



# Algoritmos

● Criado em @25 de dezembro de 2025 15:57

≡ Tags

## O que é um algoritmo?

<https://youtu.be/Ohi-MGIBEws?si=X1xXT6yhtAOYFSiC>

👉 Algoritmo é uma receita de bolo para resolver um problema.

Ele é:

- **Passo a passo**
- **Em ordem**
- **Finito** (tem começo e fim)
- **Sem ambiguidade** (cada passo é claro)

Se você mudar a ordem ou pular um passo, o resultado pode dar errado.

## Exemplo do dia a dia: atravessar a rua

**Problema:** atravessar a rua sem ser atropelado

**Solução:** seguir uma sequência lógica de ações

---

### Algoritmo correto (forma clara)

```
AlgoritmoAtravessarRua
```

```
    Olhar para a esquerda
```

```
    Olhar para a direita
```

```
    Se estiver vindo carro
```

```
        Não atravesse
```

```
    senão
```

```
        Atravesse
```

```
    Fim-Se
```

```
    Fim-Algoritmo
```



**Por que isso faz sentido?**

- Primeiro você **coleta informações** (olhar)
- Depois você **toma uma decisão**
- Só então você **executa a ação**

Isso é **lógica**.

---

### Mesmo algoritmo, escrito de outra forma (também correto)

```
AlgoritmoAtravessarRua
```

```
    Olhar para a esquerda
```

```
    Olhar para a direita
```

```
    Se não estiver vindo carro
```

```
        Atravesse
```

```
    senão
```

```
        Não atravesse
```

Fim-Se  
Fim-Algoritmo

👉 Continua certo porque:

- A **decisão acontece antes da ação**
- A condição é clara
- O resultado final é o mesmo

## Maneira ERRADA (por quê?)

```
AlgoritmoAtravessarRua
Atravesse
Se não estiver vindo carro
Olhar para a direita
senão
Olhar para a esquerda
Fim-Se
NãoAtravesse
Fim-Algoritmo
```

✖ Erros lógicos aqui:

1. Você **atravessa antes de olhar**
2. A decisão vem **depois da ação**
3. O algoritmo se contradiz:
  - manda atravessar
  - depois manda não atravessar

💥 Na vida real: atropelamento

💥 No computador: erro lógico (bug)

## Regra de ouro pra você lembrar sempre

|  Todo algoritmo segue este fluxo:

- 1 Entrada / observação (olhar, receber dados)
- 2 Processamento / decisão (se, senão)
- 3 Saída / ação (atravessar ou não)

Se inverter isso → **algoritmo errado**

## Dica final (pra fixar de vez)

Sempre se pergunte:

|  “Esse passo depende de alguma informação anterior?”

Se sim, **ele não pode vir antes.**

## Vídeo:

<https://youtu.be/8mei6uVttho?si=kJm58-FkMAQgERo1>

# Introdução aos Algoritmos e Variáveis

## Apresentação da aula

Nesta aula, você aprenderá **o que são algoritmos computacionais**, sua **importância no desenvolvimento de sistemas** e como começar a estruturar sua **lógica de programação**.

Você irá conhecer ferramentas e formas de representação que ajudam a organizar o pensamento lógico, como:

-  **Fluxogramas**
-  **Diagramas de Nassi-Shneiderman**
-  **Pseudocódigo (Portugol)**

Além disso, será apresentado o **Visualg**, uma ferramenta essencial para **praticar lógica de programação**, muito utilizada por iniciantes e estudantes de TI.

---

## O que são Algoritmos Computacionais?

**Algoritmos computacionais** são **conjuntos de passos organizados e executados em uma ordem correta** para realizar uma determinada tarefa.

Esses passos são seguidos por:

-  **Um módulo processador** → qualquer coisa capaz de executar o algoritmo (computador, celular, sistema, aplicativo).
  -  **Usuários** → você e todas as outras pessoas que utilizam o sistema.
- 👉 Sempre que você usa um sistema, existe um algoritmo funcionando por trás.

## Exemplos de algoritmos no dia a dia

- Fazer um **PIX** no aplicativo do banco
- Entrar no **Instagram**
- Usar um **sistema de gestão** como a Hiper
- Criar anotações no **Notion**

Tudo isso funciona porque alguém criou **algoritmos** para essas tarefas.

---

## Como criar um algoritmo?

O processo acontece em duas etapas:

**1 Criar a lógica de programação**

(pensar nos passos e decisões)

**2 Transformar essa lógica em uma linguagem de programação**

(Portugol, JavaScript, Python, etc.)

Quando isso acontece, temos um **sistema funcionando**.

---

## Exemplo de Fluxograma (conceito)

Um fluxograma representa visualmente os passos de um algoritmo.

Exemplo de ideia (em texto):

```
Início
↓
Mostrar"Olá, Mundo!"
↓
Mostrar"Me livrei da maldição"
↓
Fim
```

👉 O fluxograma ajuda a **visualizar a ordem** antes de escrever código.

## Exemplo em Portugol

```
- Área dos algoritmos ( Edição do código fonte ) -> N
  1 algoritmo "primeiro"
  2 var
  3   |
  4 inicio
  5       Escreval("Olá, Mundo!")
  6       Escreva("Me livrei da maldição")
  7 fimalgoritmo
```

```
algoritmo "primeiro"
var

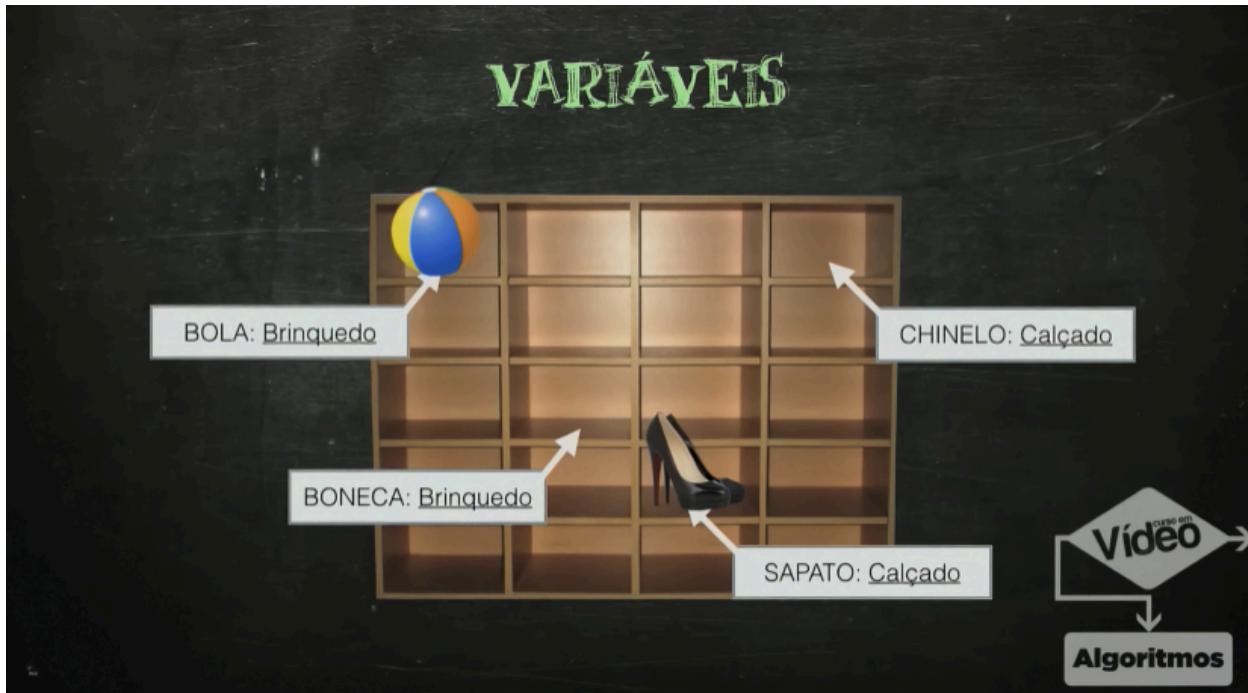
inicio
    Escreval("Olá, Mundo!")
    Escreva("Me livrei da maldição")
fimalgoritmo
```

## Comandos de saída

- `Escreva("texto")` → escreve algo na tela
- `Escreval("texto")` → escreve algo na tela **e pula linha**

# O que são Variáveis?

<https://youtu.be/SEQ57illdy4?si=WnuZbfJHIXh0et6U>



**💡 Variável é um espaço na memória do computador usado para guardar valores.**

Uma boa forma de entender é imaginar **uma estante com prateleiras vazias**:

- 📦 Cada prateleira tem um **nome**
- 📦 Você pode **guardar algo dentro**
- 📦 Pode **trocar o conteúdo**
- 📦 Pode **ver o que está guardado**

É exatamente assim que uma variável funciona.

👉 Uma variável **não guarda dois valores ao mesmo tempo**.

Se quiser mudar o valor, você **substitui o anterior**.

## Nome e tipo da variável

Assim como numa estante:

-  **BOLA**: Brinquedo
-  **SAPATO**: Calçados

Na programação, sempre usamos:

identificador :tipo

## O que é um identificador?

 Identificador é o nome da variável.

Existem **6 regras** para criar identificadores:

1. Deve começar com **letra**
2. Pode conter **letras e números**
3.  Não pode usar símbolos (exceto )
4.  Não pode conter espaços
5.  Não pode conter acentos
6.  Não pode ser uma **palavra reservada** (palavras reservadas pela própria linguagem)

Dica. Não faça **abreviações**, use nomes intuitivos

## Exemplos de identificadores

| Identificador | Correto?  | Motivo            |
|---------------|---|-------------------|
| Nota1         |  | Nome válido       |
| Média         |  | Acento            |
| Salário Bruto |  | Espaço e acento   |
| 9dade         |  | Começa com número |

| Identificador    | Correto? | Motivo             |
|------------------|----------|--------------------|
| Algoritmo        | ✗        | Palavra reservada* |
| Inicio_Algoritmo | ✓        | Nome válido        |

## O que são palavras reservadas?

- Área dos algoritmos ( Edição do código fonte ) -> N

```

1 algoritmo "primeiro"
2 var
3   |
4 inicio
5   Escreval("Olá, Mundo!")
6   Escreva("Me livrei da maldição")
7 finalgoritmo

```

🚫 Palavras reservadas são palavras que **já fazem parte da linguagem**.

No Portugol, por exemplo:

- `algoritmo`
- `var`
- `inicio`
- `fimalgoritmo`

👉 Você não pode usar essas palavras como identificadores.

## Exemplo de uso prático de Variável



Nesse caso quando o botão é pressionado é enviado para as variáveis usuário e telefone os dados colocados no formulário e exibe a mensagem de inscrição

realizada com os dados nas variáveis.

Falaremos melhor sobre comandos de entrada mais a frente.

O principal uso da variável é fazer reuso de valor já armazenado.

## Tipos de Dados

✖ **Observação:** usamos ponto (.) e não vírgula para números decimais, isso em qualquer linguagem.

✖ Em **Caractere** as regras dos identificadores não se aplica, podendo escrever qualquer coisa dentro das aspas.

- **Inteiro/Integer** → 1, 3, 5, 19, 0
- **Real/Float (Fracionados)** → 0.5, 5.0, 9.8, 77.3
- **Caractere/String** → "Marcus", "Algoritmos"
- **Lógico/Boolean** → Verdadeiro ou Falso
- **Vetor/Array** → ["Carlos", falso, "Cachorro", 65,5]

Vetor é uma variável composta, na maioria das linguagens ela é definida por colchetes [ ] e podemos adicionar valores de diversos tipos, lembrando que nos vetores a contagem dos números começa sempre com 0, a posição de cada valor nós chamamos de **índice**, veja o exemplo abaixo:



## Declaração de variáveis

```
var  
    identificador: tipo
```

## Exemplo:

Área dos algoritmos ( Edição do código fonte ) -> N

```
1 algoritmo "primeiro"
2 var
3     msg: caractere
4 inicio
5         Escreval("Olá, Mundo!")
6         Escreva("Me livrei da maldição")
7 fimalgoritmo
```

```
var  
    msg: caractere
```

👉 O computador cria um espaço chamado **msg** que só pode guardar **texto**.

## Atribuição de valor

| Áreas das variáveis de memória (Global) |      |      |               |
|---|------|------|---------------|
| Escopo                                  | Nome | Tipo | Valor         |
| GLOBAL                                  | MSG  | C    | "Olá, Mundo!" |
|   |      |      |               |

Para colocar algo dentro da variável, usamos **atribuição**:

```
msg ← "Olá mundo!"
```

📌 A seta significa:

| A variável msg recebe o valor "Olá mundo!"

```
rea dos argumentos ( Largura do e
1 algoritmo "primeiro"
2 var
3     msg: caractere
4 inicio
5         msg <- "Olá, Mundo!"
6         Escreval(msg)
7 fimalgoritmo
```

## Variável x texto literal

Escreva("msg")

🖨 Saída:

msg

Escreva(msg)

🖨 Saída:

Olá mundo!

The screenshot shows a software interface with two main windows. On the left, there is a code editor with the following pseudocode:

```
rea dos argumentos ( Largura do e
1 algoritmo "primeiro"
2 var
3     msg: caractere
4 inicio
5         msg <- "Olá, Mundo!"
6         Escreval(msg)
7 fimalgoritmo
```

On the right, there is a terminal window titled "Console simulando o modo texto do MS-DOS". It displays the output of the algorithm execution:

```
Olá, Mundo!
>>> Fim da execução do programa !
```

👉 Aspas → texto literal

👉 Sem aspas → conteúdo da variável

## Misturando texto e variável

```
Escreva("Mensagem:", msg)
```

🖨 Saída:

```
Mensagem: Olá mundo!
```

The screenshot shows a VisualG IDE interface. On the left, there is a code editor with the following pseudocode:

```
:algoritmo "primeiro"
:var
:    msg: caractere
:inicio
;    msg <- "Olá, Mundo!"
;    Escreval("mensagem ", msg)
:fimalgoritmo
```

On the right, there is a terminal window titled "Console simulando o modo texto do MS-DOS". It displays the output of the program:

```
mensagem Olá, Mundo!
>>> Fim da execução do programa !
```

## Reforçando

- Variável = espaço na memória
- Identificador = nome da variável
- Tipo = o que ela pode guardar
- Atribuição = colocar valor

## Vídeo

```
https://youtu.be/M2Af7gkbbro?si=8DgWZhfTEe\_yc429
```

## Comandos de Entrada e Saída – Visualg

### 📩 Comandos de entrada

São usados para **receber dados do usuário** (informações digitadas no teclado).

| Entrada = usuário → programa

O principal comando de entrada que usamos no Visualg é o **Leia**.

Em **todas as linguagens de programação** existe algum tipo de comando de entrada, pois sem isso não há interatividade.



## Comandos de saída

São usados para **mostrar informações na tela**.

| Saída = programa → usuário

No Visualg, usamos principalmente o **Escreva**.



## Exemplo com atribuição direta

```
1 Algoritmo "MeuNome"
2
3 Var
4   Nome: caractere
5
6
7 Inicio
8
9 Escreva ("Digite seu nome ")
10  Leia (Nome)
11 Escreva ("Muito prazer ", Nome)
12
13 Fimalgoritmo
```

c:\ Console simulando o modo texto do MS-DOS

```
Digite seu nome Marcus
Muito prazer Marcus
>>> Fim da execução do programa !
```

Neste exemplo, **não há interação com o usuário**. O valor é definido diretamente no código.

```
Algoritmo "MeuNome"
```

```
Var
```

```
  Nome: Caractere
```

```
Inicio
```

```
    Nome ← "Gustavo"
```

```
    Escreva ("Muito prazer ", Nome)
```

```
FimAlgoritmo
```

📌 Aqui acontece:

- A variável `Nome` recebe diretamente o valor "Gustavo"
- O programa apenas mostra a mensagem na tela

## 🧠 Exemplo com entrada de dados (Leia)

Agora o programa **interage com o usuário**.

```
Algoritmo "MeuNome"
```

```
Var
```

```
    Nome: Caractere
```

```
Inicio
```

```
    Escreva ("Digite seu nome ")
```

```
    Leia (Nome)
```

```
    Escreva ("Muito prazer ", Nome)
```

```
FimAlgoritmo
```

📌 Fluxo do programa:

1. Mostra a mensagem pedindo o nome
2. O usuário digita o nome
3. O valor digitado é armazenado na variável `Nome`
4. O programa usa esse valor na mensagem final

# Estrutura básica de um algoritmo

## 1 Cabeçalho do algoritmo

Algoritmo "MeuNome"

Serve apenas para **identificação do programa**.

Não executa nenhuma ação.

👉 Pense como o título de um trabalho.

## 2 Declaração de variáveis

Var

Nome: Caractere

📌 O que foi feito:

- Criada a variável `Nome` (identificador)
- Definido o tipo `Caractere`

⚠️ No Visualg, `Caractere` funciona como **texto (string)**:

- Palavras
- Nomes
- Frases

👉 O programa reserva um espaço na memória para guardar texto.

## 3 Início do programa

Início

Aqui o programa **começa a executar de verdade**.

Tudo antes disso é apenas preparação.

## 4 Escreva – saída de dados

Escreva ("Digite seu nome ")

- Apenas **mostra texto na tela**
- Não lê nada do usuário

## 5 Leia – entrada de dados

Leia (Nome)

📌 O que acontece:

- O programa espera o usuário digitar algo
- Quando o usuário pressiona Enter
- O valor digitado é armazenado na variável `Nome`

Exemplo interno:

`Nome = "USUARIO"`

## 6 Escreva com texto + variável

Escreva ("Muito prazer ", Nome)

Aqui ocorre uma **concatenação**:

- Texto fixo
- Valor da variável

⚠ Variáveis **não usam aspas**.

Aspas indicam texto literal.

## 7 Fim do algoritmo

Fimalgoritmo

O programa é encerrado.

## Trabalhando com números

### Exemplo: entrada de dois números

Algoritmo "MeuNome"

Var

N1, N2: Inteiro

Inicio

Escreva ("Informe um número ")

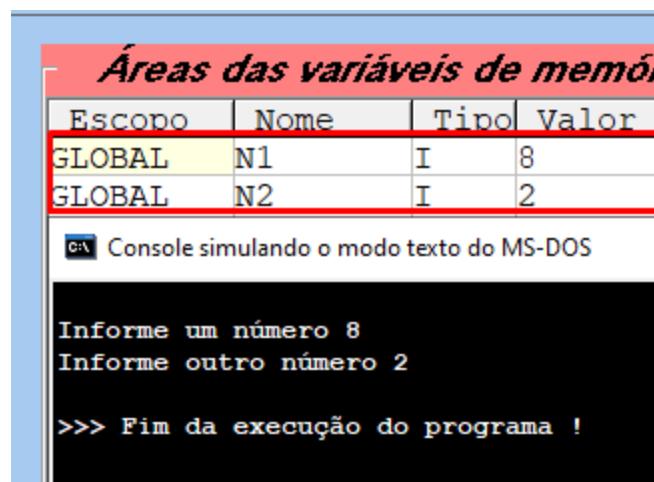
Leia (N1)

Escreva ("Informe outro número ")

Leia (N2)

FimAlgoritmo

 Os valores digitados ficam armazenados em **N1** e **N2**.



The screenshot shows a simulation interface. At the top, a table titled "Áreas das variáveis de memória" displays two global variables: N1 with value 8 and N2 with value 2. Below the table, a text area labeled "Console simulando o modo texto do MS-DOS" shows the following interaction:

```
Informe um número 8
Informe outro número 2
>>> Fim da execução do programa !
```

# Soma de dois números

Área dos algoritmos ( Edição do código-fonte ) -> Nome do arquivo

```
1 Algoritmo "MeuNome"
2
3 Var
4   N1, N2, S: Inteiro
5
6
7
8 Inicio
9
10 Escreva ("Informe um número ")
11   Leia (N1)
12 Escreva ("Informe outro número ")
13   Leia (N2)
14   S <- N1 + N2
15 Escreva (S)
16
17 FimAlgoritmo
```

c:\ Console simulando o modo texto do MS-DOS

```
Informe um número 8
Informe outro número 2
10
>>> Fim da execução do programa !
```

Algoritmo "Valores"

Var

N1, N2, S: Inteiro

Inicio

Escreva ("Informe um número ")

Leia (N1)

Escreva ("Informe outro número ")

Leia (N2)

S ← N1 + N2

Escreva ("O resultado da soma é ", S)

FimAlgoritmo

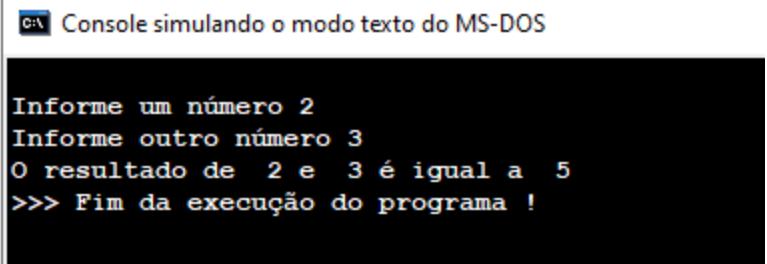
📌 A variável **s** recebe o resultado da soma.

## Saída mais detalhada

Escreva ("O resultado de ", N1, " e ", N2, " é igual a ", S)

- ✓ O comando `Escreva` aceita **vários valores separados por vírgula**.
- ✓ Espaços devem ser colocados **dentro das aspas**.

```
Algoritmo "MeuNome"  
Var  
    N1, N2, S: Inteiro  
  
Inicio  
    Escreva ("Informe um número ")  
    Leia (N1)  
    Escreva ("Informe outro número ")  
    Leia (N2)  
    S <- N1 + N2  
    Escreva ("O resultado de ", N1, " e ", N2, " é igual a ", S)  
  
Fimalgoritmo
```



```
Console simulando o modo texto do MS-DOS  
  
Informe um número 2  
Informe outro número 3  
O resultado de 2 e 3 é igual a 5  
>>> Fim da execução do programa !
```

## Operadores de Comparação

[https://youtu.be/2l1Hz3U\\_yx0?si=FzxZZGexmC9TVwn5](https://youtu.be/2l1Hz3U_yx0?si=FzxZZGexmC9TVwn5)

## Operadores Lógicos – Visualg (Anotação)

## 📌 O que são operadores lógicos?

São usados para **comparar valores** e **tomar decisões**.

Eles sempre retornam **VERDADEIRO** ou **FALSO**.

👉 Operadores lógicos são muito usados em **SE / SENO** e **ENQUANTO**.

---

## ◆ Operadores de comparação

Esses comparam **valores**:

| Operador | Significado    | Exemplo    | Resultado  |
|----------|----------------|------------|------------|
| =        | Igual          | $5 = 2$    | Falso      |
| <>       | Diferente      | $5 <> 2$   | Verdadeiro |
| >        | Maior          | $5 > 2$    | Verdadeiro |
| <        | Menor          | $5 < 2$    | Falso      |
| $\geq$   | Maior ou igual | $5 \geq 2$ | Verdadeiro |
| $\leq$   | Menor ou igual | $5 \leq 2$ | Falso      |

⚠ Lembrando que os símbolos desses operadores podem mudar de linguagem para linguagem, se apegue aos **FUNDAMENTOS**

⚠ Atenção: no Visualg  $=$  é **comparação**, não atribuição.

## Area dos algoritmos ( Edição do código fonte ) -> Nom

```
1 Algoritmo "EXERCICIO7"
2
3 Var
4     time1, time2, diferenca : inteiro
5
6 Inicio
7     Escreval("-----")
8     Escreval("          ANALISE FUTEBOL          ")
9     Escreval("-----")
10
11    Escreva("Quantos gols foi no TIME 1? ")
12    Leia(time1)
13
14    Escreva("Quantos gols foi no TIME 2? ")
15    Leia(time2)           ↗ abs quer dizer valor absoluto, o numero é
16
17    diferenca <- abs(time1 - time2)           sempre positivo
18
19    Escreval("DIFERENÇA: ", diferenca)
20
21    Se (diferenca = 0) então
22        Escreva("EMPATE")
23    Senao
24        Se (diferenca >= 3) então
25            Escreva("GOLEADA")
26        Senao
27            Escreva("JOGO NORMAL")
28        FimSe
29    FimSe
30
31 FimAlgoritmo
32
```

## ◆ Operadores lógicos principais

Esses ligam **condições**:

| Operador | Significado | Regra                     |
|----------|-------------|---------------------------|
| E        | AND         | Todas verdadeiras         |
| OU       | OR          | Pelo menos uma verdadeira |

| Operador | Significado | Regra           |
|----------|-------------|-----------------|
| NAO      | NOT         | Inverte o valor |

## Operador E

operador  ›

Você é humano E Você é Terrestre = **verdadeiro**

O inverno é frio E Polo norte é quente = **falso**

operador  ›

Se ao menos um valor for falso, o resultado final será falso.

O resultado apenas será verdadeiro, se todos os valores forem verdadeiros.

## Operador NÃO

operador **OU** »

Agua é molhada OU Fogo queima = **verdadeiro**

O circulo é redondo OU Quadrado é triangular = **verdadeiro**

A lua é quadrada OU O sol é frio = **falso**

operador **OU** »

Se ao menos um valor for verdadeiro,  
o resultado final será verdadeiro.

O resultado apenas será falso,  
se todos os valores forem falsos.

## Operador NÃO

## PRIMEIRO: o que é uma condição?

Uma condição é **uma pergunta** que só pode dar **SIM** ou **NÃO**.

Exemplos:

- 5 é maior que 3? → **SIM**
- 2 é maior que 10? → **NÃO**

O computador entende isso como:

- **SIM** = **VERDADEIRO**
- **NÃO** = **FALSO**

---

## AGORA: o que o NAO faz?

O **NAO** só faz **UMA** coisa:

Ele troca o **SIM** pelo **NÃO**  
e troca o **NÃO** pelo **SIM**.

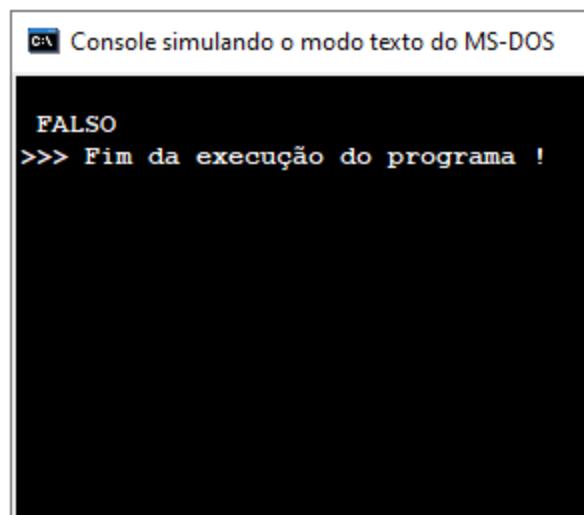
Só isso. Nada além disso.

**O NAO só inverte o resultado final APENAS ISSO**

---

### Área dos algoritmos ( Edição do código fonte ) -> Nome do arquivo

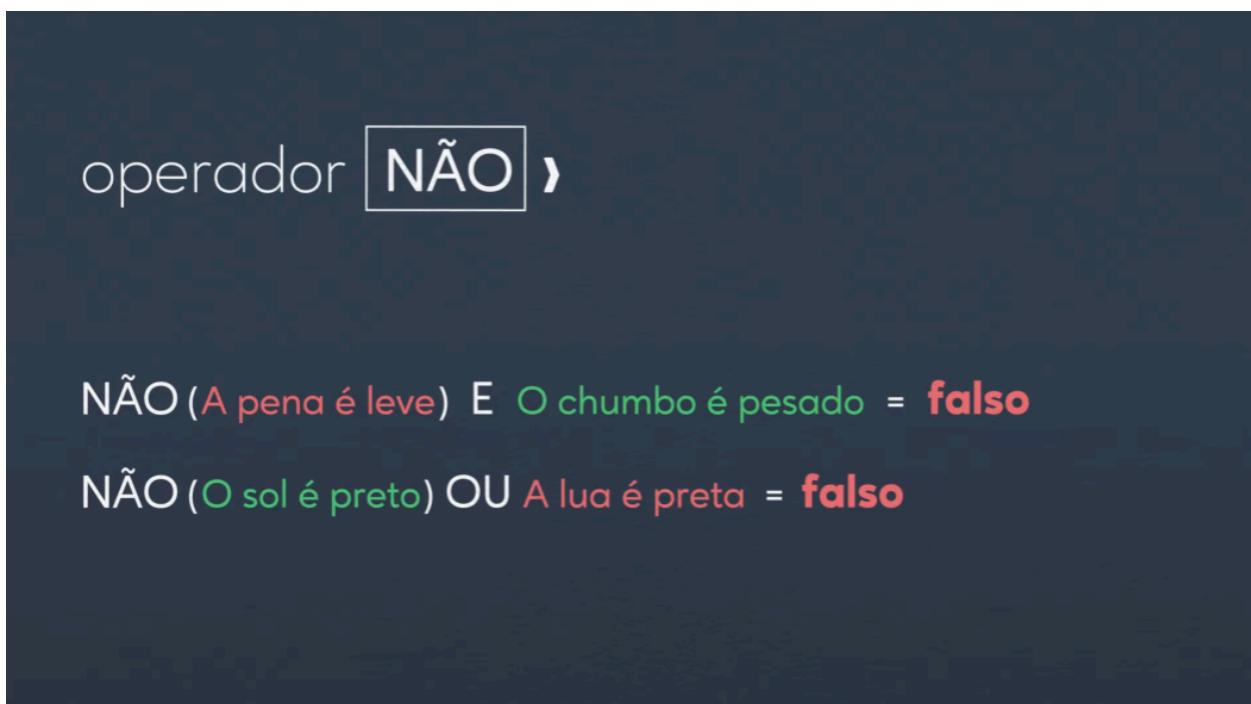
```
1 Algoritmo "semnome"
2
3 Var
4      A,B,C: inteiro
5
6
7 Inicio
8
9 A <- 2
10 B <- 3
11 C <- 5
12 Escreva (nao((A=B) ou (C>A)))
13
14 Fimalgoritmo
```



C:\> Console simulando o modo texto do MS-DOS  
FALSO  
>>> Fim da execução do programa !



Nesse exemplo só fez inverter o resultado final, se pressionar o interruptor vira falso e a luz apaga.



- No primeiro, os dois eram verdadeiros mas como o operador inverteu ele, então se tornou falso, por estarmos usando o operador E então os dois teriam que ser verdadeiros portando o resultado é falso.
- No segundo um dos dois teriam que ser verdadeiros para o operador OU e realmente era antes do operador NÃO inverter ele, então os dois se tornaram falsos portando o resultado é **falso**.



## EXEMPLO SEM COMPUTADOR (vida real)

Frase:

| "Está chovendo"

- Verdadeiro → está chovendo
- **NÃO** verdadeiro → **não** está chovendo

O NAO só coloca um "**não**" na frente.

---



## AGORA COM NÚMERO (bem devagar)

### Exemplo 1

Pergunta:

| 5 é maior que 3?

Resposta:

| SIM

Agora aplica o **NÃO**:

| NÃO é verdade que 5 é maior que 3

Resultado:

| NÃO

Pronto. Só isso.

---

## Exemplo 2

Pergunta:

| 5 é menor que 3?

Resposta:

| NÃO

Agora aplica o **NAO**:

| NÃO é verdade que 5 é menor que 3

Resultado:

| SIM

---

## AGORA EM VISUALG (sem mistério)

```
5 > 3
```

👉 isso é **VERDADEIRO**

```
NAO (5 > 3)
```

👉 isso vira **FALSO**

```
5 < 3
```

👉 isso é **FALSO**

```
NAO (5 < 3)
```

👉 isso vira VERDADEIRO

## Teste com operadores lógicos

The image shows a computer interface with two main windows. On the left is a 'Pseudocode' editor with the following code:

```
1 Algoritmo "triangulos"
2
3 Var
4     L1,L2,L3: real
5     EQ,ES,TRI: logico
6
7
8 Inicio
9
10 escreva ("Digite o primeiro lado: ")
11 Leia (L1)
12 escreva ("Digite o segundo lado: ")
13 Leia (L2)
14 escreva ("Digite o terceiro lado: ")
15 Leia (L3)
16 TRI <- (L1 < L2 + L3) e (L2 < L1 + L3) e (L3 < L1 + L2)
17 EQ <- (L1 = L2) e (L2 = L3)
18 ES <- (L1 <> L2) e (L2 <> L3)
19 Escreval ("Pode formar um triangulo? ", TRI)
20 Escreval ("É um Equilatero: ", EQ)
21 Escreval ("É um Escaleno: ", ES)
22 Fimalgoritmo
```

On the right is a terminal window titled 'Console simulando o modo texto do MS-DOS' with the following output:

```
Digit o primeiro lado: 6
Digit o segundo lado: 7
Digit o terceiro lado: 8
Pode formar um triangulo? VERDADEIRO
É um Equilatero: FALSO
É um Escaleno: VERDADEIRO
>>> Fim da execucao do programa !
```



## AGORA COM UMA VARIÁVEL (bem humano)

Idade ← 16

Pergunta:

| Idade é maior ou igual a 18?

Resposta:

| NÃO

Agora com NAO:

| NÃO é verdade que idade ≥ 18

Resultado:

| SIM

Ou seja:

| Ele é menor de idade

## TRADUÇÃO PARA PORTUGUÊS (isso ajuda MUITO)

NAO (Idade >= 18)

Leia assim:

| “Não é verdade que a idade é maior ou igual a 18”

Se isso for verdadeiro, a pessoa é menor de idade.

## ÚNICA REGRA QUE VOCÊ PRECISA LEMBRAR

| Primeiro o computador responde a pergunta.

| Depois o NAO troca a resposta.

Nada além disso.

## SE AINDA CONFUNDIR, USA ESSA TÉCNICA

Leia sempre assim:

| “NÃO é verdade que ...”

Exemplo:

NAO (Nota >= 7)

Leia:

| Não é verdade que a nota é maior ou igual a 7

---

## Operador E (AND)

Só é verdadeiro se **todas** as condições forem verdadeiras.

Idade  $\geq 18$  E Idade  $\leq 65$

✓ Verdadeiro  $\rightarrow$  idade entre 18 e 65

✗ Falso  $\rightarrow$  qualquer valor fora disso

---

## Operador OU (OR)

É verdadeiro se **pelo menos uma** condição for verdadeira.

Nota  $\geq 7$  OU Frequencia  $\geq 75$

✓ Basta UMA condição verdadeira

---

## Operador NAO (NOT)

Inverte o resultado lógico.

NAO (Idade  $\geq 18$ )

- Se idade  $\geq 18 \rightarrow$  vira FALSO
  - Se idade  $< 18 \rightarrow$  vira VERDADEIRO
- 

## Ordem de precedência lógica

1 Parênteses ( )

2 NAO

3 E

4 OU

👉 Sempre use parênteses para evitar erro.

| ORDEM DE PRECEDÊNCIA |                        |
|----------------------|------------------------|
| Aritméticos          | ( )<br>^<br>* /<br>+ - |
| Relacionais          | Todos                  |
| Lógicos              | E<br>OU<br>NÃO         |

## 💡 Exemplos práticos

### Exemplo 1 – Maior de idade

```
Se Idade >= 18 Entao  
    Escreva("Maior de idade")
```

FimSe

## Exemplo 2 – Intervalo

```
Se Idade >= 18 E Idade <= 60 Entao  
    Escreva("Faixa permitida")  
FimSe
```

## Exemplo 3 – Aprovado

```
Se Nota >= 7 OU Frequencia >= 75 Entao  
    Escreva("Aprovado")  
Senao  
    Escreva("Reprovado")  
FimSe
```

## Exemplo 4 – Negação

```
Se NAO (Senha = 1234) Entao  
    Escreva("Senha incorreta")  
FimSe
```

## DICAS PRA GRAVAR

- Comparação → = <> > < >= <=
- Todas verdadeiras → **E**
- Uma verdadeira → **OU**
- Inverte → **NAO**
- Use parênteses

✓ Se você entende operadores lógicos, você entende **decisão**.

# ÷ Operadores Aritméticos

Considerando:

A ← 5

B ← 2

| Operador | Descrição   | Resultado             |
|----------|---|-----------------------|
| +        | Adição  | 7                     |
| -        | Subtração   | 3                     |
| *        | Multiplicação   | 10                    |
| /        | Divisão   | 2.5 (não usa virgula) |
| \        | Divisão inteira   | 2                     |
| ^        | Exponenciação   | 25                    |
| %        | Módulo (ele divide o número mas ele pega o resto que sobrou da divisão) | 1                     |

**OPERADORES ARITMÉTICOS**

|   |                 | A ← 5 | B ← 2 |
|---|-----------------|-------|-------|
| + | Adição          | A + B | 7     |
| - | Subtração       | A - B | 3     |
| * | Multiplicação   | A * B | 10    |
| / | Divisão         | A / B | 2.5   |
| \ | Divisão Inteira | A \ B | 2     |
| ^ | Exponenciação   | A ^ B | 25    |
| % | Módulo          | A % B | 1     |

5 | 2  
1 ) 2

Vídeo →

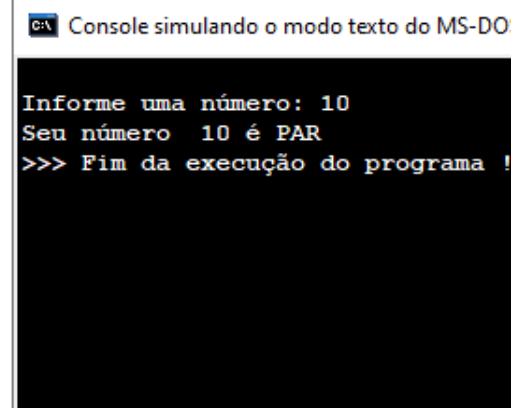
Algoritmos

**A Lembrando que os símbolos desses operadores podem mudar de linguagem para linguagem, se apegue aos FUNDAMENTOS**

<https://youtu.be/BQHcwIPxZ9E?si=-fg4OJR6TNnqtp8v>

## Exemplo Módulo

```
Algoritmo "IMPAROUPAR"  
Var  
    numero, tipo_numero:inteiro  
Inicio  
    Escreva("Informe uma número: ")  
    Leia (numero)  
    tipo_numero <- (numero % 2)  
    se (tipo_numero = 1)entao  
        escreva("Seu número ",numero," é IMPAR")  
    senao  
        escreva("Seu número ",numero," é PAR")  
    FimSe  
Fimalgoritmo
```



```
c:\ Console simulando o modo texto do MS-DO  
Informe uma número: 10  
Seu número 10 é PAR  
>>> Fim da execução do programa !
```

## ⚠️ Ordem de Precedência

# ORDEM DE PRECEDÊNCIA

|             |                        |
|-------------|------------------------|
| Aritméticos | ( )<br>^<br>* /<br>+ - |
| Relacionais | Todos                  |
| Lógicos     | E<br>OU<br>NÃO         |

O computador resolve as expressões na seguinte ordem:

- 1 Parênteses ( )
- 2 Exponenciação ^
- 3 Multiplicação / Divisão \* / \ %
- 4 Adição / Subtração + -

👉 Parênteses são essenciais para evitar resultados errados.

## Exemplo: média

## Área dos algoritmos ( Edição do código fonte ) -> Nome do arquivo: [seunome]

```
1 Algoritmo "MeuNome"
2
3 Var
4   N1, N2: Inteiro
5   M: real
6
7
8
9 Inicio
10
11 Escreva ("Informe um número ")
12   Leia (N1)
13 Escreva ("Informe outro número ")
14   Leia (N2)
15   M <- (N1 + N2)/2
16 Escreva ("A media entre ",N1," e ",N2, " é igual a ",M)
17
18 FimAlgoritmo
```

Console simulando o modo texto do MS-DOS

```
Informe um número 5
Informe outro número 2
A media entre 5 e 2 é igual a 3.5
>>> Fim da execução do programa !
```

### Algoritmo "Valores"

Var

N1, N2: Inteiro

M: Real

Inicio

Escreva ("Informe um número ")

Leia (N1)

Escreva ("Informe outro número ")

Leia (N2)

M ← (N1 + N2) / 2

Escreva ("A média entre ", N1, " e ", N2, " é igual a ", M)

FimAlgoritmo



**M** é do tipo **Real**, pois divisões podem gerar números quebrados.



Os parênteses garantem que a soma seja feita antes da divisão.



## Funções Aritméticas

| Função   | Descrição        | Exemplo       |
|----------|------------------|---------------|
| abs      | Valor absoluto   | abs(-10) = 10 |
| exp      | Exponenciação    | exp(3,2) = 9  |
| int      | Parte inteira    | int(5.9) = 5  |
| raizq    | Raiz quadrada    | raizq(81) = 9 |
| pi       | Retorna π        | 3.1415...     |
| sen      | Seno (rad)       | sen(x)        |
| cos      | Cosseno (rad)    | cos(x)        |
| tan      | Tangente (rad)   | tan(x)        |
| grauprad | Graus → radianos | grauprad(90)  |

## FUNÇÕES ARITMÉTICAS

|          |                |              |         |
|----------|----------------|--------------|---------|
| Abs      | Valor Absoluto | Abs(-10)     | 10      |
| Exp      | Exponenciação  | Exp(3,2)     | 9       |
| Int      | Valor Inteiro  | Int(3.9)     | 3       |
| RaizQ    | Raiz Quadrada  | RaizQ(25)    | 5       |
| Pi       | Retorna Pi     | Pi           | 3.14... |
| Sen      | Seno (rad)     | Sen(0.523)   | 0.5     |
| Cos      | Cosseno (rad)  | Cos(0.523)   | 0.86    |
| Tan      | Tangente (rad) | Tan(0.523)   | 0.57    |
| GraupRad | Graus para Rad | GraupRad(30) | 0.52    |


## Exemplos práticos

### Valor absoluto

```
A ← abs(-50)
```

## Raiz quadrada inteira

```
A ← int(raizq(90))
```

## Seno de um ângulo

Algoritmo "Conversor"

Var

Angulo, S: Real

Inicio

Escreva("Informe um ângulo: ")

Leia (Angulo)

S ← sen(grauprad(Angulo))

Escreva("O seno de ", Angulo, " é igual a ", S)

FimAlgoritmo

## Vídeo

<https://youtu.be/RDrfZ-7WE8c>



## Estruturas Condicionais

<https://youtu.be/ZAo9T78-32Q?si=wYoNiPcinNnZwkow>

[https://youtu.be/\\_g05aHdBAEY](https://youtu.be/_g05aHdBAEY)

<https://youtu.be/7gGHzqh4d8>

## 📌 O que são estruturas condicionais?

Estruturas condicionais servem para **tomar decisões** dentro de um algoritmo.

Elas permitem que o programa execute **um caminho ou outro**, dependendo de uma condição.

👉 Em resumo:

| SE algo for verdadeiro, faça isso. SENÃO, faça outra coisa.

## 🧠 Pensando como humano

Na vida real, você decide assim o tempo todo:

- Se estiver chovendo → levo guarda-chuva
- Senão → não levo

O computador funciona do mesmo jeito, só que usando **condições lógicas** (verdadeiro ou falso).

## ◆ Estrutura condicional simples – SE

A estrutura mais básica é o **SE**.

```
Se condicao Entao  
    comandos  
FimSe
```

📌 O que acontece:

- O computador **avalia a condição**
- Se for **VERDADEIRA**, executa os comandos
- Se for **FALSA**, pula tudo e segue o programa

### Exemplo:

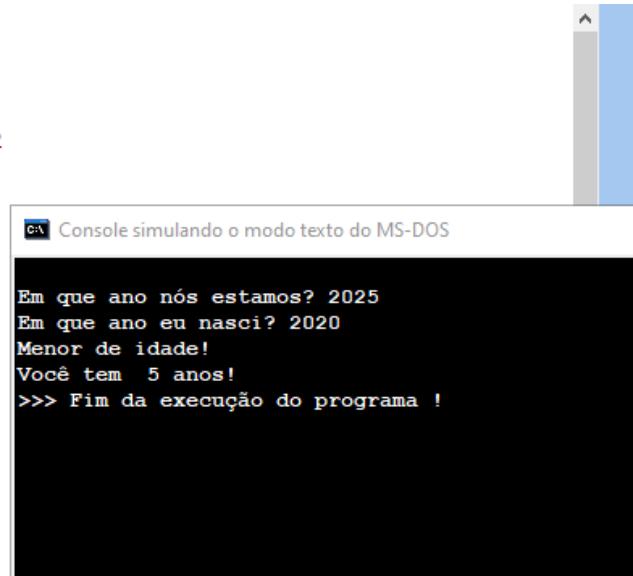
```
Se (Idade >= 18) Entao
    Escreva("Maior de idade")
FimSe
```

👉 Se a idade for 18 ou mais, a mensagem aparece.

👉 Se não for, nada acontece.

## ◆ Estrutura condicional composta – SE / SENAO

```
1 Algoritmo "semnome"
2
3 Var
4
5     ano_atual, ano_nasci, idade:inteiro
6
7 Inicio
8 Escreva("Em que ano nós estamos? ")
9     leia (ano_atual)
0 Escreva("Em que ano eu nasci? ")
1     Leia (ano_nasci)
2 idade <- (ano_atual - ano_nasci)
3 se (idade >= 18)então
4     escreval("Maior de idade!")
5 senao
6     escreval("Menor de idade!")
7 fimse
8 Escreva("Você tem ",idade," anos!")
9 Fimalgoritmo
```



```
Console simulando o modo texto do MS-DOS
Em que ano nós estamos? 2025
Em que ano eu nasci? 2020
Maior de idade!
Você tem 5 anos!
>>> Fim da execução do programa !
```



Usamos quando queremos tratar **duas possibilidades**.

```

Se condicao Entao
    comandos_se_verdadeiro
Senao
    comandos_se_falso
FimSe

```

📌 O programa **sempre executa um dos dois blocos**.

**Exemplo:**

```

Se Nota >= 7 Entao
    Escreva("Aprovado")
Senao

```

```
    Escreva("Reprovado")
FimSe
```

- 👉 Nota ≥ 7 → Aprovado
- 👉 Nota < 7 → Reprovado

## 🧠 A condição (parte mais importante)

A condição é uma **expressão lógica**, que só pode resultar em:

- **VERDADEIRO**
- **FALSO**

Ela pode usar:

- Operadores de comparação: `> < = <> >= <=`
- Operadores lógicos: `E` , `OU` , `NAO`

### Exemplo:

```
Se Idade >= 18 E Idade <= 60 Entao
    Escreva("Faixa permitida")
FimSe
```

## 🔄 Condições com NAO

O operador **NAO** inverte o resultado da condição.

```
Se NAO (Idade >= 18) Entao
    Escreva("Menor de idade")
FimSe
```

- 👉 Primeiro a condição é avaliada
- 👉 Depois o NAO inverte o resultado

## Importância dos parênteses

Parênteses deixam claro **o que deve ser avaliado primeiro**.

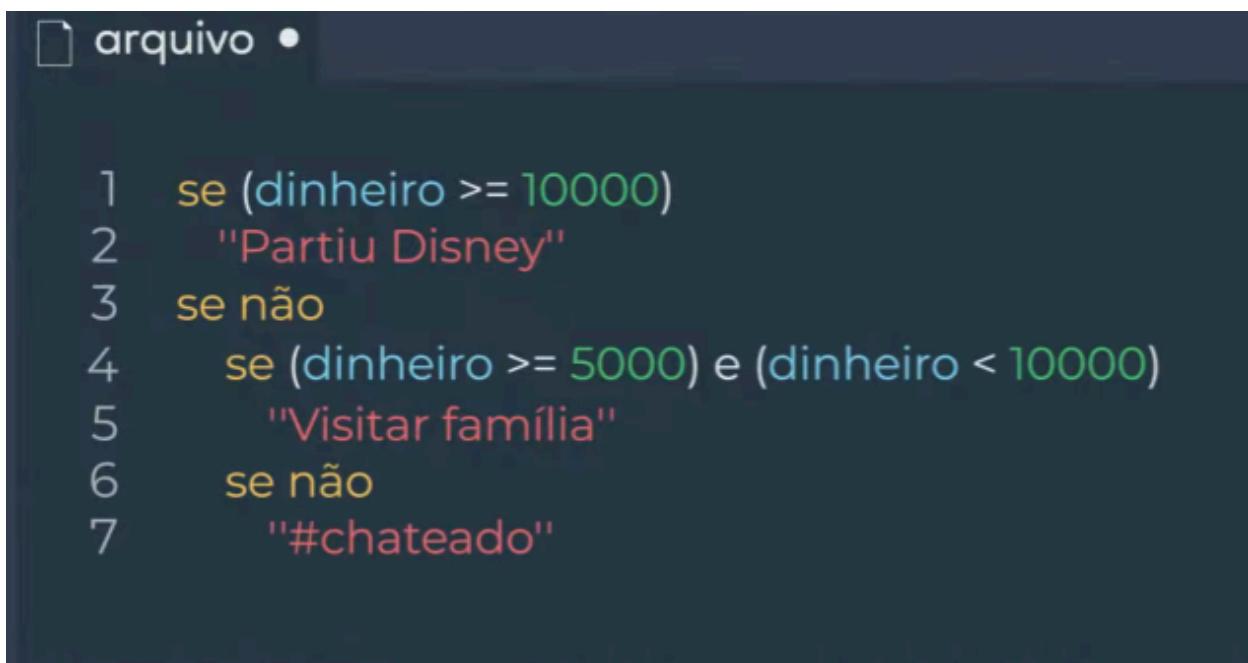
Eles evitam erros de lógica.

```
Se (Nota >= 7) OU (Frequencia >= 75) Entao  
    Escreva("Aprovado")  
FimSe
```

## Dica final (pra gravar)

- Estrutura condicional = **decisão**
- Toda condição vira **verdadeiro ou falso**
- **SE** executa apenas se for verdadeiro
- **SENAO** é o caminho alternativo
- Use parênteses

Exemplo condicional aninhada:



```
arquivo •  
  
1  se (dinheiro >= 10000)  
2      "Partiu Disney"  
3  se não  
4      se (dinheiro >= 5000) e (dinheiro < 10000)  
5          "Visitar família"  
6      se não  
7          "#chateado"
```

Se eu tiver 10000 eu vou a disney, se não, se eu tiver maior ou igual a 5000 e menor que 10000 eu vou visitar a família, se não tiver nenhum desses valores #chateado voce vai ficar em casa.

## Legibilidade de Código – Boas Práticas

### O que é legibilidade de código?

Legibilidade é o quanto **fácil** um código é de ler, entender e manter.

 Código legível não é feito só para o computador, é feito **para pessoas** (inclusive você no futuro).

Um código pode funcionar perfeitamente e ainda assim ser **ruim**, se for difícil de entender.

---

### Por que legibilidade é importante?

- Facilita o aprendizado
- Evita erros
- Ajuda na manutenção
- Permite que outras pessoas entendam o código
- Faz você entender seu próprio código depois de um tempo

 Na prática, **legibilidade vale tanto quanto o código funcionar.**

---



### Indentação (muito importante)

Indentação é o **espaçamento no início das linhas** para mostrar a estrutura do código.

### Código ruim (difícil de ler)

```
Se Idade >= 18 Entao  
    Escreva("Maior de idade")  
    Senao
```

```
Escreva("Menor de idade")
FimSe
```

## ✓ Código legível

```
Se Idade >= 18 Entao
    Escreva("Maior de idade")
Senao
    Escreva("Menor de idade")
FimSe
```

👉 A indentação mostra **o que está dentro de quê**.

## 🏷️ Nomes de variáveis claros

Use nomes que expliquem **o que a variável guarda**.

## ✗ Ruim

```
N1, N2, X
```

## ✓ Bom

```
Nota1, Nota2, Media
```

👉 Nome bom evita comentário desnecessário.

## ✍️ Comentários (quando usar)

Comentários servem para **explicar o porquê**, não o óbvio.

```
// Calcula a média do aluno
Media ← (Nota1 + Nota2) / 2
```

⚠️ Evite comentários inúteis:

```
// Soma N1 com N2
```

```
S ← N1 + N2
```

## Organização visual

Separe o código em blocos lógicos:

```
Escreva("Informe a nota 1")  
Leia(Nota1)
```

```
Escreva("Informe a nota 2")  
Leia(Nota2)
```

```
Media ← (Nota1 + Nota2) / 2
```

 Espaços ajudam o cérebro a entender melhor.

## Clareza nas condições

Use parênteses para deixar a lógica clara.

## Confuso

```
Se Nota >= 7 OU Frequencia >= 75 E Trabalhos = 1 Entao
```

## Claro

```
Se (Nota >= 7) OU (Frequencia >= 75 E Trabalhos = 1) Entao
```

## Evite complexidade desnecessária

Se a condição ficou grande demais, simplifique.

## Difícil

```
Se NAO (Nota < 7) E NAO (Frequencia < 75) Entao
```

## Melhor

```
Se Nota >= 7 E Frequencia >= 75 Entao
```

## Padronização

Mantenha sempre o mesmo padrão:

- Mesma indentação
- Mesmo estilo de nomes
- Mesmo formato de escrita

Consistência ajuda mais do que perfeição.

## Regra de ouro da legibilidade

| Código bom é código que outra pessoa entende sem você explicar.

✓ Quanto mais legível o código, menos dor de cabeça no futuro.

## Estruturas de Repetição

```
https://youtu.be/RlgdACEgA8c?si=dls\_eLEkSlwb4I04
```

### O que é repetição?

É quando você manda o computador **fazer a mesma coisa várias vezes**, sem copiar e colar código.

Um sistema de repetição precisa ter uma contagem, uma condição para a repetição parar e uma ação que é o que vai acontecer a cada repetição.

```
arquivo •  
1 pagina = 0  
2  
3 enquanto(pagina < 5)  
4 trocarDePagina()  
5 pagina = pagina + 1
```

Nessa estrutura genérica de código, a repetição faz com que a variável pagina aumente +1 até o resultado ser igual a 5.

## 1 Repetição tipo WHILE (enquanto)

👉 Enquanto a condição for verdadeira, o código roda.

### Exemplo (contador)

```
contador =1  
  
while contador <=5:  
    print(contador)  
    contador +=1
```

O que acontece passo a passo:

1. `contador = 1`
2. `contador <= 5` → verdadeiro
3. imprime `1`
4. soma `+1`
5. repete até ficar falso

 Se esquecer de aumentar o contador → **loop infinito**.

## 2 Repetição que executa pelo menos uma vez (DO WHILE)

Nem toda linguagem tem isso direto, mas a lógica é essa:

 Executa **primeiro**, testa **depois**.

### Exemplo (senha)

```
senha =""  
  
whileTrue:  
    senha =input("Digite a senha: ")  
    if senha =="1234":  
        break  
  
    print("Acesso permitido")
```

 Tradução humana:

| "Repita até a senha estar correta."

## 3 Repetição tipo FOR (quando você sabe quantas vezes)

 Melhor quando existe começo e fim claros.

### Exemplo

```
for iinrange(1,6):  
    print(i)
```



- Aqui:
- começa em 1
  - termina em 5
  - repete automaticamente
- 



## Quando usar cada um

| Situação                         | Use                          |
|----------------------------------|------------------------------|
| Não sabe quantas vezes           | <code>while</code>           |
| Precisa rodar pelo menos uma vez | repetição com teste no final |
| Sabe exatamente quantas vezes    | <code>for</code>             |

---

### ⚠ Exemplo RUIM (erro clássico)

```
contador =1  
  
while contador <=5:  
    print(contador)
```

✗ Nunca muda o contador → **programa trava**

---

### ✓ Exemplo BOM (legível e seguro)

```
contador =1  
  
while contador <=5:  
    print(contador)  
    contador +=1
```

## Regra de ouro (guarda isso):

Toda repetição precisa de:

-  1 início
-  2 condição
-  3 mudança

Se faltar um → bug.

## O que são funções?

<https://youtu.be/FCB9RIX2fqs?si=Rrffkf7NNGdp3Zsg>

Funções são **blocos de código com nome**, criados para:

- organizar o programa
- evitar repetir código
- deixar tudo mais claro e reutilizável

 Pense assim:

| Função é uma máquina: você dá algo → ela processa → devolve um resultado.

## Por que usar funções?

Sem função (ruim 🤢):

```
print(10 +5)  
print(10 +5)  
print(10 +5)
```

Com função (bom 👍):

```
def soma():  
    print(10 + 5)
```

```
soma()  
soma()  
soma()
```

👉 Escreve uma vez, usa várias.

## 📦 Estrutura básica de uma função

```
def nome_da_funcao(parametros):  
    # código  
    return resultado
```

Tem 3 partes importantes:

1. **Nome** (o que ela faz)
2. **Parâmetros** (o que ela recebe)
3. **Retorno** (o que ela devolve)

### 1 Função sem parâmetro

```
def saudacao():  
    print("Olá, seja bem-vindo!")  
  
saudacao()
```

🧠 Tradução:

“Execute esse código quando eu chamar saudacao.”

## 2 Função com parâmetro

```
defsaudacao(nome):  
    print("Olá,", nome)
```

```
saudacao("Marcus")  
saudacao("Ana")
```

👉 O valor muda, a função é a mesma.

## 3 Função com retorno (a mais importante)

```
defsoma(a, b):  
    return a + b
```

```
resultado = soma(10,5)  
print(resultado)
```

🧠 Aqui a função **devolve** um valor.

✗ Erro comum: confundir **print** com **return**

```
defsoma(a, b):  
    print(a + b)
```

✗ Isso só mostra na tela, **não devolve nada**.

✓ Correto:

```
defsoma(a, b):  
    return a + b
```

Exemplo:

```
def dobro(n):
    return n * 2

x = dobro(5)
print(x)
```

Aqui:

- `dobro` → função
- `n` → parâmetro
- `n * 2` → resultado
- `return` → devolve o valor

Nesse caso a função `dobro` recebe `n` que é o valor que vamos definir e ele retorna o valor de `n` multiplicado por 2, quando `x` chama a função `dobro` com o valor 5 é feito o cálculo da função e `print` exibe o resultado em `x`

É literalmente `return` retorna um valor e `print` somente exibe na tela.

## 4 Função usada dentro de outra

```
defdobro(n):
    return n *2

deftriplo(n):
    return n *3

print(dobro(5))
print(triplo(5))
```

👉 Funções se combinam como peças de LEGO.

## Boas práticas (importante)

- ✓ Nome claro
- ✓ Função faz **uma coisa só**
- ✓ Poucas linhas
- ✓ Retorno previsível

Exemplo bom:

```
defcalcular_media(nota1, nota2):  
    return (nota1 + nota2) /2
```

## Regra de ouro das funções

| Se você repetiu código, deveria ter uma função.

## Exemplo real (uso de verdade)

```
defpode_dirigir(idade):  
    return idade >= 18  
  
if pode_dirigir(20):  
    print("Pode dirigir")  
else:  
    print("Não pode dirigir")
```

## Criança Esperança (VISÃO GERAL)

Esse algoritmo:

1. Mostra um **menu de opções de doação**
2. Lê a **opção escolhida pelo usuário**
3. Usa **Escolha** (switch/case) para decidir o valor da doação

#### 4. Mostra o valor final doado

---

## ◆ 1. NOME DO ALGORITMO

Algoritmo "CriancaEsperanca"

👉 Dá um nome ao programa.

Serve só para **organização**, não interfere na lógica.

---

## ◆ 2. DECLARAÇÃO DAS VARIÁVEIS

Var  
D : Inteiro  
Valor : Real

🔍 O que cada uma faz:

- **D** → guarda a **opção do menu** (1 a 5 lembrando que nas opções do menu só pode numero inteiro)
- **Valor** → guarda o **valor da doação** (por isso é **Real**, pode ter centavos)

👉 Regra importante:

Antes de usar qualquer variável, você **precisa declarar o tipo**.

---

## ◆ 3. INÍCIO DO PROGRAMA

Início

👉 Aqui começa a execução do algoritmo.

## ◆ 4. APRESENTAÇÃO NA TELA

```
EscrevaL("-----")
EscrevaL("      CRIANÇA ESPERANÇA      ")
EscrevaL("-----")
```

🧠 O que é **EscrevaL** ?

- **EscrevaL** = **escreve e pula linha**
- Serve só para **organizar visualmente** o texto

👉 Isso **não afeta lógica nenhuma**, é só interface.

## ◆ 5. MENU DE OPÇÕES

```
EscrevaL("Muito obrigado por ajudar")
EscrevaL("[1] para doar R$10 ")
EscrevaL("[2] para doar R$20 ")
EscrevaL("[3] para doar R$30 ")
EscrevaL("[4] para doar outros valores ")
EscrevaL("[5] para cancelar ")
```

👉 Aqui o programa **mostra as opções** para o usuário escolher.

## ◆ 6. LEITURA DA ESCOLHA

```
Leia (D)
```

🧠 O computador pensa assim:

| "Vou esperar o usuário digitar um número e guardar em D"

Exemplo:

- Usuário digita `2`
- Agora `D = 2`

## ◆ 7. ESTRUTURA **ESCOLHA** (O CORAÇÃO DO PROGRAMA)

Escolha D

👉 Aqui o programa diz:

| "Dependendo do valor de D, vou executar um caminho diferente"

Isso é parecido com vários `se`, mas **mais organizado**.

### ◆ CASO 1

Caso 1

Valor  $\leftarrow 10$

🧠 Se `D = 1`

→ o valor da doação vira **10 reais**

### ◆ CASO 2

Caso 2

Valor  $\leftarrow 25$

 Se D = 2

→ o valor da doação vira **25 reais**

 Aqui tem um detalhe importante:

| No menu diz R\$20, mas no código está 25

| Isso é um **erro de lógica**, não de sintaxe.

## ◆ CASO 3

Caso 3

Valor ← 50

 Se D = 3

→ o valor vira **50 reais**

 Mesmo problema: no menu fala R\$30, mas no código usa 50.

## ◆ CASO 4 (INTERATIVO)

Caso 4

Escreva ("Qual o valor da doação? ")

Leia (Valor)

 Aqui o programa:

1. Pergunta quanto a pessoa quer doar
2. Lê o valor digitado
3. Guarda em Valor

 Esse caso é diferente porque **depende da entrada do usuário**.

## ◆ CASO 5 (CANCELAR)

Caso 5

Valor ← 0

🧠 Se o usuário cancelar:

- A doação vira **zero**
- Mas o programa **continua rodando normalmente**

## ◆ 8. FIM DO ESCOLHA

FimEscolha

👉 Encerra a tomada de decisão.

## ◆ 9. EXIBIÇÃO DO RESULTADO

```
EscrevaL("-----")
EscrevaL(" SUA DOAÇÃO FOI DE R$",Valor)
EscrevaL(" MUITO OBRIGADO! ")
EscrevaL("-----")
```

🧠 O programa mostra:

- O valor final da doação
- Mesmo se for **0**, ele mostra

## ◆ 10. FIM DO ALGORITMO

Fimalgoritmo

👉 Aqui o programa termina.

## Estrutura de Repetição

para entender isso voce tem que entender que para ser uma estrutura de repetição precisa de tres coisas

- Contagem(quantidade de vezes que vai se repetir)
- Condição(a condição que a contagem pare)
- Ação (trocar de página)

Exemplo com código generico

```
pagina ← 0
enquanto (pagina <= 5) faça
    passarPagina
    pagina ← pagina + 1
FimEnquanto
```

Nesse exemplo pagina começa igual a 0, quando usamoso enquanto ele até ser maior ou igual a 5 vai somar +1 na variavel pagina, até que chegue em 5 e a repetição acabe., lembrando que ele vai fazendo linha a linha e quando chega no fimenquanto ele volta para o enquanto até o valor ser ≤5.

Exemplo no Visual

```
Algoritmo "conteate10"
```

```
Var
```

```
    contador : inteiro
```

Inicio

```
contador ← 0
enquanto (contador <= 100) faça
    EscrevaL(contador)
        contador ← contador + 10
fim enquanto
Escreval("Contagem terminada")
```

Fimalgoritmo

Nesse código ele conta de 0 até 100 somando em 10 em 10.

Algoritmo "contador"

Var

```
contador, numero1, numero2 : inteiro
```

Inicio

```
Escreval("Digite um número: ")
Leia (numero1)
Escreval("Digite outro número: ")
Leia (numero2)
    contador ← numero1
    enquanto (contador <= numero2) faça
        EscrevaL(contador)
            contador ← contador + 10
    fim enquanto
    Escreval("Contagem terminada")
```

Fimalgoritmo

Nesse aqui o usuário define os números

Algoritmo "conte"

Var

  contador,numero1,quant : inteiro

Inicio

  Escreval("Quer contar ate quanto? ")

  Leia (numero1)

  Escreval("Quer contar em quanto em quanto? ")

  Leia(quant)

    contador ← 0

    enquanto (contador <= numero1) faca

      Escreval(contador)

      contador ← contador + quant

    fimenquanto

  Escreval("Contagem terminada")

Fimalgoritmo

Outro exemplo

algoritmo "somadornumerico"

var

  cont,N,soma : inteiro

inicio

  cont ← 1

  soma ← 0

  enquanto (cont <= 5)faca

    escreva("Digite o ",cont,"o. valor: ")

    leia(N)

    soma ← soma + N

    cont ← cont + 1

  fimenquanto

```
Escreva("A soma de todos os valores é: ",soma)
fimalgoritmo
```

## O que é a função (ou estrutura) enquanto?

O **enquanto** é usado quando você quer **repetir algo várias vezes, enquanto** uma condição for verdadeira.

👉 Em outras palavras:

"Enquanto isso for verdade, faça isso."

## Exemplo bem simples (pensando no dia a dia)

| Enquanto eu não terminar o dever, continuo estudando.

- Condição: *não terminei o dever*
- Ação: *estudar*
- Quando terminar o dever, a condição deixa de ser verdadeira → o laço para.

## Exemplo em programação (bem básico)

```
contador =1

while contador <=5:
    print(contador)
    contador = contador +1
```

## O que está acontecendo aqui?

1. `contador` começa com valor **1**
2. O programa pergunta:
  - `contador` é menor ou igual a **5**?
3. Se **sim** → executa o código dentro do `while`

4. Soma 1 ao contador
5. Volta ao passo 2
6. Quando `contador` vira **6**, a condição é falsa → o programa para o `while`

📌 Resultado:

```
1  
2  
3  
4  
5
```

## Regra de ouro do enquanto

⚠ Sempre tem que mudar algo dentro do `while`, senão ele entra em **loop infinito** (nunca para).

Exemplo errado:

```
while1 <5:  
    print("Nunca para")
```

👉 Isso roda para sempre, porque `1 < 5` **sempre será verdade**.

## Resumo bem curtinho

- **enquanto** repete um bloco de código
- Ele roda **enquanto a condição for verdadeira**
- Quando a condição vira falsa, ele para
- Sempre precisa de algo que faça a condição mudar

## Diferença entre ENQUANTO e REPITAj

A diferença principal é quando a condição é testada.

## 1 ENQUANTO (while)

👉 Testa a condição no início

| Enquanto a condição for verdadeira, executa.

**Funcionamento:**

1. Testa a condição
2. Se for **verdadeira** → executa
3. Se for **falsa** → nem entra

**Exemplo:**

```
contador ← 1

enquanto contador <= 3 faca
    escreva(contador)
    contador ← contador + 1
fimenquanto
```

**Resultado:**

```
1
2
3
```

**Importante:**

Se a condição já começar **falsa**, o código **não executa nenhuma vez**.

```
contador ← 5

enquanto contador < 3 faca
```

```
escreva("Não entra")
fim enquanto
```

👉 Não aparece nada.

## 2 REPITA (do...while)

👉 Testa a condição no final

| Executa primeiro, testa depois.

Ele é basicamente a inversão lógica do ENQUANTO

### Funcionamento:

1. Executa o bloco
2. Testa a condição
3. Se for **falsa** → repete
4. Se for **verdadeira** → para

### Exemplo:

```
contador ← 5

repita
    escreva(contador)
    contador ← contador + 1
    ate contador > 3
```

### Resultado:

```
5
```

👉 Mesmo a condição sendo “estranha”, **executa pelo menos uma vez**.

## Diferença-chave (guarda isso)

| Estrutura | Testa quando? | Executa pelo menos 1 vez? |
|-----------|---------------|---------------------------|
| enquanto  | No início     | ✗ Não                     |
| repita    | No final      | ✓ Sim                     |

```
C<-1
Enquanto (C<=10) faça
    EscrevaL(C)
    C<-C+1
FimEnquanto

C<-1
repita
    EscrevaL(C)
    C<-C+1
    Até (C>10)
```

## Quando usar cada um?

### ● Use ENQUANTO quando:

- Você só quer executar se a condição já for verdadeira
- Exemplo:
  - | Enquanto houver saldo, continue comprando

### ● Use REPITA quando:

- Você precisa executar pelo menos uma vez

- Muito comum com **entrada de dados**

Exemplo clássico:

```
repita
    escreva("Digite um número maior que zero: ")
    leia(numero)
    ate numero > 0
```

👉 Mesmo que o usuário digite errado, o programa **sempre pergunta pelo menos uma vez.**

## Respondendo sua última pergunta: “qual é melhor?”

💡 **Nenhum é melhor**

👉 Eles servem para **situações diferentes**

- **enquanto** = testa antes
- **repita** = executa antes

Frase pra nunca esquecer:

| ENQUANTO pergunta antes de entrar

| REPITA entra antes de perguntar

## Exemplo Contador de Negativos

```
Algoritmo "ContaNegativos"
Var
    N, C, TotN: Inteiro
Inicio
    C <-1
    TotN ← 0
    Repita
        Escreva ("Digite um numero: ")
        Leia (N)
```

```

Se (N<0) entao
    TotN ← TotN +1
    FimSe
    C ← C + 1
    Ate (C > 5)
    Escreval ("Foram digitados ",TotN," valores negativos")
Fimalgoritmo

```

## Exemplo Fatorial:

```

Algoritmo "Fatorial"
Var
    C, N, F: Inteiro
    R: Caractere
Inicio
Repita
    Escreva("Digite um numero: ")
    Leia (N)
    C ← N
    F ← 1
    repita
        F ← F * C
        C ← C - 1
    ate(C < 1)
    EscrevaL ("O valor do fatural de ",N," é igual a ",F)
    Escreva ("Quer continuar? [S/N]")
    Leia(R)
    LimpaTela
Ate(R = "N")
Fimalgoritmo

```

## Exemplo Números primos:

```

Algoritmo "NumeroPrimo"
Var

```

```

C,N,ContDiv:Inteiro
Inicio
    C ← 1
    ContDiv ← 0
    Escreva("Digite um numero: ")
    Leia(N)
    Repita
        Se (N % C = 0)entao
            ContDiv ← ContDiv + 1
        FimSe
        C ← C + 1
        Ate (C > N)
        Se (ContDiv > 2) entao
            EscrevaL("O valor ", N," não é primo!")
        senao
            EscrevaL("O valor ", N," é primo!")
        fimse
    Fimalgoritmo

```

## Vídeo

[https://youtu.be/fP49L1i\\_-HU](https://youtu.be/fP49L1i_-HU)

Nossa última estrutura vai ser o PARA

```

Para variavel ← inicio ate fim [passo salto] faca
    BLOCO
FimPara

```

Algoritmo "conta1a10PARA"

Var

C : inteiro

Inicio

```
para C ← 1 ate 10 passo 1 faca  
    Escreval(C)  
fimpara
```

Fimalgoritmo

Nada me impede de usar estruturas condicionais junto:

Algoritmo "valoresPares"

Var

cont, V : inteiro

Inicio

Escreva("Digite um valor: ")

leia (V)

se(V % 2=1)entao

V ← V - 1

fimse

Para cont ← V ate 0 passo -2 faca

escreval(cont)

fimpara

Fimalgoritmo

## 📌 Código

Algoritmo "Combinacoes"

Var

C1, C2 : Inteiro

Inicio

Para C1 ← 1 ate 3 faca

Para C2 ← 1 ate 3 faca

```
Escreval(C1, C2)
fimpara
fimpara
Fimalgoritmo
```

## Ideia principal

Esse algoritmo gera **todas as combinações possíveis** de dois números de **1 a 3**.

👉 Por isso o nome **Combinações**.

## O que é isso?

Aqui temos um **PARA dentro de outro PARA**

Isso se chama **laço de repetição aninhado**.

- O **C1** é o **laço externo**
- O **C2** é o **laço interno**

## Como o código funciona na prática

### 1 Primeira volta do laço externo

- **C1 = 1**

Agora entra no laço interno:

| C1 | C2 | Saída |
|----|----|-------|
| 1  | 1  | 11    |
| 1  | 2  | 12    |
| 1  | 3  | 13    |

Quando **C2** chega em 3, o laço interno termina.

### 2 Segunda volta do laço externo

- C1 = 2

Laço interno roda **de novo do zero**:

| C1 | C2 | Saída |
|----|----|-------|
| 2  | 1  | 21    |
| 2  | 2  | 22    |
| 2  | 3  | 23    |

---

### 3 Terceira volta do laço externo

- C1 = 3

Laço interno mais uma vez:

| C1 | C2 | Saída |
|----|----|-------|
| 3  | 1  | 31    |
| 3  | 2  | 32    |
| 3  | 3  | 33    |

---

### 💻 Saída final na tela

```
11
12
13
21
22
23
31
32
33
```



### Resumo mental (bem importante)

- O **PARA de fora** muda devagar

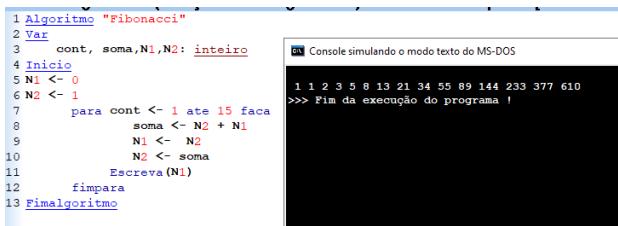
- O **PARA** de dentro roda inteiro **toda vez**
- Para cada valor de **C1**, o **C2** vai de 1 até 3

👉 Pense assim:

| "Para cada valor de C1, combine com todos os valores de C2"

## 🌀 Quando isso é usado?

- Combinações
- Tabelas (linhas × colunas)
- Matrizes
- Tabuada
- Jogos
- Menus em grade



```

1 Algoritmo "Fibonacci"
2 Var
3   cont, soma,N1,N2: inteiro
4 Inicio
5 N1 <- 0
6 N2 <- 1
7   para cont <- 1 ate 15 faca
8     soma <- N2 + N1
9     N1 <- N2
10    N2 <- soma
11    Escreva(N1)
12  fimpara
13 Fimalgoritmo

```

Console simulando o modo texto do MS-DOS

```

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610
>>> Fim da execução do programa !

```

### Algoritmo "Fibonacci"

```

Var
  cont, soma,N1,N2: inteiro
Inicio
  N1 <- 0
  N2 <- 1
  para cont <- 1 ate 15 faca
    soma <- N2 + N1
    N1 <- N2
    N2 <- soma
    Escreva(N1)

```

fimpara  
Fimalgoritmo

A variável que fica no para é somente um contador de quantas vezes vai repetir.

| escopo | nome | tipo | val |
|--------|------|------|-----|
| GLOBAL | CONT | I    | 15  |
| GLOBAL | SOMA | I    | 987 |
| GLOBAL | N1   | I    | 610 |
| GLOBAL | N2   | I    | 987 |

## ◆ 1 Sobre o PARA

O que você disse está certíssimo:

- A variável do PARA é só um contador
- Ela recebe 1
- Vai até 15
- Isso significa:
  - 👉 "Execute esse bloco 15 vezes"

Ela não participa da lógica, ela só controla quantas repetições acontecem.

## ◆ 2 O que acontece dentro do PARA

Agora vem o coração do Fibonacci, e você explicou muito bem:

**Situação inicial:**

N1 = 0  
N2 = 1

Dentro do PARA, em cada repetição:

1. Você soma os dois valores:

soma = N1 + N2

1. Você faz o deslocamento:

N1 recebe N2

N2 recebe soma

Isso é exatamente o que mantém a sequência andando.

👉 Esse passo é o **segredo** do Fibonacci.

## ◆ 3 Por que essa ordem é importante?

Você fez certo ao perceber isso:

| "antes de N2 receber a soma, eu fiz N1 receber N2"

Se você invertesse:

- perderia o valor antigo
- quebraria a sequência

Então essa ordem não é aleatória — é **lógica pura**.

## ◆ 4 Sobre o **Escreva(N1)**

Sim, seu pensamento está correto 👍

Quando você escreve **N1 depois da atualização**, ele:

- sempre contém o valor correto da sequência naquele momento
- mostra a progressão certinha do Fibonacci

Você poderia escrever **N2** também, dependendo de **qual termo você quer mostrar** — mas isso já é detalhe fino.

## Comparação direta das estruturas

### ◆ ENQUANTO

```
enquanto (condicao) faca  
    bloco  
fimenquanto
```

- Testa a condição **ANTES**
- Pode executar **0 vezes**
- Depende de algo **mudar dentro** do laço
- Usado quando **não se sabe quantas vezes** vai repetir

 "Enquanto isso for verdadeiro, continua"

---

### ◆ REPITA

```
repita  
    bloco  
ate (condicao)
```

- Testa a condição **DEPOIS**
- Executa **pelo menos 1 vez**
- Também depende de condição lógica
- Usado quando você **precisa executar ao menos uma vez**

 "Repete até isso ser verdadeiro"

---

### ◆ PARA

```
para i ← inicio ate fim faca  
    bloco
```

fimpara

- **NÃO depende de condição lógica**
- Depende de **quantidade**
- Executa um número **fixo** de vezes
- O contador é controlado automaticamente

👉 "Faça isso X vezes"

## 🧠 A frase que fixa tudo

| ENQUANTO e REPITA dependem de condição

| **PARA depende de quantidade**

🔧 Dá pra mudar a quantidade no **PARA** ?

Sim. Exatamente como você falou.

Exemplo:

```
para cont ← 1 ate 15 faca
```

Significa:

| "Repita 15 vezes"

Se você mudar para:

```
para cont ← 1 ate 30 faca
```

👉 Agora repete **30 vezes**.

Nada mais muda.

# ! O **PARA** pode virar um **ENQUANTO** ?

Conceitualmente, **sim**, mas **não é pra fazer isso** em lógica básica.

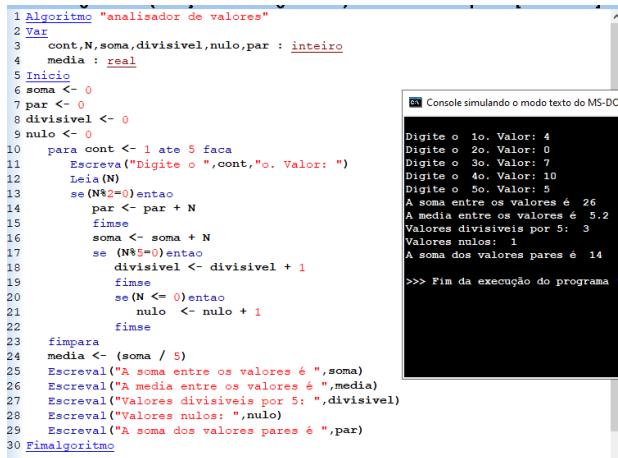
Por quê?

- o **PARA** é feito pra **controle fixo**
- o **ENQUANTO** é feito pra **condição variável**

Misturar os dois conceitos deixa o código confuso.

## Resumo final (pra guardar)

- **ENQUANTO** → repete **enquanto condição for verdadeira**
- **REPITA** → repete **até condição ficar verdadeira**
- **PARA** → repete **X vezes**
- **PARA** **não depende de condição lógica**
- você controla o **PARA** mudando a **quantidade**



```
1 Algoritmo "analizador de valores"
2 Var
3   cont,N,soma,divisivel,nulo,par : inteiro
4   media : real
5   Inicio
6     soma <- 0
7     par <- 0
8     divisivel <- 0
9     nulo <- 0
10    para cont <- 1 ate 5 faca
11      Escreva ("Digite o ",cont,"o. Valor: ")
12      Leia(N)
13      se (N%2=0) entao
14        par <- par + N
15      fimse
16      soma <- soma + N
17      se (N%5=0) entao
18        divisivel <- divisivel + 1
19      fimse
20      se (N <= 0) entao
21        nulo <- nulo + 1
22      fimse
23    fimpara
24    media <- (soma / 5)
25    Escreval("A soma entre os valores é ",soma)
26    Escreval("A media entre os valores é ",media)
27    Escreval("Valores divisiveis por 5: ",divisivel)
28    Escreval("Valores nulos: ",nulo)
29    Escreval("A soma dos valores pares é ",par)
30  Fimalgoritmo
```

Algoritmo "analizador de valores"

Var

cont,N,soma,divisivel,nulo,par : inteiro  
 media : real

Inicio

soma <- 0

```

par ← 0
divisivel ← 0
nulo ← 0
para cont ← 1 ate 5 faca
    Escreva("Digite o ",cont,"o. Valor: ")
    Leia(N)
    se (N%2=0)entao
        par ← par + N
    fimse
    soma ← soma + N
    se (N%5=0)entao
        divisivel ← divisivel + 1
    fimse
    se (N <= 0)entao
        nulo ← nulo + 1
    fimse
fimpara
media ← (soma / 5)
Escreval("A soma entre os valores é ",soma)
Escreval("A media entre os valores é ",media)
Escreval("Valores divisiveis por 5: ",divisivel)
Escreval("Valores nulos: ",nulo)
Escreval("A soma dos valores pares é ",par)
Fimalgoritmo

```

**LEMBRAR DE TODA VEZ QUE ABRIR UM SE JÁ FECHAR COM O FIMSE PRA NÃO ERRAR**

Lembrando tambem que nesse caso media ficou fora do "para" porque não faz sentido ficar calculando media varias vezes, no final ele só faz o calculo uma vez

**Exercicio 12:**

**Como calcular desconto/porcetagem:**

```

Algoritmo "exercicio12"
Var

```

```

preco, promo : real
Inicio
    escreva("Preço do produto: ")
    leia(preco)
    //Preço Normal vezes a % (nesse caso 5%), dividido por 100.
    promo ← preco - (preco * 5 / 100)

    escreva("Preço na promoção: ", promo)
Fimalgoritmo

```

### Exercício 13:

#### Como dar aumento em %:

mesma coisa que o anterior só que inves de subtrair ele vai fazer adição.

```

Algoritmo "exercicio13"
Var
    salario, aumento : real
Inicio
    escreva("Escreva o salário: ")
    leia(salario)
    aumento ← salario + (salario * 15 / 100)
    escreva("Novo salário com 15% de aumento: R$", aumento)
Fimalgoritmo

```

## Procedimentos (Funções)

### O que é um procedimento?

Um **procedimento** é um bloco de código identificado por um nome, criado para **executar uma tarefa específica** dentro do algoritmo.

Ele serve para:

- Organizar melhor o código

- Evitar repetição de comandos
- Facilitar leitura e manutenção
- Deixar o algoritmo mais limpo e profissional

O procedimento **não executa sozinho**.

Ele só é executado quando é **chamado** no algoritmo principal.

---

## 📌 Estrutura básica de um procedimento

```
Procedimento NomeDoProcedimento()
  Inicio
    // comandos
  FimProcedimento
```

- **Procedimento** → palavra-chave
  - **NomeDoProcedimento** → nome que identifica o bloco
  - Dentro ficam os comandos que serão executados
- 

## 📌 Exemplo prático: Procedimento **Topo()**

```
Procedimento Topo()
  inicio
    LimpaTela
    EscrevaL("-----")
    EscrevaL(" D E T E C T O R D E P E S A D O ")
    EscrevaL(" Maior Peso até agora:", Mai, "Kg" )
    EscrevaL("-----")
  FimProcedimento
```

O que esse procedimento faz?

Sempre que `Topo()` é chamado, ele:

1. Limpa a tela
2. Mostra o título do programa
3. Exibe o maior peso registrado até o momento
4. Organiza a saída visual

Esse procedimento é usado apenas para **exibição**, não para cálculos.

---

## 📌 Como um procedimento é executado?

A execução acontece quando o nome do procedimento é chamado:

```
Topo()
```

Isso significa:

“Execute agora todos os comandos definidos dentro do procedimento Topo e depois volte para o ponto onde o programa parou.”

---

## 📌 Onde o procedimento é usado no algoritmo?

No algoritmo **DetectorPesado**, o procedimento `Topo()` é chamado:

- No início do programa → para mostrar a tela inicial
- Dentro do laço `Para` → para atualizar a tela a cada repetição
- No final do programa → para exibir o resultado final

Isso evita repetir várias vezes o mesmo bloco de código.

---

## 📌 Por que usar procedimentos?

Sem procedimento, o mesmo código teria que ser escrito várias vezes, o que:

- Polui o algoritmo

- Aumenta chance de erro
- Dificulta alterações futuras

Com procedimento:

- O código fica mais curto
- Qualquer alteração é feita em um único lugar
- O algoritmo fica mais legível

## 📌 Procedimento x Função (diferença importante)

### ◆ Procedimento

- Executa ações
- **Não retorna valor**
- Exemplo: `Topo()`

### ◆ Função

- Executa ações
- **Retorna um valor**
- Exemplo:

```
Funcao Dobro(x: Inteiro): Inteiro
```

```
    Retorne x * 2
```

```
FimFuncao
```

No algoritmo analisado, `Topo()` é um **procedimento**, não uma função.

## 📌 Conceito importante: escopo de variáveis

O procedimento `Topo()` usa a variável `Mai`, mesmo ela não estando declarada dentro dele.

Isso acontece porque:

- `Mai` é uma variável **global**
- Procedimentos podem acessar variáveis globais

## 🎯 Conclusão

Procedimentos são usados para:

- Organizar o código
- Evitar repetição
- Melhorar a legibilidade
- Facilitar manutenção

O procedimento `Topo()` é um exemplo correto e bem aplicado, pois centraliza toda a parte visual do programa em um único lugar.

## Par ou Impar Procedimento:

```
algoritmo "ParOuImparProcedimento"
var
    N: Inteiro
Procedimento ParOuImpar(V: Inteiro)
    Inicio
        Se(V%2 = 0)entao
            EscrevaL("O número ", V, " é PAR")
        senao
            EscrevaL("O número ", V, " é IMPAR")
        FimSe
    FimProcedimento
    inicio
        Escreva("Digite um número: ")
        Leia (N)
        ParOuImpar(N)
    finalgoritmo
```

# Parâmetros em Procedimentos e Funções

(Passagem por Valor x Passagem por Referência)

## Quando você define um procedimento ou função

Exemplo:

```
Procedimento Soma(A, B: Inteiro)
```

Isso significa:

- `A` e `B` são **parâmetros**
- Eles recebem valores **quando a função é chamada**
- Eles **existem só dentro** do procedimento/função

## Passagem por VALOR (sem `var`)

```
Procedimento Soma(A, B: Inteiro)
```

O que acontece aqui?

- O valor passado é **copiado**
  - `A` recebe uma **cópia do valor**
  - Alterar `A` **não altera** a variável original
- 👉 O valor fora da função **continua o mesmo**

## Exemplo prático

```
Procedimento Teste(A: Inteiro)
```

```
  Inicio
```

```
    A ← 10
```

```
  FimProcedimento
```

```
  Inicio
```

```
    X ← 5
```

```
    Teste(X)
```

```
    Escreva(X)
```

```
  FimAlgoritmo
```

## Resultado:

```
5
```

## Por quê?

- `X` vale 5
- `A` recebe uma **cópia do 5**
- `A` vira 10, mas **X não muda**
- Eles **não estão conectados**

👉 Isso é passagem por valor

---

## 🧠 Analogia simples

É como tirar **xerox**:

- Você pode rabiscar a cópia
  - O original continua intacto
-

## ◆ Passagem por REFERÊNCIA ( var )

Procedimento Teste(var A: Inteiro)

Agora muda tudo.

### O que acontece?

- A não recebe uma cópia
- A aponta para a mesma variável
- Alterar A altera o valor original

👉 Eles ficam conectados

### 📦 Exemplo prático

Procedimento Teste(var A: Inteiro)

Inicio

    A ← 10

FimProcedimento

Inicio

    X ← 5

    Teste(X)

    Escreva(X)

FimAlgoritmo

### Resultado:

10

## Por quê?

- `A` está ligado diretamente a `X`
  - Quando `A` muda, `X` muda também
  - Não existe cópia, existe **referência**
- 

## 🧠 Analogia simples

É como mexer no **original**:

- Qualquer alteração afeta o valor real
- 

## VS Comparação direta (pra fixar)

| Sem <code>var</code> | Com <code>var</code>    |
|----------------------|-------------------------|
| Passagem por valor   | Passagem por referência |
| Cópia do valor       | Conexão direta          |
| Não altera fora      | Altera fora             |
| Mais seguro          | Mais poderoso           |
| Usado para cálculo   | Usado para modificar    |

---

## 📌 Então o que você falou, traduzido corretamente:

| "Se tiver sem o var, ele não altera o valor final"

✓ CORRETO

| "Ele só pega um número"

✓ CORRETO — ele recebe uma cópia

| "Com var ele se conecta um no outro"

## ✓ PERFEITO

| "O valor de A se torna o outro valor A"

## ✓ Isso se chama **referência à mesma variável**

---

## ⚠ Quando usar `var` ?

Use `var` quando:

- Você quer alterar a variável original
- O procedimento serve para modificar algo

Exemplos:

- Atualizar saldo
  - Trocar valores
  - Incrementar contador
  - Alterar estado de algo
- 

## ✗ Quando NÃO usar `var` ?

Não use `var` quando:

- Só quer calcular
  - Só quer verificar
  - Só quer retornar algo
  - Não quer efeitos colaterais
- 

## 🔗 Ligação com linguagens reais (importante)

- **Python** → tudo é referência, mas com regras próprias

- **JavaScript** → primitivos por valor, objetos por referência
- **C / Pascal** → explícito com `var` ou ponteiro

👉 Esse conceito que você entendeu agora é **nível base de programação real**.

---

## 🎯 Conclusão final

- Sem `var` → **cópia**
- Com `var` → **ligação**
- Função/procedimento define **o que faz**
- Parâmetros definem **como os dados entram**
- `var` decide **se altera o mundo externo ou não**

## 🧠 Regra de ouro (grava isso)

- ◆ Variável global → lida no programa principal
- ◆ **Variável local** → usada só dentro do procedimento
- ◆ Procedimento = ações, não retorno

## 🧠 LÓGICA DO ANO BISSEXTO (importante!)

Um ano é **bissexto** se:

1 É divisível por 4

E

2 NÃO é divisível por 100

👉 OU

3 É divisível por 400

Em forma de lógica:

```
(ano %4 =0 E ano %100 <>0) OU (ano %400 =0)
```

## Exemplos rápidos

| Ano  | Resultado   | Motivo                             |
|------|---|------------------------------------|
| 2024 | Bissexto  | Divisível por 4                    |
| 1900 |  Não | Divisível por 100, mas não por 400 |
| 2000 |  Sim | Divisível por 400                  |
| 2023 |  Não | Não divisível por 4                |

## Algoritmo em Visualg

Algoritmo "exercicio21"

Var

ano : inteiro

Início

Escreva("Digite um ano: ")

Leia(ano)

Se ((ano % 4 = 0) e (ano % 100 <> 0)) ou (ano % 400 = 0) entao

    EscrevaL("O ano ", ano, " é BISSEXTO.")

Senao

    EscrevaL("O ano ", ano, " NAO é bissexto.")

FimSe

FimAlgoritmo

## Explicação passo a passo

 1  ano % 4 = 0

 → Verifica se o ano é divisível por 4

 2  ano % 100 <> 0

→ Garante que **anos como 1900 não sejam bissextos**

3 **ano % 400 = 0**

→ Exceção: anos como 2000 **são bissextos**

---

## 🧠 REGRA PRA DECORAR (prova)

- ✓ Divisível por 400 → bissexto
- ✓ Divisível por 4 e NÃO por 100 → **bissexto**
- ✗ Divisível por 100 → **não** (exceto 400)

## ◆ O que é uma Função?

Uma **função** é uma **rotina que executa um processamento e DEVOLVE um valor para quem a chamou.**

👉 Diferente do **procedimento**, a função **sempre retorna algo**.

Pense assim:

| ✅ Função = calcula algo e devolve o resultado

---

## ◆ Estrutura básica de uma Função

Toda função tem **4 partes principais**:

1. **Nome**
2. **Parâmetros (opcional)**
3. **Processamento**
4. **Retorno ( **retorne** )**

**Exemplo simples (pseudocódigo / Visualg):**

```
funcao dobro(numero: inteiro): inteiro
var
    resultado: inteiro
```

```
inicio  
    resultado ← numero * 2  
    retorne resultado  
fimfuncao
```

## ◆ O que está acontecendo aqui?

Vamos quebrar isso em partes ↗

### 1 Nome da função

funcão dobro(...)

- O nome é **dobro**
- Você vai usar esse nome depois para chamar a função

### 2 Parâmetros

(numero: inteiro)

- Parâmetros são **valores que a função recebe**
- Funcionam como **variáveis locais**
- Se você **não usar var**, o valor **não é alterado fora da função**

🔗 Exemplo:

A função recebe um número, faz a conta, mas não muda a variável original

### 3 Processamento interno

```
resultado ← numero * 2
```

- Aqui acontece a lógica
- Você pode usar:
  - contas
  - condições ( `se` )
  - repetições ( `para` , `enquanto` )
  - variáveis internas

## 4 Retorno ( `retorne` )

```
retorne resultado
```

### 💡 Isso é obrigatório em funções

- O `retorne` encerra a função
- Ele devolve o valor para quem chamou

## ◆ Como chamar uma Função?

Uma função pode ser usada **como se fosse um valor**.

**Exemplo:**

```
var  
  x: inteiro  
  
inicio  
  x ← dobro(5)  
  escreva(x)
```

fim

📌 Resultado:

10

👉 A função:

- Recebe 5
- Calcula 5 \* 2
- Retorna 10
- O valor 10 vai para x

## ◆ Função NÃO imprime (regra de ouro)

🚫 Função não serve para mostrar coisas na tela

✓ Serve para calcular e retornar valores

Errado:

```
escreva(resultado)
```

Certo:

```
retorne resultado
```

| Quem imprime é o programa principal ou um procedimento

## ◆ Função sem parâmetro (exemplo)

```

funcao numeroAleatorio(): inteiro
  inicio
    retorno 7
  fimfuncao

```

Uso:

```
x ← numeroAleatorio()
```

## ◆ Diferença clara: Função x Procedimento

| Função                   | Procedimento                 |
|--------------------------|------------------------------|
| Retorna valor            | Não retorna valor            |
| Usa <code>retorno</code> | Não usa <code>retorno</code> |
| Pode ser usada em contas | Não pode                     |
| Ex: cálculo              | Ex: mostrar mensagem         |

📌 Exemplo de uso:

```

Funcao Soma(A, B: Inteiro) : Inteiro
  var
    S: Inteiro
  Inicio
    S ← A + B
    Retorne S
  FimFuncao
  Inicio
    N1 ← 5
    N2 ← 4
    RES ← Soma(N1,N2)
    EscrevaL("A soma é ", RES)
  FimAlgoritmo

```

```

total ← soma(2, 3) // função
mostrarMensagem() // procedimento

```

## ◆ Quando usar Função?

Use função quando:

- ✓ Precisa de um **resultado**
- ✓ Vai usar esse cálculo várias vezes
- ✓ Quer código mais organizado
- ✓ Quer reutilizar lógica

## ◆ Frase pra você nunca esquecer

Função calcula e retorna.

Procedimento executa ações.

**Exemplo explicado:**

```
algoritmo "SomaFunção"
var
  V1, V2, S: Inteiro
  Funcao Soma(X, Y: Inteiro): Inteiro
  inicio
    Retorne X + Y
  FimFuncao
  inicio
    Escreva("Digite o primeiro valor: ")
    Leia(V1)
    Escreva("Digite o segundo valor: ")
    Leia(V2)
    S ← Soma(V1,V2)
    Escreva("O resultado da soma de ", V1, " e ", V2, " é ", S)
  finalgoritmo
```

## 📌 Código de exemplo: Função Soma

```
algoritmo "SomaFunção"
```

- ◆ Define o nome do algoritmo.

Serve só para identificação.

## ◆ Declaração de variáveis globais

```
var  
    V1, V2, S: Inteiro
```

Aqui temos **variáveis do algoritmo principal**:

- `V1` → primeiro valor digitado pelo usuário
- `V2` → segundo valor digitado pelo usuário
- `S` → vai guardar o **resultado retornado pela função**

📌 Essas variáveis **existem fora da função**.

## ◆ Declaração da Função

```
Funcao Soma(X, Y: Inteiro): Inteiro
```

Vamos quebrar isso 🤔

- `Soma` → nome da função
- `X, Y` → **parâmetros**
- `Inteiro` depois dos parênteses → **tipo do valor retornado**

📌 Essa função:

| recebe dois inteiros e retorna um inteiro

## ◆ Corpo da Função

```
inicio
    Retorne X + Y
FimFuncao
```

O que acontece aqui:

- `X + Y` → cálculo
- `Retorne` → devolve o resultado da soma
- Quando o `Retorne` executa, a função **termina**

 Regras importantes:

- Função **não usa** `Escreva`
- Função **apenas calcula e retorna**

## ◆ Início do algoritmo principal

```
inicio
```

A partir daqui começa o **programa principal**, que vai:

- conversar com o usuário
- chamar a função
- mostrar o resultado

## ◆ Entrada de dados

```
Escreva("Digite o primeiro valor: ")
Leia(V1)
```

- Mostra uma mensagem

- Guarda o valor digitado em `V1`

```
Escreva("Digite o segundo valor: ")  
Leia(V2)
```

- Mostra outra mensagem
- Guarda o valor digitado em `V2`

## ◆ Chamada da Função

```
S ← Soma(V1,V2)
```

Essa é a **linha mais importante do código** 🔥

O que acontece passo a passo:

1. `Soma(V1, V2)` chama a função
  2. `V1` vai para `X`
  3. `V2` vai para `Y`
  4. A função calcula `X + Y`
  5. O `Retorne` devolve o valor
  6. Esse valor é guardado em `S`
- 📌 A função funciona como se fosse um **valor**

## ◆ Saída de dados

```
Escreva("O resultado da soma de ", V1," e ", V2, " é ", S)
```

- Exibe os valores digitados

- Exibe o resultado que veio da função
- 📌 Quem imprime é o **algoritmo**, não a função.
- 

## ◆ Fim do algoritmo

```
fimalgoritmo
```

Encerra o programa.

---

## 🧠 Resumo mental pra lembrar depois

- ◆ A função Soma recebe valores
- ◆ Faz o cálculo  $X + Y$
- ◆ Usa `Retorne` para devolver o resultado
- ◆ O algoritmo chama a função
- ◆ O valor retornado vai para a variável `s`

Funções prontas no visual:

A chalkboard with a dark background and white chalk writing. It lists several functions and their outputs. In the top right corner, there is a drawing of a computer monitor displaying the word "Site" above the text "“CursoEmVideo”".

|                                 |                           |
|---------------------------------|---------------------------|
| <code>Compr(Site)</code>        | 12                        |
| <code>Copia(Site, 6, 2)</code>  | Em                        |
| <code>Maiusc(Site)</code>       | <code>CURSOEMVIDEO</code> |
| <code>Minusc(Site)</code>       | <code>cursoemvideo</code> |
| <code>Pos("Video", Site)</code> | 8                         |
| <code>Asc("C")</code>           | 67                        |
| <code>Carac(67)</code>          | C                         |

```

algoritmo "AnalizadorStrings"
var
  N: caractere
  C: inteiro
inicio
  escreva("Digite seu nome: ")
  Leia(N)
  Escreval("Total de letras do seu nome: ", Compr(N))
  Escreval("Seu nome em maiúsculas é: ", Maiusc(N))
  Escreval("Seu nome em minusculas é: ", Minusc(N))
  Escreval("A primeira letra do seu nomeé : ", Copia(N,1,1))
  Escreval("A última letra do seu nome é: ", Copia(Maiusc(N),Compr(N),1))
  Escreval("A letra A do seu nome fica na posição: ", Pos("A", Maiusc(N)))
  Escreval("O código da letra A é: ", Asc("A"))
  Escreval("A letra de codigo 65 é: ", Carac(65))

  Para C ← Compr(N) ate 1 passo -1 faca
    escreva(Copia(Maiusc(N), C,1))
  fimpara
fimalgoritmo

```

## Algoritmo: AnalisadorStrings

### Objetivo do algoritmo

Analizar um texto (nome) usando **funções prontas de string do Visualg**, sem criar nenhuma função manual.

### ◆ Declaração de variáveis

```

var
  N: caractere

```

C: inteiro

- `N` → guarda o **texto digitado** (o nome)
- `C` → contador usado no `para` (posição das letras)

## ◆ Entrada de dados

```
escreva("Digite seu nome: ")  
Leia(N)
```

O usuário digita um nome e ele fica armazenado em `N`.

## Funções prontas usadas (uma por uma)

Agora vem a parte mais importante 



**Compr(texto)**

→ Retorna a **quantidade de caracteres**

```
Escreval("Total de letras do seu nome: ", Compr(N))
```

 Se `N = "Marcus"` → retorna `6`



**Maiusc(texto)**

→ Converte o texto para **letras maiúsculas**

```
Escreval("Seu nome em maiúsculas é: ", Maiusc(N))
```

 "Marcus" → "MARCUS"

ab  
cd

## Minusc(texto)

→ Converte o texto para **letras minúsculas**

```
Escreval("Seu nome em minusculas é: ", Minusc(N))
```

 "Marcus" → "marcus"

scissors

## Copia(texto, inicio, quantidade)

→ Copia **parte do texto**

```
Escreval("A primeira letra do seu nome é: ", Copia(N,1,1))
```

 Posição começa em 1

 Copia **1 letra** a partir da posição 1

 Retorna a **primeira letra**

## ◆ Última letra do nome

```
Escreval("A última letra do seu nome é: ", Copia(Maiusc(N),Compr(N),1))
```

Aqui acontece uma combinação de funções:

1. `Maiusc(N)` → garante maiúscula
2. `Compr(N)` → pega a última posição
3. `Copia(..., 1)` → copia 1 caractere

 Resultado: **última letra do nome**



## Pos(letra, texto)

→ Retorna a **posição** da letra no texto

```
Escreval("A letra A do seu nome fica na posição: ", Pos("A", Maiusc(N)))
```

📌 Converte o nome para maiúsculo

📌 Procura "A"

📌 Retorna a posição da **primeira ocorrência**

⚠ Se não existir, geralmente retorna 0.

12  
34

## Asc(caractere)

→ Retorna o **código ASCII** do caractere

```
Escreval("O código da letra A é: ", Asc("A"))
```

📌 "A" → 65

AB  
CD

## Carac(código)

→ Retorna o **caractere** a partir do código ASCII

```
Escreval("A letra de codigo 65 é: ", Carac(65))
```

📌 65 → "A"



## Estrutura de repetição com string

◆ **Para invertido (do fim para o início)**

```
Para C ← Compr(N) ate 1 passo -1 faca  
    escreva(Copia(Maiusc(N), C,1))  
fimpara
```

### O que acontece aqui:

- Começa da **última letra**
- Vai até a **primeira**
- Passo **1** → anda para trás
- Em cada volta:
  - pega **uma letra**
  - imprime na tela

📌 Resultado: o nome **aparece ao contrário**, letra por letra.

## 📌 Classificação das Variáveis (Simples x Compostas)

### ◆ Variáveis Simples

- Guardam **apenas um valor**
- Ex: **inteiro** , **real** , **caractere** , **lógico**

```
idade: inteiro
```

### ◆ Variáveis Compostas

- Guardam **vários valores**
- Ex: vetores, matrizes



# Classificação das Variáveis Compostas

As variáveis compostas podem ser classificadas de várias formas ↗

**VARIÁVEIS COMPOSTAS  
HOMOGENEAS**

```
var
    n1: inteiro
    n2, n3, n4: inteiro
    n: vetor[1..4] de inteiro
inicio
    n[1] <- 3
    n[2] <- 5
    n[3] <- 1
    n[4] <- 0
```

|    |    |    |    |    |
|----|----|----|----|----|
|    | n1 | n2 | n3 | n4 |
| n3 |    |    |    |    |
| n2 |    |    |    |    |
| n1 |    |    |    |    |
| n  | 3  | 5  | 1  | 0  |
|    | 1  | 2  | 3  | 4  |

<https://youtu.be/j9473xQ39vY>

```
algoritmo "TesteVetor"
var
    v: vetor [1..6] de Inteiro
    c: inteiro
inicio
    para c ← 1 ate 6 faca
        escreva("Digite o ", c, "o. valor:")
        leia(v[c])
    fimpara
    para c ← 1 ate 6 faca
        escreva("{",v[c],"}")
```

```
fimpara  
fimalgoritmo
```

Se for aumentar a quantidade é só alterar na variável por exemplo [1..10] e nas estruturas de repetição no caso do para seria 1 ate 10

## 1 Homogêneas x Heterogêneas

### ◆ Variáveis Compostas Homogêneas

✓ Todos os valores são do **mesmo tipo**

📌 Exemplo:

```
notas: vetor[1..5] de real
```

- Todas as posições guardam **real**

### ➡ Vetores e matrizes são homogêneos

### ◆ Variáveis Compostas Heterogêneas

✓ Guardam **tipos diferentes de dados** na mesma estrutura

📌 Exemplo (registro / struct):

```
aluno: registro  
    nome: caractere  
    idade: inteiro  
    media: real  
fimregistro
```

- Cada campo tem um tipo diferente

📌 No Visualg isso é chamado de **registro**

## 🧠 Regra de prova

- ◆ Vetor → homogêneo
- ◆ Matriz → homogênea
- ◆ Registro → heterogênea

## 2 Unidimensionais x Multidimensionais

### ◆ Variáveis Compostas Unidimensionais

- ✓ Possuem **apenas uma dimensão**
- ✓ São acessadas por **um índice**

↗ Exemplo:

nomes: vetor[1..10] de caractere

→ nomes[3]

◆ Chamamos isso de **vetor**

### ◆ Variáveis Compostas Multidimensionais

- ✓ Possuem **mais de uma dimensão**
- ✓ São acessadas por **mais de um índice**

↗ Exemplo:

notas: vetor[1..3, 1..2] de real

→ notas[2,1]

◆ Chamamos isso de **matriz**

### 🧠 Regra rápida

| Tipo   | Dimensões |
|--------|-----------|
| Vetor  | 1         |
| Matriz | 2 ou mais |

## 3 Estáticas x Dinâmicas (conceito geral)

| Esse conceito aparece mais em linguagens reais, mas é bom conhecer.

### ◆ Estáticas

- Tamanho definido **na declaração**
- Não muda durante o programa

📌 Visualg usa **vetores estáticos**

```
vetor[1..5] de inteiro
```

### ◆ Dinâmicas

- Tamanho pode mudar
- Usado em linguagens como Python, JS

```
lista = []
lista.append(10)
```

## 4 Sequenciais x Associativas (extra – nível avançado)

### ◆ Sequenciais

- Acesso por **posição**
- Ex: vetores, matrizes

---

## ◆ Associativas

- Acesso por **chave**
- Ex: dicionários (Python), objetos (JS)

```
aluno = {"nome": "Marcus", "idade": 20}
```



## Tabela-resumo (perfeita pra Notion)

Variáveis Compostas:

Homogêneas:

- Mesmo tipo de dado
- Vetores
- Matrizes

Heterogêneas:

- Tipos diferentes
- Registros

Unidimensionais:

- 1 índice
- Vetores

Multidimensionais:

- 2 ou mais índices
- Matrizes



## Frase-chave pra memorizar

Vetor é homogêneo e unidimensional.  
Matriz é homogênea e multidimensional.  
Registro é heterogêneo.

## O que é um vetor (em linguagem simples)

Um **vetor** é como uma **gaveta com várias divisões**.

- Todas as divisões guardam **o mesmo tipo de dado**
- Cada divisão tem um **número (índice)**
- Você acessa cada valor usando esse número

No seu caso:

```
v: vetor [1..6] de inteiro
```

Isso significa:

- ➔ Você criou **6 variáveis inteiros**
- ➔ Todas se chamam `v`
- ➔ Mas cada uma tem um número:

```
v[1]  
v[2]  
v[3]  
v[4]  
v[5]  
v[6]
```

É como se fosse isso (mas de forma organizada):

```
v1, v2, v3, v4, v5, v6 : inteiro
```

Só que MUITO melhor 😊

## 📌 Analisando seu código linha por linha

### ◆ Declaração das variáveis

```
var  
v: vetor [1..6] de Inteiro  
c: inteiro
```

- `v` → vetor com **6 posições**
- `c` → contador do `para`, usado para percorrer o vetor

### 🔄 PRIMEIRO `para` → LER os valores

```
para c ← 1 ate 6 faca  
    escreva("Digite o ", c, "o. valor:")  
    leia(v[c])  
fimpara
```

### O que está acontecendo aqui?

O `para` vai repetir **6 vezes**:

| Volta          | Valor de <code>c</code> | O que acontece                  |
|----------------|-------------------------|---------------------------------|
| 1 <sup>a</sup> | 1                       | Lê valor para <code>v[1]</code> |
| 2 <sup>a</sup> | 2                       | Lê valor para <code>v[2]</code> |
| 3 <sup>a</sup> | 3                       | Lê valor para <code>v[3]</code> |
| 4 <sup>a</sup> | 4                       | Lê valor para <code>v[4]</code> |
| 5 <sup>a</sup> | 5                       | Lê valor para <code>v[5]</code> |
| 6 <sup>a</sup> | 6                       | Lê valor para <code>v[6]</code> |

## Importante:

`v[c]` significa:

| "Guarde o valor na posição c do vetor"

## SEGUNDO `para` → MOSTRAR os valores

```
para c ← 1 ate 6 faca  
    escreva("{"v[c]","}")  
fimpara
```

Aqui você está:

-  Percorrendo o vetor de novo
-  Mostrando cada valor armazenado

Se o usuário digitou:

```
5 7 8 6 9 10
```

A saída será:

```
{5}{7}{8}{6}{9}{10}
```

## Exemplo prático: notas da escola

Você pensou certo 

Vetores são PERFEITOS para notas.

Exemplo:

```
var  
notas: vetor[1..6] de real  
i: inteiro
```

- `notas[1]` → nota do aluno 1
- `notas[2]` → nota do aluno 2
- ...
- `notas[6]` → nota do aluno 6

A partir disso você pode:

- calcular média
- achar maior nota
- achar menor nota
- contar aprovados

Tudo com **um único vetor**.

---

## Ideia importante pra fixar

👉 **Vetor não “cria várias variáveis diferentes”**

👉 Ele cria **uma variável com várias posições**

E o `para` serve exatamente para:

| passar por todas essas posições sem repetir código

---

## Regra de ouro (guarda isso)

- Vetor → guarda **muitos valores**
- Índice (`[c]`) → diz **qual posição**
- `para` → percorre o vetor

- Tudo dentro do vetor tem **o mesmo tipo**

## Valores pares com vetores compostos:

```

algoritmo "ValoresPares"
var
  val: vetor[1..7] de Inteiro
  i,TotPar: Inteiro
inicio
  TotPar ← 0
  Para i ← 1 ate 7 faca
    Escreva("Digite o ", i, "o. valor: ")
    leia(val[i])
  FimPara

  Para i ← 1 ate 7 faca
    se(val[i] % 2 = 0)entao
      TotPar ← TotPar + 1
      Escreval("Valor PAR na posição ",i)
    fimse
  fimpara
  EscrevaL("O total de pares foi ", TotPar)
fimalgoritmo

```

## Notas Alunos com vetores:

```

algoritmo "Turma"
var
  nome: vetor[1..4] de Caractere
  n1: vetor [1..4] de real
  n2: vetor [1..4] de real
  media: vetor [1..4] de real
  i: Inteiro
inicio
  para i ← 1 ate 4 faca

```

```

Escreval("Aluno ",i)
escreva("Nome: ")
leia(nome[i])
escreva("Primeira nota: ")
leia(n1[i])
escreva("segunda nota: ")
leia(n2[i])
media[i] ← (n1[i]+n2[i])/2
fimpara

limpatela
escreval("Listagem de Alunos")
escreval("=====")
Para i ← 1 ate 4 faca
escreval(nome[i]:15, media[i]:4:1)
fimpara
fimalgoritmo

```

**Mesmo código de antes porem com adicional de quais alunos acima da média:**

```

algoritmo "Turma"
var
  nome: vetor[1..4] de Caractere
  n1: vetor [1..4] de real
  n2: vetor [1..4] de real
  media: vetor [1..4] de real
  SM, MT: REAL
  i,Tot: Inteiro
inicio
  para i ← 1 ate 4 faca
    Escreval("Aluno ",i)
    escreva("Nome: ")
    leia(nome[i])
    escreva("Primeira nota: ")
    leia(n1[i])
    escreva("segunda nota: ")

```

```

leia(n2[i])
media[i] ← (n1[i]+n2[i])/2
SM ← SM + media[i]
fimpara
MT ← SM/4
limpatela
escreval("Listagem de Alunos")
escreval("=====")
Para i ← 1 ate 4 faca
escreval(nome[i]:15, media[i]:4:1)
se (media[i] > MT) entao
    Tot ← Tot + 1
fimse
fimpara
escreval("Ao todo temos ", Tot, " alunos acima da media que é ",MT:4:1)
fimalgoritmo

```

## Só com começo C

```

algoritmo "SoComC"
var
    nome: caractere
    soC: vetor[1..10] de caractere
    c, tot: Inteiro
inicio
    tot ← 0
    para c ← 1 ate 10 faca
        escreva("Digite seu nome: ")
        leia(nome)
        se (copia(maisc(nome),1,1) = "C") entao
            tot ← tot + 1
            soC[tot] ← nome
        fimse
        LimpaTela
        Escreval("Listagem Final")

```

```
para c ← 1 ate tot faca
    escreval(soC[c])
fimpara
fimalgoritmo
```

### ◆ algoritmo "SoComC"

Define o nome do algoritmo.

Aqui a ideia já fica clara: **guardar só nomes que começam com C**.

### ◆ nome: caractere

Variável simples para **ler um nome por vez**.

💡 Importante:

No Visualg, **caractere** pode guardar **texto (string)**, não só uma letra.

### ◆ soC: vetor[1..10] de caractere

Cria um vetor com **10 posições**, que vai guardar **somente os nomes que começam com C**.

Exemplo mental:

```
soC[1], soC[2], soC[3] ... soC[10]
```

### ◆ c, tot: Inteiro

- **c** → contador do **para**
- **tot** → **quantidade de nomes válidos** (que começam com C)

### ◆ tot <- 0

Inicializa o contador.

📌 Regra de ouro:

| Todo contador começa em 0

## ➡ PRIMEIRO para — leitura e filtro

```
para c ← 1 ate 10 faca
```

Esse laço:

- roda **10 vezes**
- permite digitar **10 nomes**

◆ escreva("Digite seu nome: ")

Mostra a mensagem para o usuário.

◆ leia(nome)

Lê o nome digitado e guarda na variável nome .

◆ se (copia(maiusc(nome),1,1) = "C") entao

Essa é a **linha mais importante do código**. Vamos quebrar ela:

1 maiusc(nome)

Transforma o nome em **maiúsculo**

Exemplo:

```
"carlos" → "CARLOS"
```

👉 Isso evita erro se o usuário digitar carlos ou Carlos .

2 copia(maiusc(nome),1,1)

- `copia(texto, início, quantidade)`
- aqui você está pegando:
  - **a partir da posição 1**
  - **1 caractere apenas**

Ou seja:

"Carlos" → "C"

"Ana" → "A"

 = "C"

Compara a primeira letra com `"C"`.

 Resultado:

- Se começar com **C**, entra no `se`
- Se não, ignora

 `tot <- tot + 1`

Se o nome começa com C:

- aumenta o contador de nomes válidos

Exemplo:

`tot = 0` → 1 → 2 → 3

 `soC[tot] <- nome`

Guarda o nome no vetor.

 **Esse detalhe é muito importante:**

Você **não usa** `c`, você usa `tot`.

👉 Isso faz com que:

- o vetor fique **sem espaços vazios**
- só tenha nomes válidos

Exemplo:

```
soC[1] = "Carlos"  
soC[2] = "Clara"
```

◆ **fimse / fimpara**

Finaliza o **se** e o laço de leitura.

📌 Até aqui:

- Você leu 10 nomes
- Guardou **apenas os que começam com C**
- **tot** diz quantos foram guardados



**LimpaNtela**

LimpaNtela para mostrar o resultado final organizado.

◆ **Escreval("Listagem Final")**

Mostra o título da listagem.

⌚ **SEGUNDO para — mostrar os nomes**

```
para c ← 1 ate tot faca
```



## O que significa ordenar um vetor?

Ordenar um vetor é **colocar os valores em ordem**.

Exemplos:

- Crescente: **1 3 5 7 9**
- Decrescente: **9 7 5 3 1**

📌 Isso não acontece sozinho.

O programa precisa **comparar valores e trocar de lugar** quando necessário.

---



## Ideia básica da ordenação

Para ordenar, você precisa:

1. Comparar dois valores
2. Ver se estão fora de ordem
3. Trocar eles de lugar
4. Repetir isso várias vezes

👉 Por isso usamos **dois para + uma variável auxiliar**

---



## Estrutura padrão para ordenar vetor

Você SEMPRE vai ver isso:

- Vetor
  - Dois contadores (**i** e **j**)
  - Uma variável auxiliar (**aux**)
- 



## Exemplo simples (ordem crescente)

Vamos ordenar 5 números.

---



## Código completo (Bubble Sort – o mais didático)

```

algoritmo "OrdenaVetor"
var
    v: vetor[1..5] de inteiro
    i, j, aux: inteiro
inicio
    // leitura
    para i ← 1 ate 5 faca
        escreva("Digite o ", i, "o. valor: ")
        leia(v[i])
    fimpara

    // ordenação
    para i ← 1 ate 4 faca
        para j ← i+1 ate 5 faca
            se (v[i] > v[j]) entao
                aux ← v[i]
                v[i] ← v[j]
                v[j] ← aux
            fimse
        fimpara
    fimpara

    // exibição
    escreval("Vetor ordenado:")
    para i ← 1 ate 5 faca
        escreval(v[i])
    fimpara
fimalgoritmo

```



## Agora a explicação linha a linha da parte MAIS IMPORTANTE

---



## aux: inteiro

Variável auxiliar para **guardar temporariamente um valor**.

📌 Sem ela, você perde um dos valores na troca.

## ⟳ Laços de ordenação

para i ← 1 ate 4 faca

- **i** representa a **posição atual**
- vai até **4** porque a última posição não precisa comparar

para j ← i+1 ate 5 faca

- **j** começa **depois de i**
- compara **v[i]** com todos à frente

## ◆ Condição de comparação

se ( $v[i] > v[j]$ ) entao

Se o valor da posição **i** for **maior** que o da posição **j**,  
então eles estão **fora de ordem crescente**.

## ⟳ Troca de valores (parte crucial)

```
aux ← v[i]
v[i] ← v[j]
```

```
v[j] ← aux
```

Passo a passo:

1. Guarda  $v[i]$  em  $aux$
2. Coloca  $v[j]$  em  $v[i]$
3. Coloca o valor guardado em  $v[j]$

 Isso é a **troca segura**.

---



## Visualização mental

Antes:

```
v[i] = 9  
v[j] = 3
```

Depois da troca:

```
v[i] = 3  
v[j] = 9
```



## Ordenação decrescente (muda só um sinal)

```
se ( $v[i] < v[j]$ ) entao
```

O resto é IGUAL.

---

# 📌 Regra de ouro pra ordenar vetor

- ✓ Sempre dois **para**
- ✓ Sempre uma variável **aux**
- ✓ Comparar e trocar
- ✓ Repetir até tudo estar em ordem

## Vídeo Vetores:

<https://youtu.be/j9473xQ39vY>

## Ordenar Valor

```
algoritmo "OrdenaVetor"  
var  
    vet:vetor[1..4] de Inteiro  
    i, j, aux: inteiro  
inicio  
    para i ← 1 ate 4 faca  
        escreva("Digite um valor: ")  
        Leia(vet[i])  
    fimpara  
  
    para i ← 1 ate 3 faca  
        para j ← i + 1 ate 4 faca  
            se(vet[i] > vet[j]) entao  
                aux ← vet[i]  
                vet[i] ← vet[j]  
                j ← vet[j] ← aux  
            fimse  
        fimpara  
    fimpara
```

```
para i ← 1 ate 4 faca
    escreva("{"vet[i],"}")
fimpara
fimalgoritmo
```



## O que é uma matriz?

<https://youtu.be/hkE9WrjpAAk>

Uma **matriz** é um **vetor de vetores**.

Enquanto o vetor tem **uma dimensão**:

v[1] v[2] v[3]

A matriz tem **duas dimensões**:

m[linha][coluna]

Pensa assim 

 **tabela / planilha / grade**

 **Exemplo visual (3x3)**

## MAIS DIMENSÕES

```
var  
    m: vetor[1..3, 1..2] de inteiro  
  
inicio  
    m[1,2] <- 4  
    m[2,2] <- 5  
    m[3,1] <- 8
```

|   |   |   |   |  |  |  |  |
|---|---|---|---|--|--|--|--|
|   |   | m |   |  |  |  |  |
| 1 |   |   |   |  |  |  |  |
| 2 |   |   | 4 |  |  |  |  |
| 3 |   |   | 5 |  |  |  |  |
|   | 1 |   |   |  |  |  |  |
|   |   | 2 |   |  |  |  |  |

Colunas →

123

```
+-----+  
L1 |5|8|2|  
i +-----+  
n2 |7|1|9|  
h +-----+  
a3 |4|6|3|  
+-----+
```

Acesso:

- `m[1][1]` → 5
- `m[2][3]` → 9



## Como declarar uma matriz (Visualg)

```
var  
m: vetor[1..3, 1..3] de inteiro
```

📌 Leitura:

| matriz de 3 linhas e 3 colunas



## Percorrendo uma matriz (2 para )

⚠ Regra de ouro:

| matriz = dois laços para

```
para i ← 1 ate 3 faca  
    para j ← 1 ate 3 faca  
        // trabalha com m[i][j]  
    fimpara  
fimpara
```

- `i` → linha
- `j` → coluna



## Exemplo completo: ler e mostrar matriz 2x2

```
algoritmo "ExemploMatriz"  
var  
m: vetor[1..2, 1..2] de inteiro  
i, j: inteiro  
inicio
```

```

// leitura
para i ← 1 ate 2 faca
    para j ← 1 ate 2 faca
        escreva("Digite o valor [", i, ",", j, "]: ")
        leia(m[i][j])
    fimpara
fimpara

// exibição
escreval("Matriz:")
para i ← 1 ate 2 faca
    para j ← 1 ate 2 faca
        escreva(m[i][j], " ")
    fimpara
    escreval("")
fimpara
fimalgoritmo

```

## Explicando o essencial

### ◆ **m[i][j]**

- **i** → linha
- **j** → coluna
- acessa **uma célula específica**

### ◆ Por que dois **para** ?

Porque você precisa:

- um laço para as **linhas**
- outro para as **colunas**

## Ligando com vetor (comparação direta)

| Vetor  | Matriz  |
|--------|---------|
| v[i]   | m[i][j] |
| 1 para | 2 para  |
| Lista  | Tabela  |



## Exemplo prático: soma dos elementos

```
soma ← 0
para i ← 1 ate 3 faca
    para j ← 1 ate 3 faca
        soma ← soma + m[i][j]
    fimpara
fimpara
```



## Exemplo clássico: diagonal principal

```
para i ← 1 ate 3 faca
    escreval(m[i][i])
fimpara
```

Porque:

- linha = coluna



## Regras mentais pra nunca errar

- ✓ Matriz = tabela
- ✓ Dois índices  $[i][j]$
- ✓ Dois **para**

✓ Mesmo tipo de dado

✓ [linha][coluna]

## Matriz Par

```
algoritmo "matrizPar"
var
    valores : vetor[1..3,1..3]de inteiro
    l,c,TotPar: inteiro
inicio
    para l ← 1 ate 3 faca
        para c ← 1 ate 3 faca
            escreva("Digite o valor da posição [",l,";",c,":]:" )
            leia (valores[l,c])
        fimpara
    fimpara
    escreval()
    escreval("MATRIZ:")
    escreval("=====")
    =====)
    TotPar ← 0
    para l ← 1 ate 3 faca
        para c ← 1 ate 3 faca
            se (valores[l,c] % 2 = 0)entao
                escreva("{",valores[l,c]:2,"}")
                TotPar ← TotPar + 1
            senao
                escreva(valores[l,c]:4)
            fimse
        fimpara
        escreva()
    fimpara
    Escreval("Ao todo foram digitados ",TotPar, " valores pares.")
fimalgoritmo
```

## Matriz Identidade

```
algoritmo "matrizIdentidade"
var
    mID: vetor[1..3,1..3] de Inteiro
    i, j: inteiro
inicio
    para i ← 1 ate 3 faca
        para j ← 1 ate 3 faca
            se (i=j) entao
                mID[i,j] ← 1
            senao
                mID[i,j] ← 0
            fimse
        fimpara
    fimpara

    para i ← 1 ate 3 faca
        para j ← 1 ate 3 faca
            escreva(mID[i,j]:3)
        fimpara
    escreval()
fimpara
fimalgoritmo
```

## Valores Matriz

```
algoritmo "valoresMatriz"
var
    m: vetor[1..4,1..4] de Inteiro
    l, c, sDP, p2L, mai3C: Inteiro
inicio
    sDP ← 0
    p2L ← 1
    para l ← 1 ate 4 faca
```

```

para c ← 1 ate 4 faca
    escreva("Digite o valor da Posição [",l,";",c,":]")
    leia(m[l,c])
    se(l = c) entao
        sDP ← sDP + m[l,c]
    fimse
fimpara
fimpara

para l ← 1 ate 4 faca
    para c ← 1 ate 4 faca
        escreva(m[l,c])
    fimpara
    escreval()
fimpara

para c ← 1 ate 4 faca
    p2L ← p2L * m[2,c]
fimpara

para l ← 1 ate 4 faca
    se(m[l,3] > mai3C)entao
        mai3C ← m[l,3]
    fimse
fimpara

Escreval("A soma dos valores da Diagonal Principal é ",sDP)
escreval("O produto dos valores da Segunda Linha é ", p2L)
escreval("O maior valor da Terceira Coluna é ", mai3C)
fimalgoritmo

```

## Dissencando Matrizes

```

algoritmo "Dissecando_ Matrizes"
var

```

```

m : vetor[1..4,1..4] de inteiro
c,I,D,S : inteiro
inicio
  S ← 0
    para c ← 1 ate 4 faca
      para I ← 1 ate 4 faca
        escreva("Digite o valor [", I, ",", c, "]: ")
        leia(m[c,I])
      fimpara
    fimpara
  LimpaTela
  repita
    escreval("")
    escreval("MENU DE OPÇÕES")
    escreval("===== ")
    EscrevaL("[1] Mostrar a Matriz ")
    EscrevaL("[2] Diagonal Principal ")
    EscrevaL("[3] Triângulo Superior ")
    EscrevaL("[4] Triângulo Inferior ")
    EscrevaL("[5] Sair ")
    EscrevaL("===== OPÇÃO: ")
    Leia(D)
    Escolha D
    Caso 1
      LimpaTela
      escreval("Matriz:")
      para c ← 1 ate 4 faca
        para I ← 1 ate 4 faca
          escreva(m[c,I], " ")
        fimpara
      escreval("")
      fimpara
    Caso 2
      LimpaTela
      escreval("Diagonal Principal:")
      para I ← 1 ate 4 faca

```

```

para c ← 1 ate 4 faca
  se (l = c) entao
    escreva(m[l,c]:4)
  senao
    escreva("  ")
  fimse
fimpara
escreval("")
```

fimpara

### Caso 3

LimpaTela

```

  escreval("Triângulo Superior:")
para l ← 1 ate 4 faca
  para c ← 1 ate 4 faca
    se (l < c) entao
      escreva(m[l,c]:4)
    senao
      escreva("  ")
    fimse
fimpara
escreval("")
```

fimpara

### Caso 4

LimpaTela

```

  escreval("Triângulo Inferior:")
para l ← 1 ate 4 faca
  para c ← 1 ate 4 faca
    se (l > c) entao
      escreva(m[l,c]:4)
    senao
      escreva("  ")
    fimse
fimpara
escreval("")
```

fimpara

### Caso 5

```
escreva("ENCERRANDO...")
S ← S + 1
FimEscolha
ate S >= 1
fimalgoritmo
```