instrução avalia uma condição diferente. Lembre-se de que a ordem das instruções "if/else" é importante, pois cada condição é avaliada em ordem e, assim que uma condição for verdadeira, o bloco de código associado a ela será executado e o restante das condições serão ignoradas.

```
programa
{
        funcao inicio()
        {
                inteiro idade

                escreva("Digite a sua idade: ")
                leia(idade)

                se (idade >= 18)
                  escreva("\n", "Você é Adulto.")
                senao
                  se (idade >= 12)
                    escreva("\n", "Você é Adolescente.")
                  senao
                    se (idade >= 3)
                      escreva("\n", "Você é Criança.")
                escreva("\n")
        }
}
```

**Quadro 14 - Código exemplo de uso de estruturas “SE ENTÃO SENÃO” aninhadas no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada “inicio” e nela declara uma variável do tipo inteiro chamada "idade", solicita que o usuário forneça um valor através de uma mensagem na tela usando o comando "escreva" e armazena o valor na variável "idade" através do comando "leia". Em seguida usa três estruturas condicionais "SE ENTÃO" e "SENÃO" aninhadas.

```
Digite a sua idade: 54

Você é Adulto.

Programa finalizado. Tempo de execução: 4887 milissegundos
```

**Figura 25 - Saída do exemplo de uso de estruturas “SE ENTÃO SENÃO” aninhadas no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de uso de estruturas “SE ENTÃO SENÃO” aninhadas no Portugol Studio. Retorna a mensagem de acordo com a idade digitada, no caso, foi digitado a idade “54”, e é exibida a mensagem "Você é Adulto".

### 2.2.3 Estrutura de controle caso selecione / select - case

Quando você precisa avaliar uma expressão e executar uma ação com base no valor dela, a estrutura de controle "selecione caso" pode ser muito útil. Ela começa com a palavra "selecione"

seguida da expressão a ser avaliada. Depois, você lista diversos casos, cada um com um valor diferente que pode ser comparado com a expressão.

Se um caso corresponder ao valor da expressão, o bloco de código associado a esse caso é executado e o controle é transferido para fora da estrutura. Se nenhum caso corresponder, você pode executar um bloco de código padrão. O "selecione caso" ajuda a simplificar o código e deixá-lo mais fácil de entender quando você precisa lidar com múltiplos valores.

```
programa
{
        funcao inicio()
        {
                caracter opcao

                escreva("Digite uma opção (A, B ou C): ")
                leia(opcao)

                escolha(opcao)
                {
                        caso 'A':
                         escreva("\n", "Opção A selecionada!")
                         pare
                        caso 'B':
                         escreva("\n", "Opção B selecionada!")
                         pare
                        caso 'C':
                         escreva("\n", "Opção C selecionada!")
                         pare
                        caso contrario:
                         escreva("\n", "Opção inválida!")
                         pare
                }
                escreva("\n")
        }
}
```

**Quadro 15 - Código exemplo de uso de estrutura "CASO SELECIONE" no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada "inicio" e nela declara uma variável do tipo inteiro chamada "opcao", em seguida utiliza o comando "escreva" para apresentar 3 opções "A", "B " e "C " e solicita que o usuário escolha uma opção. Através do comando "leia" recebe a opção do usuário e inicia o bloco "CASO SELECIONE" com a variável "opcao" como parâmetro, em cada bloco de opção da estrutura é exibida uma mensagem diferente através do comando "escreva".

```
Digite uma opção (A, B ou C): A

Opção A selecionada!

Programa finalizado. Tempo de execução: 2552 milissegundos
```

**Figura 26 - Saída do exemplo de uso de estrutura "CASO SELECIONE" no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi selecionada no programa do exemplo de uso de estrutura condicional "CASO SELECIONE" no Portugol Studio. No caso, a opção digitada foi "A", o programa retorna a mensagem "Opção A selecionada!".

**DICA IMPORTANTE!**

Sempre use a opção padrão da estrutura CASO SELECIONE, ela retorna informações interessantes em caso de nenhum dos testes (CASOS) funcionar.

**DICA IMPORTANTE!**

A parada de execução PARE dentro dos CASOS evita que os testes seguintes sejam executados poupando processamento e tornando seu algoritmo mais eficiente.

## 2.3 Estruturas de repetição

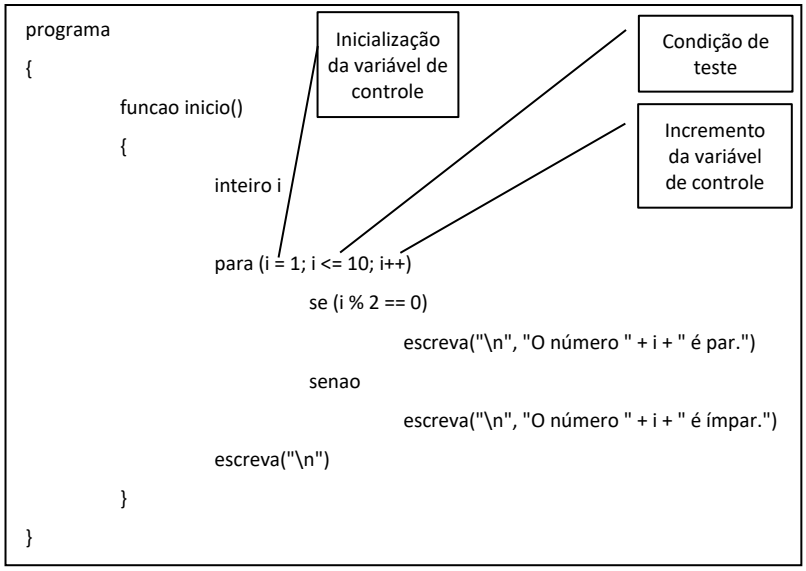
Quando você utiliza uma estrutura de repetição (LOOP), você precisa definir uma condição lógica para determinar se o conjunto de instruções dentro do bloco de código deve continuar sendo repetido ou não. Existem basicamente dois tipos de LOOP'S:

- **Repetição Determinada:** Quando, de forma prévia, se tem o número de vezes que uma determinada estrutura de comandos deve ser executada.
- **Repetição Indeterminada:** Quando não se tem um número pré-determinado de vezes que a estrutura de comandos deve ser executada.

### 2.3.1 Estrutura de repetição para faça

Com a estrutura de repetição "para faça", você pode repetir um bloco de código várias vezes enquanto uma condição é verdadeira. Para começar, você usa a palavra-chave "para" e inicializa uma variável de controle, define uma condição de teste e incrementa a variável. Em seguida, você define o bloco de código a ser repetido.

Se a condição de teste for verdadeira, o bloco de código será executado e a variável de controle será incrementada. O processo de avaliação e execução do bloco de código é repetido até que a condição de teste seja falsa. Essa estrutura é ideal para executar um bloco de código um número fixo de vezes, determinado pela condição de teste.




**Quadro 16 - Código exemplo de uso de estrutura "PARA FAÇA" no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada "inicio" e nela declara uma variável do tipo inteiro chamada "i". Em seguida inicia uma estrutura de repetição "PARA FAÇA" com a cláusula "PARA seguida de seu teste lógico entre parênteses e seus parâmetros separados por ";", sendo eles o valor inicial atribuído a "i = 1", a condição de parada que é "i <= 10"(logo, enquanto "i <= 10" o laço deve se repetir) e o valor de incremento de "i" sendo 1 (que é expresso pela expressão "i++").

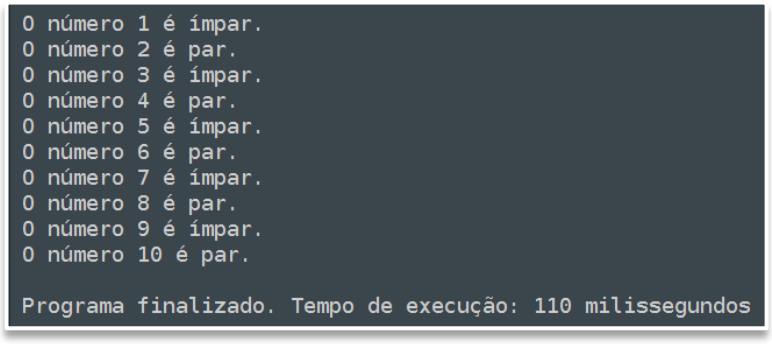



**Figura 27 - Saída do exemplo de uso de estrutura "PARA FAÇA" no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a mensagem se o número é par ou ímpar no programa do exemplo de uso de estrutura de repetição "PARA FAÇA" no Portugol Studio.

## 2.3.2 Estrutura de repetição para faça aninhada

Quando você precisa executar um bloco de código várias vezes em um loop "para faça" e tomar diferentes ações com base em uma ou mais condições, pode usar a estrutura de repetição "para faça aninhada".

Isso evita a repetição de vários loop's "para faça" separados para diferentes conjuntos de ações que precisam ser tomadas, tornando o código mais eficiente e legível. Durante cada iteração do loop, as condições são verificadas e as ações apropriadas são executadas.

```
programa
{
	funcao inicio()
	{
		inteiro contador1, contador2, numeroLinhas = 3, numeroColunas = 3

		para(contador1 = 0; contador1 <= numeroLinhas; contador1++)
		{
			para(contador2 = 0; contador2 <= numeroColunas; contador2++)
			{
				escreva(" ",contador1+contador2," ")
			}
			escreva("\n")
		}
	}
}
```

**Quadro 17 - Código exemplo de uso de estrutura "PARA FAÇA" aninhada no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada início e nela declara quatro variáveis do tipo inteiro chamadas "contador1", "contador2", "numeroLinhas" e "numeroColunas", sendo estas duas últimas com os valores pré-definidos 3. Em seguida começa a primeira estrutura de repetição, com o teste lógico tendo "contador1" = 0, "contador1" <= "numeroLinhas" e "contador1 ++". Dentro da primeira estrutura de repetição inicia-se a segunda, com o teste lógico "contador2" = 0, "contador2" <= "numeroColunas" e "contador2 ++". Dentro da segunda estrutura de repetição, através do comando "escreva" é exibida a soma das variáveis "contador1" e "contador2".

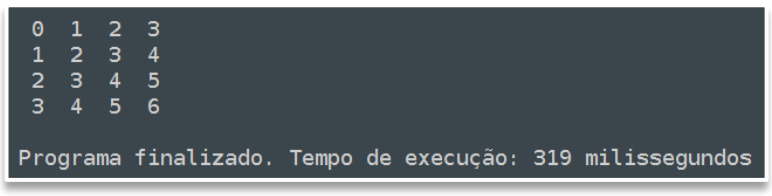



**Figura 28 - Saída do exemplo de uso de estrutura "PARA FAÇA" aninhada no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de uso de estrutura condicional "PARA FAÇA" aninhada no Portugol Studio. Retorna uma matriz numérica com 4 linhas e 4 colunas, sendo na primeira linha os valores "0, 1, 2, 3", na segunda linha os valores "1, 2, 3, 4", na terceira linha os valores "2, 3, 4, 5" e na quarta linha os valores "3, 4, 5, 6".

## 2.3.3 Estrutura de repetição enquanto faça

Quando você precisa executar um conjunto de instruções repetidamente, mas não sabe quantas vezes isso precisa acontecer, a estrutura "enquanto faça" é útil. Essa estrutura de controle executa as instruções enquanto a condição definida for verdadeira e, em cada iteração do loop, a condição é verificada novamente.

Quando a condição se tornar falsa, o loop será interrompido e a execução continuará a partir do ponto imediatamente após o fim do loop. A estrutura "enquanto faça" é uma forma eficiente de lidar com situações em que o número de repetições não é definido antecipadamente.

```
programa
{
        funcao inicio()
        {
                inteiro i = 1, soma = 0

                enquanto (i <= 100)
                {
                        soma = soma + i
                        i++
                }

                escreva("A soma dos números de 1 a 100 é: ", soma, "\n")
        }
}
```

**Quadro 18 - Código exemplo de uso de estrutura “ENQUANTO FAÇA” no Portugol Studio.**
**Fonte:** O autor

**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada “inicio” e nela declara duas variáveis do tipo inteiro chamadas "i" e “soma”, sendo "i" com o valor predefinido 1 e “soma” com o valor predefinido 0, em seguida vem a estrutura de repetição “ENQUANTO FAÇA” com a cláusula "ENQUANTO" e seu teste lógico entre parênteses verificando se "i <= 100”. Depois do teste lógico da estrutura de repetição vem o corpo delimitado entre chaves "{}", nele, o contador “soma” acumula a soma dos números até que a condição seja falsa, a cada passagem “i” é incrementado em 1. Quando a condição for falsa o comando escreva exibe a mensagem com o total da soma dos números.

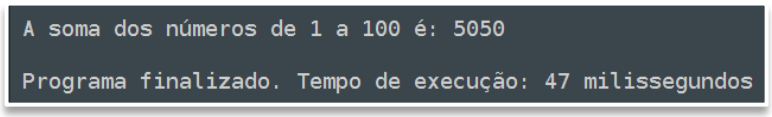



**Figura 29 - Saída do exemplo de uso de estrutura "ENQUANTO FAÇA" no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de uso de estrutura condicional "ENQUANTO FAÇA" no Portugol Studio. Retorna, a soma dos números de 1 a 100.

## 2.3.4 Estrutura de repetição faça enquanto

Quando você precisa executar um conjunto de instruções pelo menos uma vez e repeti-las enquanto uma condição seja verdadeira, você pode usar a estrutura de repetição "faça enquanto". Isso é útil quando não se sabe quantas vezes as instruções precisam ser executadas, mas sabe-se que pelo menos uma vez é necessário.

Após a primeira execução, a condição é verificada e, se for verdadeira, o conjunto de instruções é repetido. A cada iteração, a condição é verificada novamente. Quando a condição for avaliada como falsa, o loop é encerrado e a execução continua a partir do ponto imediatamente após o fim do loop.

```
programa
{
	funcao inicio()
	{
		inteiro senha

		faca
		{
			escreva("Digite a senha: ")
			leia(senha)
		}
		enquanto (senha != 1234)
			escreva("\n", "Senha correta!", "\n")
	}
}
```

**Quadro 19 - Código exemplo de uso de estrutura "FAÇA ENQUANTO" no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada "inicio" e nela declara uma variável do tipo inteiro chamada "senha", Em seguida é iniciada uma estrutura de repetição "FAÇA ENQUANTO" e, após a cláusula "FACA", dentro do corpo da estrutura, delimitado entre chaves "{}", através do comando "escreva", é solicitado que o usuário informe a senha e o valor é recebido na variável "senha" através do comando "leia". Depois do conteúdo do corpo da estrutura é posicionada a cláusula "ENQUANTO" seguida da condição de parada delimitada entre parênteses, sendo esta condição a expressão lógica "'senha != 1234". Após a estrutura de repetição é exibida a mensagem "Senha correta!" através do comando "escreva".

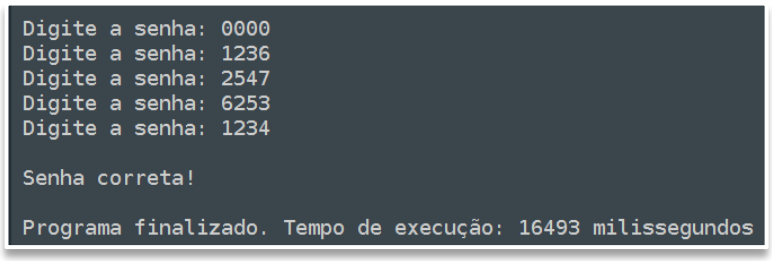



**Figura 30 - Saída do exemplo de uso de estrutura "FAÇA ENQUANTO" no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi determinada no exemplo de uso de estrutura condicional "FAÇA ENQUANTO" no Portugol Studio. No caso, na primeira iteração, foi exibida a mensagem "Digite a senha:", em seguida foram fornecidas quatro senhas incorretas nas quatro primeiras iterações, na quinta iteração foi fornecida a senha correta que retornou a mensagem "Senha correta!".

**DICA IMPORTANTE!**
Tome cuidado ao utilizar a estrutura de repetição "enquanto faça", pois se a condição nunca se tornar falsa, o loop se tornará infinito e o programa ficará preso em um laço sem fim.

**DICA IMPORTANTE!**
Em algoritmos com LOOP'S, sejam do tipo PARA FAÇA, ENQUANTO FAÇA ou FAÇA ENQUANTO, é comum surgir a necessidade do uso de variáveis do tipo contador e/ou acumulador.

## 2.4 Conceitos: Vetores, Matrizes, Procedimentos, Funções

Agora vamos abordar conceitos adicionais presentes em todas as linguagens de programação. Até agora vimos que uma variável armazena um único valor por vez, porém existem maneiras mais práticas de se lidar com esse problema.

### 2.4.1 Vetores

Você pode usar vetores para armazenar uma coleção de valores do mesmo tipo, como números, strings ou objetos. Eles são compostos por uma sequência de elementos que são

identificados por um índice numérico. Com vetores, você pode acessar rapidamente qualquer elemento da coleção, basta informar o seu índice.

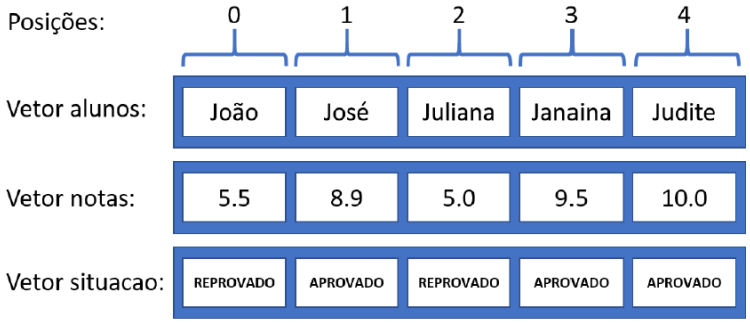



**Figura 31 - Exemplo gráfico de estruturas de "VETORES".**
**Fonte:** O autor
**Descrição da figura:** Três vetores, "alunos", "notas" e "situacao", representados por três retângulos longos, um em acima do outro, com cinco retângulos menores subdividindo cada vetor e representando as posições ou espaços que os vetores possuem. Cada posição possui um índice, e estes vão de 0 a 4. Os vetores estão preenchidos. "alunos" possui "João", "José", "Juliana", "Janaina" e "Judite". "notas" possui "5.5", "8.9", "5.0", "9.5" e "10.0". "situacao" possui "reprovado", "aprovado", "reprovado", "aprovado" e "aprovado". Pode-se interpretar que o aluno "João" tem a nota "5.5" e a situação "reprovado".

**DICA IMPORTANTE!**

No Portugol o índice de vetores e de matrizes começa com 0. Ao trabalhar com estas estruturas fique atento.
Podemos declarar vetores e matrizes apenas com as dimensões ou já atribuindo os seus valores.

Os quadros a seguir apresentam um exemplo da utilização de vetores no Portugol Studio, nele três vetores de tipos diferentes são criados.

A figura 31 explica graficamente como os dados resultantes do algoritmo ficam distribuídos nos vetores criados.

```
programa
{
	funcao inicio()
	{
		real notas[] = {5.5,8.9,5.0,9.5,10.0}
		cadeia alunos[] = {"João","José","Juliana","Janaina","Judite"}, situacao[5]
		inteiro contador
		para(contador = 0; contador<5; contador++)
		{
			se(notas[contador]>7)
			{
				situacao[contador] = "APROVADO"
			}
			senao
			{
				situacao[contador] = "REPROVADO"
			}
		}
		para(contador = 0; contador<5; contador++)
		{
			escreva("ALUNO: ", alunos[contador], "\n")
			escreva("NOTA: ", notas[contador], "\n")
			escreva("SITUAÇÃO: ", situacao[contador], "\n\n")
		}
	}
}
```

**Quadro 20 - Código exemplo de uso de "VETORES" no Portugol Studio.**
**Fonte:** O autor

**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada "inicio" e nela declara dois vetores. O primeiro vetor é do tipo real e é chamado "notas", atribui a este os valores "5.5", "8.9", "5.0", "9.5" e "10.0". O segundo vetor é do tipo cadeia e é chamado "alunos", atribui a este os valores "João", "José", "Juliana", "Janaina" e "Judite". Logo após, declara uma variável do tipo inteiro chamada "contador". Em seguida inicia uma estrutura de repetição "PARA" com os parâmetros ("contador" = 0; "contador"<5; "contador"++), dentro do corpo desta estrutura de repetição é criada uma estrutura condicional "SE SENÃO" com o teste lógico da cláusula "SE" (notas[contador]>7) e no seu corpo situacao[contador]="APROVADO" e no corpo da cláusula "SENÃO" situacao[contador]="REPROVADO". Depois do fechamento da primeira estrutura de repetição "PARA" é iniciada outra com os mesmos parâmetros ("contador" = 0; "contador"<5; "contador"++), dentro do corpo desta estrutura, através do comando "escreva" são exibidos os alunos, as notas e a situação dos alunos extraindo estas informações das posições dos vetores com o índice igual ao valor de "contador".

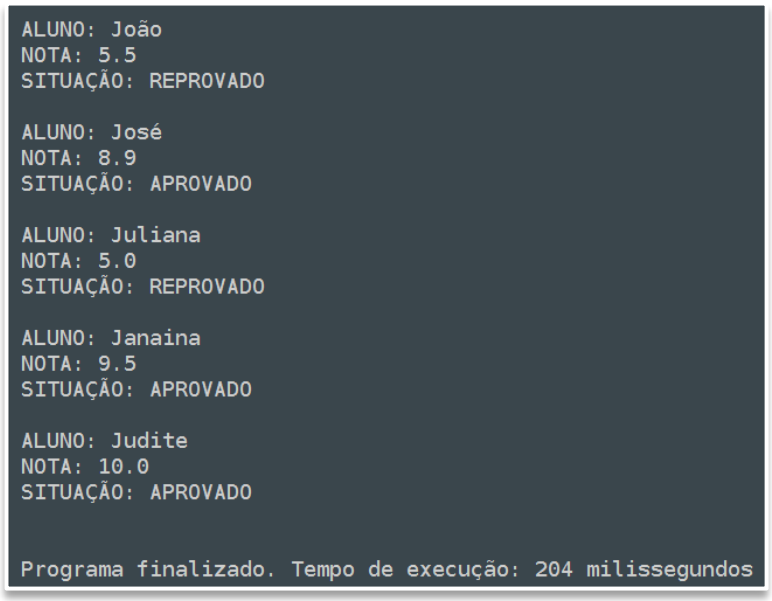



**Figura 32 - Saída do exemplo de uso de"VETORES" no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de uso de "VETORES" no Portugol Studio. No caso, são exibidos o nome, a nota e a situação de cada aluno. ALUNO: João, NOTA: 5.5, Situação: REPROVADO. ALUNO: José, NOTA: 8.9, Situação: APROVADO. ALUNO: Juliana, NOTA: 5.0, Situação: REPROVADO. ALUNO: Janaina, NOTA: 9.5, Situação: APROVADO. ALUNO: Judite, NOTA: 10.0, Situação: APROVADO.

**ASSISTA O VÍDEO ABAIXO PARA AMPLIAR OS CONHECIMENTOS DE VETORES EM ALGORITMOS, PARTE 1 e 2.**
https://www.youtube.com/watch?v=Tp1iHwek9Hg
https://www.youtube.com/watch?v=he9k99LOD64

## 2.4.2 Matrizes

Você pode pensar nas matrizes como uma evolução dos vetores. Ao invés de armazenar apenas uma dimensão de valores, elas armazenam múltiplas dimensões de valores. Elas são como uma coleção de vetores, e são muito úteis para armazenar tabelas de valores, imagens e outras estruturas de dados multidimensionais.

Com as matrizes, você pode acessar valores específicos por meio de índices em cada uma de suas dimensões.

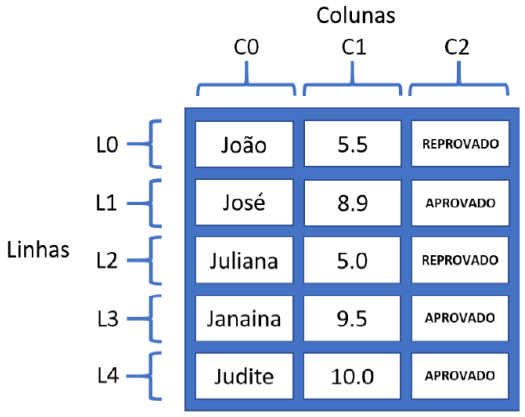



**Figura 33 - Exemplo gráfico de estruturas de "MATRIZES".**
**Fonte:** O autor
**Descrição da figura:** Pseudocódigo genérico da declaração de "MATRIZES", começa com o tipo de variável da matriz, seguido do nome e a quantidade de linhas delimitada entre colchetes "[ ]" e da quantidade de colunas, também delimitada entre colchetes.

**DICA IMPORTANTE!**

Em algumas linguagens as matrizes são consideradas como um array de array's, ou seja, um vetor que contêm vetores.
Podemos percorrer os elementos de uma matriz usando estruturas de repetição e contadores de índices/posição.

Os quadros a seguir apresentam um exemplo da utilização de matrizes no Portugol Studio.

A figura 33 explica graficamente como os dados resultantes do algoritmo ficam distribuídos na matriz.

```
programa
{
	funcao inicio()
	{
		inteiro contadorLinhas, contadorColunas
		cadeia matriz[5][3] =
		{{"João    ","5.5  ","Reprovado"},
		{"José    ","8.9  ","Aprovado"},
		{"Juliana  ","5.0  ","Reprovado"},
		{"Janaina  ","9.5  ","Aprovado"},
		{"Judite   ","10.0 ","Aprovado"}}

		para(contadorLinhas=0; contadorLinhas<=4; contadorLinhas++)
		{
			para(contadorColunas=0; contadorColunas<=2; contadorColunas++)
			{
				escreva(matriz[contadorLinhas][contadorColunas]," ")
			}
			escreva("\n")
		}
	}
}
```

**Quadro 21 - Código exemplo de uso de "MATRIZES" no Portugol Studio.**
**Fonte:** O autor

**Descrição da figura:** Este exemplo apresenta um programa de computador que possui uma função chamada "inicio" e nela declara duas variáveis do tipo inteiro chamadas "contadorLinhas" e "contadorColunas". Em seguia é declarada uma matriz do tipo "cadeia" com 5 linhas e 3 colunas e com os valores pré-definidos. linha 1: "João", "5.5", "Reprovado". linha 2: "José", "8.9", "Aprovado". linha 3: "Juliana", "5.0", "Reprovado". linha 4: "Janaina", "9.5", "Aprovado". linha 5: "Judite", "10.0", "Aprovado". Depois das declarações é iniciada uma estrutura de repetição "PARA" com os parâmetros (contadorLinhas=0; contadorLinhas<=4; contadorLinhas++). Dentro do corpo desta primeira estrutura é criada outra do mesmo tipo com os parâmetros (contadorColunas=0; contadorColunas<=2; contadorColunas++) e dentro do corpo desta, através do comando "escreva" é apresentado o elemento da matriz com o índice de linha igual a "contadorLinhas" e o índice de colunas igual a "contadorColunas". Após fechar a segunda estrutura de repetição, ainda no corpo da primeira, é dada uma instrução para pular linha através do comando "escreva" e do parâmetro "\n".

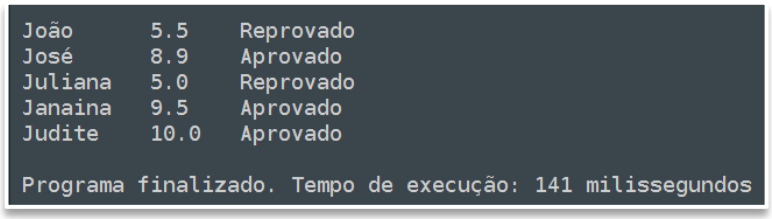



**Figura 34 - Saída do exemplo de uso de "MATRIZES" no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/

**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de uso de "MATRIZES" no Portugol Studio. Através do comando "escreva" e de estruturas de repetição "PARA" aninhadas são exibidas as informações dos alunos armazenadas nas matrizes, sendo um aluno por linha. Linha 1: "João", "5.5", "Reprovado". Linha 2: "José", "8.9", "Aprovado". Linha 3: "Juliana", "5.0", "Reprovado". Linha 4: "Janaina", "9.5", "Aprovado". Linha 5: "Judite", "10.0", "Aprovado".

### 2.4.3 Procedimentos

Quando você quer que um bloco de código execute uma tarefa específica várias vezes, você pode criar um procedimento. Ele é definido uma única vez e depois é chamado várias vezes durante a execução do programa.

Os procedimentos ajudam a organizar o código, deixando-o mais fácil de entender e de manter. Além disso, você pode passar parâmetros para o procedimento e obter valores de retorno como resultado da sua execução.

### 2.4.4 Funções

Quando você está programando, é comum usar funções para executar tarefas específicas e retornar valores. A diferença entre funções e procedimentos é que as funções sempre retornam um valor. Você pode passar um ou mais parâmetros para a função e ela executará um bloco de código e retornará um valor como resultado.

Agora vou mostrar um exemplo de função simples que calcula a média de três notas. Você usa a palavra-chave "funcao" para definir uma função e especifica o tipo de valor que a função retornará. Em seguida, você nomeia a função e lista os parâmetros necessários entre parênteses. Quando a função é chamada, ela executa o bloco de código e retorna o valor calculado.

```
programa
{
	funcao inicio()
	{
		real nota1, nota2, nota3, media

		escreva("Digite a primeira nota: ")
		leia(nota1)

		escreva("Digite a segunda nota: ")
		leia(nota2)

		escreva("Digite a terceira nota: ")
		leia(nota3)

		media = calcular media(nota1, nota2, nota3)

		escreva("\n", "A média das notas é: ", media, "\n")
	}

	funcao real calcular media(real nota1, real nota2, real nota3)
	{
		real media = (nota1 + nota2 + nota3) / 3
		retorne media
	}
}
```

**Quadro 22 - Código exemplo de uso de “FUNÇÕES” no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa que possui uma função chamada “inicio” e nela declara quatro variáveis do tipo real chamadas "nota1", "nota2", "nota3" e "media" que recebe o valor de uma função chamada "calcular media" que recebe como parâmetros "nota1", "nota2" e "nota3". Em seguida, através do comando "escreva" é exibida a média das notas. Após o fechamento da função início é colocada a declaração da função "calcular media" recebendo três variáveis do tipo real como parâmetros, na primeira linha do corpo da função é calculada a média das três
notas e na outra linha o comando "retorne" que retorna a média.

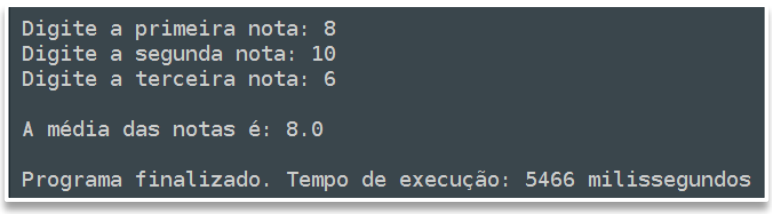



**Figura 35 -Saída do exemplo de uso de “FUNÇÕES” no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de uso de “FUNÇÕES” no Portugol Studio. No caso, é exibida a mensagem "A média das notas é: 8.0".

**ASSISTA O VÍDEO ABAIXO PARA AMPLIAR O CONCEITO DE FUNÇÕES PARTE 2, 3 e 4**

https://www.youtube.com/watch?v=8Ivum_dHgL0
https://www.youtube.com/watch?v=RhsXgFpcNlI
https://www.youtube.com/watch?v=TL46G0ajoJg

**ASSISTA O VÍDEO ABAIXO PARA AMPLIAR O CONCEITO DE BIBLIOTECAS DE FUNÇÕES NO PORTUGOL STUDIO.**

https://www.youtube.com/watch?v=rs8ihN08bgU

### 2.4.5 Recursividade

Você sabia que recursividade é quando uma função chama a si mesma durante sua execução? Isso significa que a função é executada novamente com novos parâmetros de entrada até que uma condição de parada seja alcançada e um valor de retorno seja fornecido.

Esse valor é então passado de volta para a chamada anterior da função, e assim por diante, até que a função original termine. Um exemplo de função recursiva é o retorno do fatorial de um número.

```
programa
{
	funcao inicio()
	{
		inteiro numero, resultado

		escreva("Digite um número inteiro positivo: ")
		leia(numero)

		resultado = fatorial(numero)

		escreva("\n", "O fatorial de ", numero, " é: ", resultado, "\n")
	}

	funcao inteiro fatorial(inteiro numero)
	{
		se(numero <= 1)
			retorne 1
		senao
			retorne numero * fatorial(numero - 1)
	}
}
```

**Quadro 23 - Código exemplo de uso de "RECURSIVIDADE" no Portugol Studio.**
**Fonte:** O autor
**Descrição da figura:** Este exemplo apresenta um programa onde é declarada uma função chamada "inicio" e nela declara duas variáveis do tipo inteiro, "numero" e "resultado", em seguida é solicitado que se digite um número inteiro positivo através do comando "leia", em seguida a variável "resultado" recebe o retorno da função "fatorial" com o resultado da operação.

```
Digite um número inteiro positivo: 10

O fatorial de 10 é: 3628800

Programa finalizado. Tempo de execução: 2893 milissegundos
```

**Figura 36 - Saída do exemplo de uso de "RECURSIVIDADE" no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de uso de "RECURSIVIDADE" no Portugol Studio. No caso, é exibida uma mensagem "O fatorial de 10 é: 3628800".

**VIDEOAULA!**
Não deixe de assistir à videoaula, lá vou explicar os princípios ensinados nessa Unidade.

Agora que você assistiu à videoaula e estudou esta Unidade, acesse o **Fórum de Dúvidas: Atividade Prática Semanal da Unidade 2** e siga as orientações para a realização da atividade prática.

Para entender quais tópicos foram apresentados na Unidade, **acesse o podcast** disponível no Ambiente Virtual de Aprendizagem (AVA).

# Unidade 03 | Reconhecer o que é Pensamento Computacional

## 3.1 O Pensamento computacional e suas bases

Pensamento computacional é quando você usa ideias e técnicas da ciência da computação para pensar de forma organizada e resolver problemas de um jeito mais fácil. Isso inclui separar o problema em partes menores e mais simples (decomposição), ignorar detalhes menos importantes e focar no que importa (abstração), identificar padrões e repetições (reconhecimento de padrões) e criar um conjunto de instruções para resolver o problema passo a passo (algoritmos).

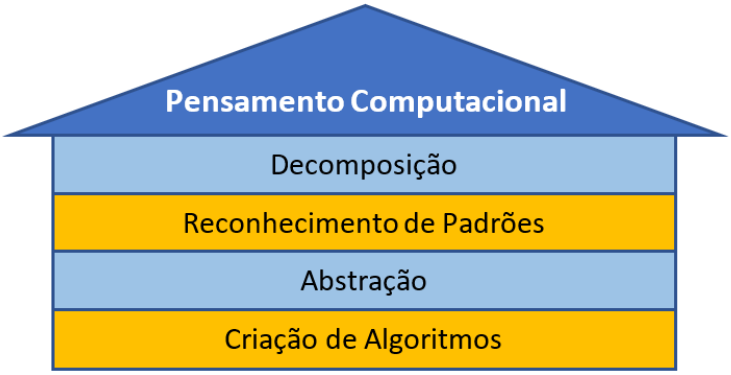



**Figura 37 - Imagem ilustrativa das bases do pensamento computacional.**
**Fonte:** O autor
**Descrição da figura:** A figura acima representa as bases do pensamento computacional, decomposição, abstração, reconhecimento de padrões e a criação de algoritmos.

### 3.1.1 Decomposição de problemas

Para Liukas (2015), esse é um processo no qual os problemas são quebrados em partes menores. Podemos usar como exemplo a decomposição de refeições, receitas culinárias e as fases que compõem um jogo. Na realidade trata-se de quebrar um problema em partes menores, que são mais fáceis de entender e resolver.

Segundo Brackmann (2017), quando a decomposição é aplicada a elementos físicos, como, por exemplo, consertar um carro através da decomposição de suas partes, a manutenção torna-se mais fácil.

Para Csizmadia (2015), essa técnica da decomposição possibilita resolver problemas complexos de forma mais simples, facilita a compreensão de novas situações e possibilita projetar sistemas de grande porte.

**Figura 38 - Imagem ilustrativa da decomposição de um problema em partes menores.**
**Fonte:** O autor
**Descrição da figura:** A figura acima representa a decomposição, que é um dos pilares do pensamento computacional, envolve identificar um problema e quebrá-lo em partes menores.

### 3.1.2 Reconhecimento de padrões

O reconhecimento de padrões no pensamento computacional é a capacidade de identificar similaridades e padrões nas estruturas a fim de extrair conhecimento. Liukas (2015) define o Reconhecimento de Padrões como encontrar similaridades e padrões com o intuito de resolver problemas complexos de forma mais eficiente.

Isso envolve o uso de técnicas para procurar padrões e elementos que são mais similares, e por meio da decomposição de problemas alcançar a solução mais rapidamente. Ainda segundo Brackmann (2017), um exemplo prático é através da identificação de similaridades, padrões ou ligações, veja o exemplo abaixo:


**Figura 39 - Imagem ilustrativa da decomposição em partes menores.**
**Fonte:** Brackmann (2017)
**Descrição da figura:** A figura acima representa a decomposição em partes para encontrar padrões, similaridades e isolar as partes identificando possíveis ligações.

### 3.1.3 Abstração

Abstração é um conceito importante do pensamento computacional que se refere a habilidade de identificar as partes mais importantes de um problema e deixar de lado os detalhes menos importantes. Em outras palavras, você precisa saber simplificar um problema complexo e torná-lo mais fácil de entender.

Na prática, a abstração pode ser usada em diferentes níveis do problema, desde os objetivos gerais até os detalhes mais específicos. Por exemplo, na programação, você pode usar a abstração para identificar os principais componentes do sistema, como as entradas e saídas, as funções e a lógica. E então, ignorar detalhes menos importantes, como a forma como os dados são armazenados na memória.

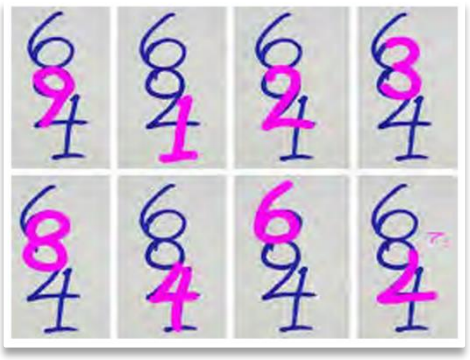


**Figura 40 - Imagem ilustrativa de exercício de abstração.**
**Fonte:** Computational Think (Google)
**Descrição da figura:** A figura acima representa uma abstração indicando as partes mais importantes da imagem e descartando as menos importantes.

### 3.1.4 Criação de algoritmos

Para você conseguir pensar de forma estruturada e resolver problemas, é importante conhecer os algoritmos. Eles são uma sequência de passos bem definidos que permitem que você execute uma tarefa de forma organizada. Isso significa que você precisa ter lógica para criar uma lista de instruções que possam ser seguidas de forma ordenada para resolver um problema.

Quando você consegue criar um algoritmo eficiente, você consegue resolver problemas de forma mais rápida e eficaz. Segue abaixo um exemplo interessante de um algoritmo para ordenar uma lista de valores selecionando repetidamente o menor valor da lista não ordenada e colocando-o no início da lista ordenada.

Agora vamos ver como o algoritmo de ordenação funciona, em seguida vou detalhar cada passo para que você compreenda claramente a sequência de passos:

1. Definir uma lista de valores a ser ordenada;
2. Definir uma lista vazia para armazenar os valores ordenados;
3. Enquanto a lista não ordenada não estiver vazia, repetir os passos 4 e 5;
4. Encontrar o menor valor na lista não ordenada;
5. Remover o menor valor da lista não ordenada e adicioná-lo ao final da lista ordenada.

Agora suponha que temos a seguinte lista de valores: 5, 2, 4, 6, 1, 3. O algoritmo de ordenação por seleção funcionaria da seguinte forma:

1. Definir uma lista de valores a ser ordenada chamada (lista não ordenada) como os valores: 5, 2, 4, 6, 1, 3;
2. Definir uma lista vazia chamada (lista ordenada) para armazenar os valores ordenados;
3. Enquanto a lista não ordenada não estiver vazia, repetir os passos 4 e 5;
4. Encontrar o menor valor na lista não ordenada: o menor valor é 1;
5. Remover o menor valor da lista não ordenada e adicioná-lo ao final da lista ordenada: a lista ordenada agora é 1;
6. Encontrar o menor valor na lista não ordenada: o menor valor é 2;
7. Remover o menor valor da lista não ordenada e adicioná-lo ao final da lista ordenada: a lista ordenada agora é 1, 2;
8. Encontrar o menor valor na lista não ordenada: o menor valor é 3;
9. Remover o menor valor da lista não ordenada e adicioná-lo ao final da lista ordenada: a lista ordenada agora é 1, 2, 3;
10. Encontrar o menor valor na lista não ordenada: o menor valor é 4;
11. Remover o menor valor da lista não ordenada e adicioná-lo ao final da lista ordenada: a lista ordenada agora é 1, 2, 3, 4;
12. Encontrar o menor valor na lista não ordenada: o menor valor é 5;

14. Remover o menor valor da lista não ordenada e adicioná-lo ao final da lista ordenada: a lista ordenada agora é 1, 2, 3, 4, 5;
15. Encontrar o menor valor na lista não ordenada: o menor valor é 6;
16. Remover o menor valor da lista não ordenada e adicioná-lo ao final da lista ordenada: a lista ordenada agora é 1, 2, 3, 4, 5, 6.

**O resulta do final é a lista ordenada com os valores: 1, 2, 3, 4, 5, 6**

## 3.2 Computação desplugada

Quando se trata de pensamento computacional, a computação desplugada é uma maneira de aprender usando atividades e ferramentas não digitais. Essa abordagem ajuda a ensinar conceitos importantes de programação e pensamento computacional que você poderá usar na vida real, sem precisar de dispositivos eletrônicos ou conexão com a internet.

Segue abaixo alguns exemplos de computação desplugada que você pode usar para entender conceitos de programação e pensamento computacional:

1. **Jogo do labirinto:** Você pode desenhar um labirinto em papel e usar um lápis para guiar um personagem através dele. O objetivo é programar o caminho que o personagem deve seguir para chegar ao final do labirinto.
2. **Cartões de codificação:** Você pode usar cartões com comandos de programação (como "avançar", "virar à esquerda", "virar à direita" e "parar") para criar um programa que guia um robô de papel por um percurso.
3. **Jogo de memória:** Você pode desenhar pares de cartões com símbolos de programação (como símbolos para "avançar", "virar à esquerda", "virar à direita" e "parar") e jogam um jogo de memória, combinando símbolos com ações correspondentes.

Segue abaixo exemplo de um jogo de conversão com um baralho binário. É importante que você lembre que as informações são armazenadas e transmitidas para o computador como uma série de zeros e uns. A Figura abaixo mostra as cartas para conversão binária.

A organização dos cartões deve ser da direita para esquerda, obedecendo à ordem crescente, neste caso, o dobro do número de pontos da carta anterior (BELL et al., 2011).

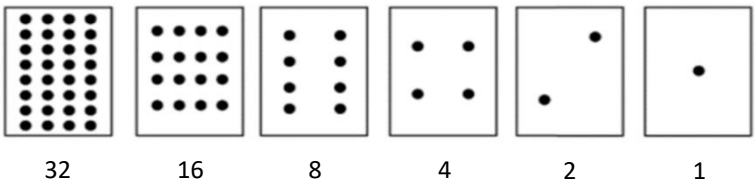



**Figura 41 - Cartas de um baralho binário.**
**Fonte:** O autor
**Descrição da figura:** A figura acima mostra 6 cartas de um baralho binário.

Quando se trata do sistema binário, só existem os números 0 e 1 para representar os números. Para converter um número binário usando cartões, basta colocar a carta virada para baixo para representar “0” e virada para cima para representar “1”. Para converter para a base decimal, é só contar quantos pontos aparecem nas cartas viradas para cima.

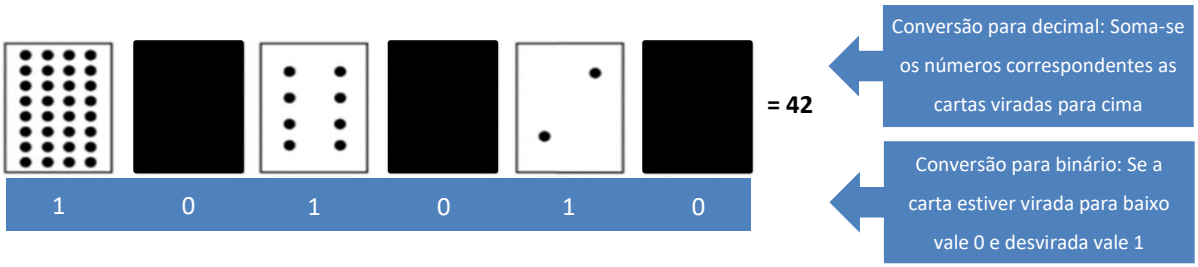



**Figura 42 - Cartas de um baralho binário.**
**Fonte:** O autor
**Descrição da figura:** A figura acima mostra 6 cartas com informações de conversão para decimal e para binário.

## 3.3 Computação plugada

Você já ouviu falar sobre a computação plugada? É uma ideia muito legal onde os dispositivos são criados para se conectarem uns aos outros fisicamente, criando uma rede super flexível e adaptável. Em vez de depender de redes de computadores comuns, os dispositivos plugados conversam diretamente entre si, criando uma rede distribuída e descentralizada.

Isso faz com que seja super fácil adicionar ou remover dispositivos da rede, sem precisar se preocupar em configurar um servidor. Além disso, é uma opção mais eficiente em termos de energia, pois os dispositivos podem ser desligados quando não estão em uso.

Só que é importante ficar ligado nos desafios de segurança e privacidade dos dados. Como os dispositivos conversam diretamente uns com os outros, é essencial garantir que as conexões sejam seguras e que os dados sejam criptografados direitinho.

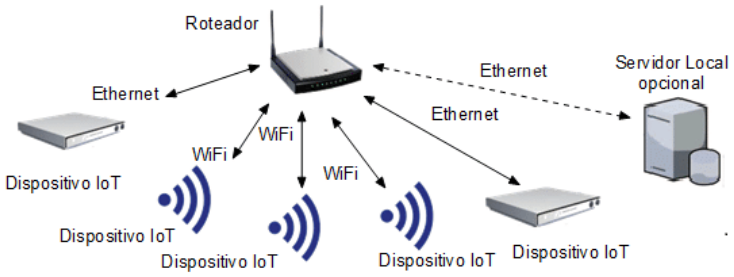



**Figura 43 - Dispositivos IoT.**
**Fonte:** https://embarcados.com.br/modelos-de-computacao-para-a-internet-das-coisas-iot
**Descrição da figura:** IoT (Internet das Coisas) é um exemplo de computação plugada. A computação plugada refere-se ao uso de dispositivos eletrônicos que são conectados a outras máquinas ou redes através de um cabo ou conexão sem fio.

**VIDEOAULA!**
Não deixe de assistir à videoaula, lá vou explicar os princípios ensinados nessa Unidade.

Agora que você assistiu à videoaula e estudou esta Unidade, acesse o **Fórum de Dúvidas: Atividade Prática Semanal da Unidade 3** e siga as orientações para a realização da atividade prática.

Para entender quais tópicos foram apresentados na Unidade, **acesse o podcast** disponível no Ambiente Virtual de Aprendizagem (AVA).

# Unidade 04 | Identificar como desenvolver o Pensamento Computacional

## 4.1 Formular problemas

Quando você está tentando resolver um problema usando o pensamento computacional, é importante seguir um processo organizado. Isso significa usar habilidades como abstração, decomposição, reconhecimento de padrões e algoritmos para entender o problema de forma clara e identificar as etapas que você precisa seguir para resolvê-lo. **É como um mapa que te ajuda a navegar pelo problema de maneira mais eficiente e eficaz.**

Um exemplo prático de como formular um problema usando as boas práticas do pensamento computacional é o seguinte: Suponha que você tenha um conjunto de números inteiros e deseje verificar se há algum número repetido neste conjunto.

Para resolver esse problema, podemos aplicar o pensamento computacional da seguinte maneira:

- **Definir o problema:** Verificar se há algum número repetido em um conjunto de números inteiros.
- **Dividir o problema em partes menores:** a) Ler os números do conjunto; b) Verificar se há algum número repetido comparando cada número com os demais números do conjunto; c) Se houver um número repetido, retornar "sim", caso contrário, retornar "não".
- **Desenvolver um algoritmo:** Para resolver cada uma das partes menores, considerando as restrições do problema.
- **Implementar o algoritmo:** Utilizar uma linguagem de programação.
- **Testar o programa:** Verificar se ele identifica corretamente se há ou não algum número repetido no conjunto.
- **Refinar o algoritmo:** Será necessário, para melhorar sua eficiência ou precisão.

```
programa
{
	funcao inicio()
	{
		inteiro i, n, numero, num_maior

		escreva("Digite a quantidade de números: ")
		leia(n)

		escreva("\n", "Digite o 1º número: ")
		leia(num_maior)

		para(i=2; i<=n; i++)
		{
			escreva("\n", "Digite o ", i, "º número: ")
			leia(numero)

			se(numero > num_maior)
				num_maior = numero
		}

		escreva("\n", "O maior número é: ", num_maior, "\n")
	}
}
```

**Quadro 24 - Código exemplo para encontrar o maior número em um conjunto de números inteiros no Portugol Studio.**

**Fonte:** O autor

**Descrição da figura:** Neste exemplo, o algoritmo lê a quantidade de números a serem inseridos pelo usuário, lê o primeiro número e armazena-o na variável "num_maior". Em seguida, utiliza um loop "para" para ler os números restantes e comparar cada um com o número armazenado em "num_maior". Se o número lido for maior do que o número armazenado em "num_maior", então ele se torna o novo número "num_maior". Ao final do loop, a variável "num_maior" armazena o maior número do conjunto, que é impresso na tela por meio do comando “escreva” com a mensagem "O maior número é: 6".

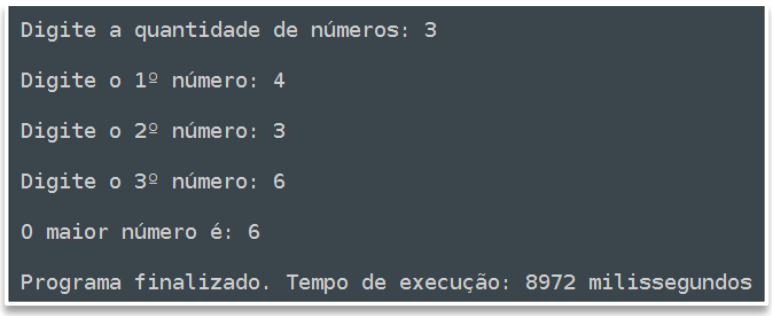



**Figura 44 - Saída do exemplo para encontrar o maior número em um conjunto de números inteiros no Portugol Studio.**

**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/

**Descrição da figura:** É exibida a saída que foi calculada no programa para encontrar o maior número em um conjunto de números inteiros no Portugol Studio. No caso, é exibida a mensagem "O maior número é: 6".

## 4.2 Organizar os dados de forma lógica

Organizar os dados de forma lógica é um conceito fundamental do pensamento computacional. Para fazer isso, é preciso que você siga alguns passos:

- **Identifique o problema:** Antes de começar a organizar os dados, é importante entender o problema que você está tentando resolver e o tipo de dados envolvidos. Certifique-se de que você tem uma compreensão clara do que está sendo solicitado e do que é necessário para solucionar o problema.
- **Determine as variáveis:** Depois de identificar o problema, determine quais são as variáveis envolvidas. Essas variáveis podem incluir informações de entrada, como dados de entrada do usuário ou dados de sensores, bem como informações de saída, como resultados de cálculos ou respostas para o usuário.
- **Defina a estrutura de dados:** Em seguida, defina a estrutura de dados que será usada para armazenar e manipular as informações. As estruturas de dados mais comuns incluem arrays, listas, dicionários e bancos de dados. Certifique-se de escolher uma estrutura de dados que seja apropriada para as informações envolvidas.
- **Escolha um algoritmo:** Depois de definir a estrutura de dados, escolha um algoritmo que seja apropriado para o problema. Lembrando que algoritmo é uma sequência de passos que o computador seguirá para resolver o problema. Certifique-se de escolher um algoritmo que funcione bem com a estrutura de dados escolhida.
- **Teste e ajuste:** Por fim, teste o sistema para garantir que ele esteja funcionando corretamente. Se houver erros ou problemas, ajuste o algoritmo e a estrutura de dados até que o problema seja resolvido.

Um exemplo simples de como organizar os dados de forma lógica usando o pensamento computacional é criar um programa para calcular a média ponderada de duas notas. Nesse caso, a estrutura seria organizada em três partes: a entrada dos dados, o cálculo da média e a saída dos dados.

```
programa
{
	funcao inicio()
	{
		real notas[2], soma = 0.0, peso = 0.0
		inteiro i
		const inteiro p = 2

		// Entrada dos dados
		para(i = 0; i<2; i++)
		{
			escreva("Digite o valor da nota: ")
			leia(notas[i])
			escreva("\n")
		}

		// Cálculo dos dados
		para(i = 0; i<2; i++)
		{
			soma = soma + (notas[i]*p)
			peso = peso + p
		}

		// Saída dos dados
		escreva("A média ponderada das notas é: ", soma/peso, "\n")
	}
}
```

**Quadro 25 - Código exemplo de cálculo da média ponderada de duas notas no Portugol Studio.**
**Fonte:** O autor

**Descrição da figura:** Este exemplo apresenta um programa onde é declarado um vetor “notas” e duas variáveis do tipo real, a primeira variável é “soma” que receberá a soma das notas armazenadas no vetor “notas” e “peso” que receberá a soma dos pesos, também foram declaradas duas variáveis do tipo inteiro, “i” sendo a variável de controle para a leitura do vetor “notas” e “p” a constante com o valor do peso das notas. Em seguida é feita a entrada dos dados, os cálculos, e a saída por meio do comando escreva, que no caso do exemplo de cálculo da média ponderada de duas notas no Portugol Studio mostrará a mensagem “A média ponderada das notas é: 7.4”.

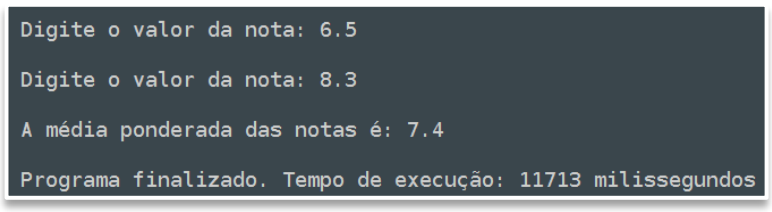



**Figura 45 - Saída do exemplo de cálculo da média ponderada de duas notas no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de cálculo da média ponderada de duas notas no Portugol Studio. No caso, é exibida uma mensagem "A média ponderada das notas é: 7.4".

## 4.3 Analisar os dados de forma lógica

Quando você analisa dados de forma lógica, você está examinando as informações de uma maneira organizada e seguindo um processo rigoroso e lógico para chegar a conclusões precisas e confiáveis.

Aqui está um exemplo simples de como analisar os dados de forma lógica: Suponha que você tenha coletado dados sobre a altura de 5 alunos de uma turma. Você deseja analisar esses dados para entender melhor a distribuição de altura na turma.

Comece organizando os dados em ordem crescente para ter uma ideia geral dos valores. Por exemplo, se os dados forem os seguintes: 1,76m, 1,68m, 1,82m, 1,64m, 1,85m. Organize-os em ordem crescente e informe a idade média da turma.

Com essas etapas, você poderá analisar de forma lógica os dados e chegar a conclusões precisas e confiáveis sobre a média de altura da turma de alunos. Vamos ver o exemplo na prática.

```
programa
{
        funcao inicio()
        {
                real altura[5] = {1.76, 1.68, 1.82, 1.64, 1.85}, auxiliar, soma = 0.0
                inteiro i, j

                para (i = 0; i <= 4; i++)
                        para (j = 0; j <= 4; j++)
                                se (altura[i] <= altura[j])
                                {
                                        auxiliar = altura[i]
                                        altura[i] = altura[j]
                                        altura[j] = auxiliar
                                }

                escreva("Vetor em ordem crescente: ")
                para (i = 0; i <= 4; i++)
                {
                        escreva("|", altura[i], “m”)
                        soma = soma + altura[i]
                }

                escreva("|","\n\n")
                escreva("A média de altura da turma é: ", soma/5, "m\n")
        }
}
```

**Quadro 26 - Código exemplo de ordenação de um vetor no Portugol Studio.**
**Fonte:** O autor

**Descrição da figura:** Este exemplo apresenta um programa onde é declarado um vetor “altura” com 5 posições preenchidas pelas respectivas alturas dos alunos de uma turma, são declaradas também duas variáveis do tipo real, “auxiliar” que armazenar temporariamente o valor do vetor e a variável “soma” que armazena a soma da altura de

todos os alunos, em seguida foram declaradas duas variáveis do tipo inteiro, “i” e “j” que são variáveis de controle para a leitura do vetor, em seguida dois laços para-faça farão a classificação do vetor dependendo dos valores testados no comando “se-entao”, em seguida outro laço “para-faca” varre o vetor para mostrar na tela o seu conteúdo classificado em ordem crescente, em seguida a variável “soma” armazena a soma das alturas para exibir a média de altura da turma por meio do comando “escreva”, que no caso do exemplo de ordenação de um vetor no Portugol Studio mostrará a mensagem “A média de altura da turma é: 1.75m”.

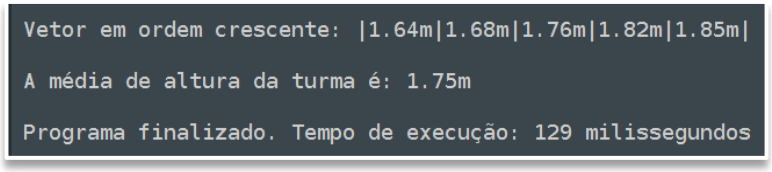



**Figura 46 - Saída do exemplo de ordenação de um vetor no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida a saída que foi calculada no programa do exemplo de ordenação de um vetor no Portugol Studio. No caso, é exibida a mensagem "A média de altura da turma é: 1.75m".

## 4.4 Representar dados por meio de abstrações

Abstração é muito importante no pensamento computacional, porque é uma forma de deixar um problema complexo mais fácil de entender. Basicamente, você consegue simplificar a complexidade e focar apenas no que é importante para resolver o problema. Segue um exemplo de como representar dados por meio de abstrações:

```
programa
{
	funcao inicio()
	{
		inteiro idade1, idade2
		cadeia nome1, nome2

		escreva("Digite o nome da primeira pessoa: ")
		leia(nome1)
		escreva("Digite a idade da primeira pessoa: ")
		leia(idade1)

		escreva("\n", "Digite o nome da segunda pessoa: ")
		leia(nome2)
		escreva("Digite a idade da segunda pessoa: ")
		leia(idade2)

		se(idade1 > idade2)
			escreva("\n", "A pessoa mais velha é: ", nome1)
		senao
			se (idade2 > idade1)
				escreva("\n", "A pessoa mais velha é: ", nome2)
			senao
				escreva("\n", "As pessoas têm a mesma idade")

		escreva("\n")
	}
}
```

**Quadro 27 - Código exemplo no Portugol Studio que faz a comparação de duas idades e informa qual o nome da pessoa mais velha.**
**Fonte:** O autor
**Descrição da figura:** Nesse código, usamos abstrações para simplificar a complexidade dos dados de idade e nome de duas pessoas e obter uma visão geral de quem é a pessoa mais velha. Primeiro, declaramos as variáveis necessárias para armazenar o nome e a idade de cada pessoa. Em seguida, pedimos ao usuário para inserir o nome e a idade de cada pessoa. Finalmente, usamos a estrutura condicional "Se...Então...Senão" para verificar quem tem a idade mais alta e exibir o nome dessa pessoa na saída. Com essas abstrações, podemos rapidamente identificar a pessoa mais velha entre as duas sem ter que verificar cada idade individualmente.

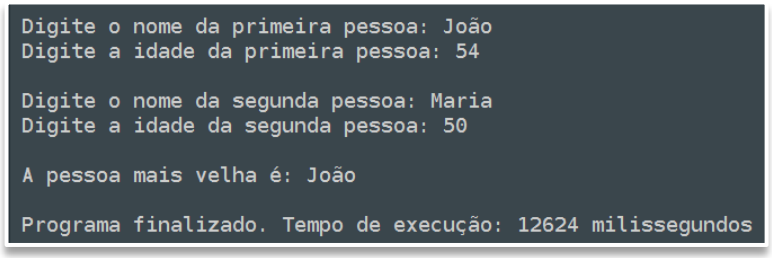



**Figura 47 - Saída do exemplo de como representar os dados por meio de abstrações no Portugol Studio.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida uma mensagem de saída no programa do exemplo de como representar os dados por meio de abstrações no Portugol Studio. No caso, é exibida a mensagem "A pessoa mais velha é: João".

## 4.5 Modelos e Simulações

Você pode utilizar modelos e simulações para entender melhor sistemas reais ou hipotéticos. Um modelo é uma representação simplificada e abstrata de um sistema, enquanto uma simulação é um programa de computador que executa um modelo em tempo real para permitir que você visualize e interaja com o sistema simulado.

Veja abaixo um exemplo de modelo e simulação de um programa de lançamento de um dado. O modelo descreve a probabilidade de cada face do dado ser sorteada. O programa de simulação executa esse modelo em tempo real, permitindo que o usuário visualize e interaja com o lançamento do dado.

```
programa
{
	inclua biblioteca Util --> u

	funcao inicio()
	{
		inteiro num

		escreva("Modelo e Simulação do Jogo de um Dado de 6 Faces", "\n")
		num = u.sorteia(1, 6)
		escreva("\n", "Número sorteado: ", num, "\n")
	}
}
```

**Quadro 28 - Código exemplo no Portugol Studio de um programa de simulação que executa o lançamento de um dado de 6 faces.**
**Fonte:** O autor
**Descrição da figura:** Nesse código, incluímos a biblioteca Util que contém a função “sorteia” que irá fazer o cálculo do possível número mínimo e máximo para o lançamento de um dado de 6 faces. Em seguida é declarada uma variável “num” do tipo inteiro que receberá o valor sorteado, ao final o programa por meio do comando “escreva” mostra a mensagem com o número sorteado na jogada.

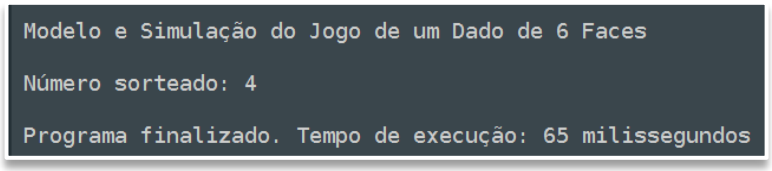



**Figura 48 - Saída do exemplo no Portugol Studio de um programa de simulação que executa o lançamento de um dado de 6 faces.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida uma mensagem de saída no programa do exemplo no Portugol Studio de um programa de simulação que executa o lançamento de um dado de 6 faces. No caso, é exibida a mensagem "Número sorteado: 4".

## 4.6 Automatizar soluções por meio do pensamento algorítmico

Quando você automatiza soluções usando o pensamento algorítmico, você está usando uma abordagem super eficaz para resolver problemas complexos. A ideia é criar um conjunto de instruções lógicas que descrevam as etapas necessárias para alcançar um determinado objetivo.

O pensamento algorítmico é superlegal porque ele permite que as soluções sejam aplicáveis a uma grande variedade de situações. Isso significa que uma vez que você cria uma solução algorítmica, você pode usá-la repetidamente, fazendo pequenas modificações para lidar com diferentes problemas.

Segue abaixo um exemplo de um programa em Portugol Studio que automatiza uma tarefa simples utilizando pensamento algorítmico:

```
programa
{
	funcao inicio()
	{
		inteiro contador = 1

		enquanto(contador <= 10)
		{
			escreva(contador)
			escreva(" ")
			contador = contador + 1
		}
		escreva("\n")
	}
}
```

**Quadro 29 - Código exemplo no Portugol Studio de um programa para automatizar a tarefa de contar até 10.**
**Fonte:** O autor
**Descrição da figura:** Nesse exemplo, o programa utiliza o pensamento algorítmico para automatizar a tarefa de contar até dez. Ele utiliza um laço "enquanto" para executar a tarefa enquanto a condição "contador <= 10" for verdadeira. Dentro do laço, o programa escreve o valor atual do contador na tela, seguido de um espaço em branco, e então incrementa o contador em 1. Quando o contador chega ao valor de 10, o laço termina e o programa encerra.

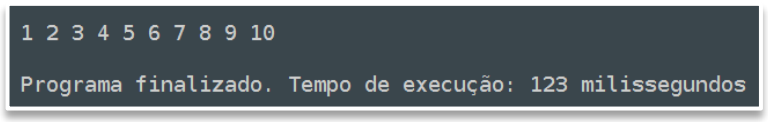



**Figura 49 - Saída do exemplo no Portugol Studio de um programa para automatizar a tarefa de contar até 10.**
**Fonte:** LITE - Univali em http://lite.acad.univali.br/portugol/
**Descrição da figura:** É exibida uma mensagem no Portugol Studio de um programa para automatizar a tarefa de contar até 10. No caso, é exibida a mensagem "1 2 3 4 5 6 7 8 9 10".

**VIDEOAULA!**
Não deixe de assistir à videoaula, lá vou explicar os princípios ensinados nessa Unidade.

Agora que você assistiu à videoaula e estudou esta Unidade, acesse o **Fórum de Dúvidas: Atividade Prática Semanal da Unidade 4** e siga as orientações para a realização da atividade prática.

Para entender quais tópicos foram apresentados na Unidade,
**acesse o podcast** disponível no Ambiente Virtual de Aprendizagem (AVA).

## Conclusão

Então estudante, chegamos ao final de nossa disciplina, durante essas quatro Unidades abordamos a importância das habilidades de lógica de programação e do pensamento computacional, não apenas para programadores, mas para todos os que desejam compreender e utilizar as tecnologias em diferentes áreas.

Você aprendeu que para desenvolver essas habilidades, é necessário aprender a pensar de forma abstrata, ou seja, desconsiderar os detalhes menos importantes e se concentrar apenas no essencial, além de praticar e buscar soluções para problemas diversos usando as boas práticas do pensamento computacional.

Essa disciplina foi dividida em quatro Unidades, que são fundamentais para formar uma base sólida e aplicável em diferentes linguagens e ambientes computacionais. É importante ressaltar que o aprendizado dessas habilidades é um processo contínuo e que você estará sempre evoluindo, tenha paciência com você mesmo.

Você está se preparando para ser um solucionador de problemas**, esteja disposto a se desafiar e persistir, sempre buscando aprender e se desenvolver. Parabéns, não desista, você vai alcançar o sucesso por meio de seus esforços**, boa sorte, bons estudos e até a próxima disciplina.

## Referências


PAIVA, Severino. Introdução A Programação e ao Pensamento Computacional. 1ª Ed. Rio de Janeiro: Ciência Moderna, 2021.

Pensamento Computacional e Engenharia de Software: Primeiros passos em direção a uma proposta de sistematização de resolução de problemas - Júlia de Avila dos Santos, Simone André da Costa Cavalheiro, Luciana Foss, Leomar S. da Rosa Jr.

Programa Educacional em Saúde Digital da Universidade Federal de Goiás - Pensamento Computacional - Organizadores: Ana Laura de Sene Amâncio Zara, Fábio Nogueira de Lucena, Rejane Faria Ribeiro Rota, Renata Dutra Braga, Rita Gorete Amaral, Sheila Mara Pedrosa, Silvana de Lima Vieira dos Santos e Taciana Novo Kudo.

Pensamento Computacional - Desenvolvido no âmbito do projeto UFRGS/MEC TED 676559/SAIFI - Avaliação de Tecnologias Educacionais com a participação direta de: Rosa Maria Vicari, Álvaro Moreira, Paulo Blauth Menezes - Com colaboração de: Crediné Solva de Menezes, Daltro Nunes e Maria Aparecida C. Livi - 2018

SOUZA, Marco A Furlan - Marcelo Marques Gomes - Marcio Vieira Soares - Ricardo Concilio. Algoritmos e Lógica de Programação. 3ª. Ed. São Paulo: Cengage Learning, 2019.

BHARGAVA, Aditya Y. Entendo Algoritmos: Uma Guia Ilustrado Para Programadores e Outros Curiosos. 1ª ed. São Paulo: Novatec Editora, 2017.

MANZANO, José Augusto N. G. e Jayr Figueiredo de Oliveira, Algoritmos: Lógica Para Desenvolvimento de Programação de Computadores. 29ª Ed. São Paulo: Editora Érica, 2019.

PASQUAL JUNIOR, Paulo Antonio. Pensamento computacional e tecnologias: reflexões sobre a educação no século XXI. Ed. Caxias do Sul, RS: Educs, 2020.

De Moraes, Paulo Sergio. Lógica de Programação. São Paulo: Curso Básico de Lógica de Programação Unicamp - Centro de Computação - DSC, 2000.

Artero, Marcio Aparecido. Algoritmos e lógica de programação. Londrina: Editora e Distribuidora Educacional S.A., 2018.

De Souza, Bruno Jefferson; Dias Júnior, José Jorge Lima; Formiga, Andrei de Araújo. Introdução a Programação. João Pessoa: Editora da UFPB, 2014.

De Carvalho, Flávia Pereira. Apostila de Lógica de Programação- ALGORITMOS -. Taquara: FACCAT-FIT - Faculdade de Informática de Taquara - Curso de Sistemas de Informação, 2007.

Departamento de Computação e Automação. Algoritmo e Lógica de Programação Algoritmos - Parte 1. Natal: Universidade Federal do Rio Grande do Norte - Centro de Tecnologia, 2004.

Steinmetz, Ernesto Henrique Radis; Fontes, Roberto Duarte. CARTILHA LÓGICA DE PROGRAMAÇÃO. Brasília: Instituto Federal de Brasília, Editora IFB, 2013.

Sebesta, Robert W. CONCEITOS DE LINGUAGENS DE PROGRAMAÇÃO. Porto Alegre: Editora BOOKMAN, 2011.

Material de Apoio do Portugol Studio, LITE - laboratório de inovação tecnológica na educação da Universidade do Vale do Itajaí (Univali) em http://lite.acad.univali.br/portugol/

Almeida Martins, Luiz Gustavo. Apostila de Introdução a Algoritmos. UFU - Universidade Federal de Uberlândia Faculdade de Computação.

Apostila de Lógica de Programação. Mestrado Profissional em Ensino e suas Tecnologias - 2019 Campos Centro -IFFluminense.

Produto Educacional - Cartilha com Atividades Desplugadas para o Ensino Médio - Universidade do Estado de Santa Catarina - UDESC - Centro de Ciências Tecnológicas - CCT - Programa de pós-graduação em Ensino de Ciências, Matemática e Tecnologias - PPGECMT.

## Minicurrículo do Professor

### Tércio Cavalcanti da Silva Santos

Possui graduação em Sistemas de Informação, pós-graduação em Engenharia e Arquitetura de Software, pós-graduação em Ciência de Dados, pós-graduação em Educação Digital e é pós-graduando em Inteligência Artificial. Possui curso de “Inovação Digital na Educação” na modalidade de extensão profissional, possui curso de “Mindset Digital na Educação” na modalidade de extensão profissional e possui também o curso MCSE (Microsoft Certified Systems Engineer). Atualmente trabalha na EMPREL (Empresa Municipal de Informática do Recife) como Analista de Processos de TI e é Professor Formador e Conteudista (EAD) na ETE - Professor Antônio Carlos Gomes da Costa - ETEPAC.