

## Introdução à análise de dados e programação

# Variáveis, Tipos de Dados, Operadores e Comando de Decisão

Copyright © CESAR School 2023 | Todos os direitos reservados.

Este material, ou qualquer parte dele, não pode ser reproduzido, divulgado ou usado de forma alguma sem autorização escrita.

# Olá!

Eu sou Taty Calixto

Doutora em Ciência da Computação, atuo no CESAR School como Consultora de Qualificação e professora do curso de Ciência da Computação






# Tutorial

Outra opção:

## Portugol Mobile (para Android)






### Portugol Mobile

Erick Leonardo Weil Produtividade

★★★★★ 591

Adicionar à lista de desejos

Instalar



Faça programas em seu celular ou tablet sem grandes problemas, não se preocupe com coisas como perder seus códigos ou acesso à internet, o aplicativo não depende d internet para funcionar.

Utilize dos Exemplos para explorar o que o aplicativo é capaz de fazer

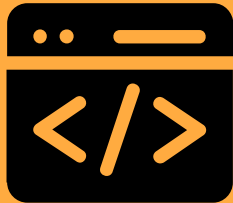
Caso queira contribuir ou tenha encontrado algum problema, adicione uma issue no repositório do





# O que veremos hoje?

- Tipos de Dados
  - Variáveis
  - Operadores
  - Comandos de Decisão
- 



# Programar

- Na programação, uma variável é um objeto (uma posição, frequentemente localizada na memória) capaz de reter e representar um valor ou expressão.
- “Espaço” em memória para guardar um valor durante a execução de um programa.
- Devem ser declaradas antes de serem utilizadas



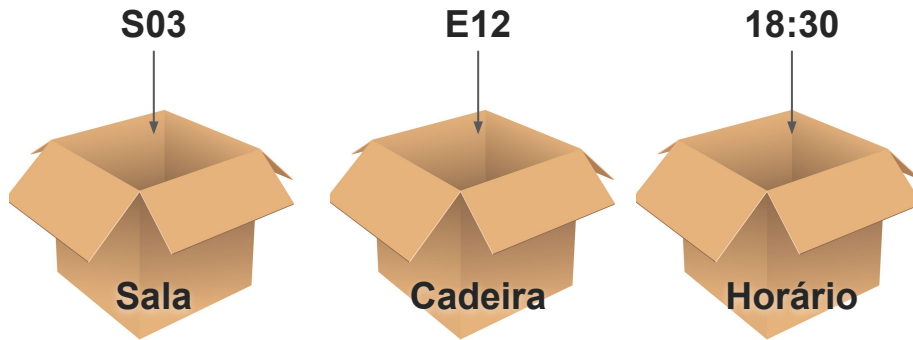
**Memória**



**Declarando variável**

Exemplo:

Você tem um programa que realiza a venda de ingressos para um cinema. Neste caso, em uma operação de compra, você precisará armazenar em uma variável qual a sala, em outra variável a cadeira escolhida pelo cliente, em outra o horário da sessão...



Elas podem ser classificadas em três categorias:

- Numéricas;
- Textual;
- Lógicas.

## Dados Cadastrais

Nome: João Guilherme  
Idade: 30  
Endereço: Rua João Pinho, 123  
Peso: 85,5  
Altura: 1,90





1 2  
3 4

# Variáveis Numéricas

Podem ser divididos em dois tipos:


- **Tipo inteiro** - Podem ser positivos, negativos ou nulos, mas não possuem componente decimal:  
*Ex.: 5; -9; 0; 189; -800.*
- **Tipo real** - Podem ser positivos, negativos ou nulos, e possuem componente decimal, marcado pelo ponto (.):  
*Ex.: 5.89; -6.8978; 0.00; 7.986; -1458.252.*

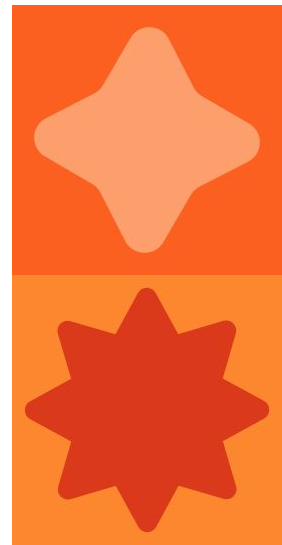
**i** *Os inteiros são compatíveis com os reais, mas os reais não são compatíveis com os inteiros;*

**i** *Os inteiros consomem menos espaço de armazenamento na memória.*

Armazenam valores de texto. Podem ser divididos em dois tipos:

- **Tipo caracter** - Contém uma informação com apenas um caracter. Esse caracter pode ser uma letra, número ou pontuação.  
Ex.: `'a'`; `'7'`; `'!'`;
- **Tipo cadeia** - Contém uma informação composta por vários caracteres, como um nome ou um endereço.  
Ex.: `"Em exemplo de texto"`

 Caracteres são definidos usando aspas simples ('a') e cadeias são definidos usando aspas duplas ("exemplo").



Armazena dados booleanos, ou seja, eles podem assumir dois valores possíveis: **verdadeiro** ou **falso**.





# Declaração e atribuição de variáveis

No código, a criação de uma variável é chamada de **declaração**. Ao colocar um valor na variável, chamamos de **atribuição** (ou inicialização). Geralmente, as declarações ocorrem no topo do programa.

```
programa
```

```
{
```

```
funcao inicio()
```

```
{
```

```
inteiro idade
```

```
real altura
```

```
caracter turma
```

```
cadeia nome
```

```
logico dirige
```

```
}
```

```
}
```

Aqui as variáveis estão sendo declaradas, mas não estão recebendo valores



# Declaração e atribuição de variáveis

No código, a criação de uma variável é chamada de **declaração**. Ao colocar um valor na variável, chamamos de **atribuição** (ou inicialização). Geralmente, as declarações ocorrem no topo do programa.

```
programa
```

```
{
```

```
funcao inicio()
```

```
{
```

```
inteiro idade
```

```
real altura = 1.72
```

```
idade = 30
```

```
}
```

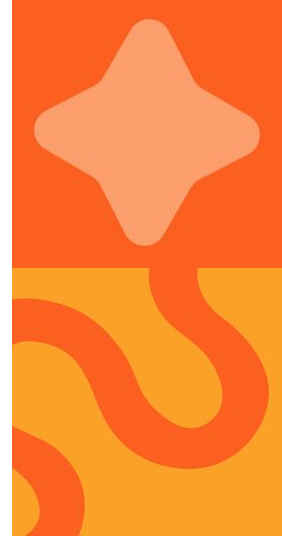
```
}
```

Declaração

Declaração com  
atribuição

Atribuição

Recebe





## ! Não é possível atribuir um valor de um tipo a uma variável de outro tipo

A partir do momento em que uma variável é declarada com um tipo, não é mais possível mudar.

```
programa
```

```
{
```

```
    funcao inicio()
```

```
    {
```

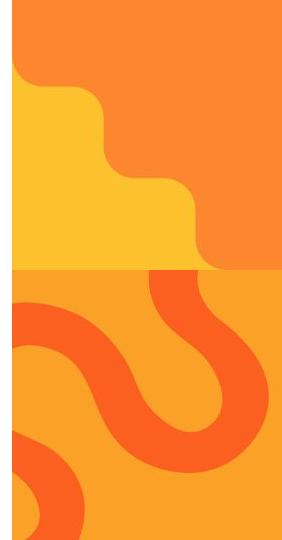
```
        inteiro idade
```

```
        real altura = 's' ❌
```

```
        idade = verdadeiro ❌
```

```
    }
```

```
}
```





# Declaração e atribuição múltipla

É possível declarar várias variáveis de um mesmo tipo em uma única linha.  
O mesmo é possível fazer com a atribuição.

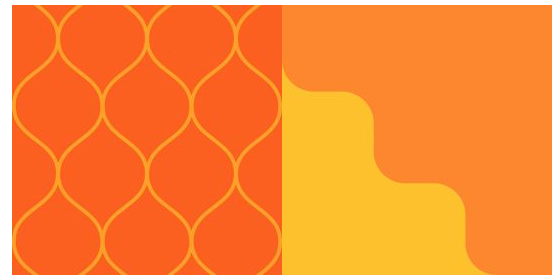
```
programa
{
    funcao inicio()
    {
        inteiro n1, n2, n3
        real largura = 16.0, altura = 8.5
    }
}
```



# Nomeando Variáveis

Há algumas regras para nomear variáveis:

- O primeiro caractere deve ser uma **letra** ou `_`;
- Não é permitido o uso de caracteres especiais (exceto o `_`);
- Espaços não são permitidos;
- Não deve ser utilizado os nomes reservados da linguagem de programação utilizada;
- Duas variáveis, mesmo que de tipos diferentes, não podem ter o mesmo nome.







# Nomeando Variáveis

Regras de estilo/dicas:

- Não utilizar acentuação;
- Letras minúsculas apenas para variáveis;
- Letras MAIÚSCULAS apenas para constantes;
- Palavras compostas: separadas por underline (`_`)  
Ex.: `nome_completo`, `dia_semana`,  
`horario_intervalo`
- Case sensitive:  
`exemplo`  $\neq$  `Exemplo`



# Constantes

@CESAR 2019 | Todos os Direitos Reservados

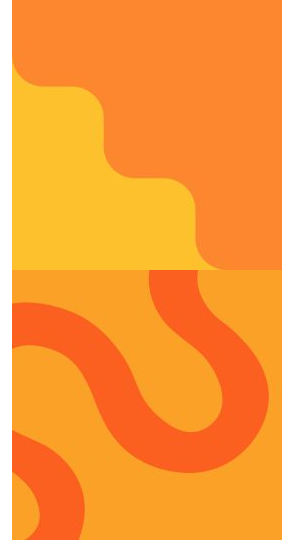
Existem algumas situações em que precisamos trabalhar com um determinado parâmetro que não é alterado durante a execução do programa. Para isso existem as constantes.

Constante é um identificador cujo valor associado não pode não pode ser alterado pelo programa durante a execução.

```
const inteiro ALTURA_MAXIMA = 190
const real PI = 3.14
const real PLANCK = 6.62607
const real ACELERACAO_GRAVIDADE = 9.8
```



O nome de uma constante deve ser escrito todo em maiúsculo!



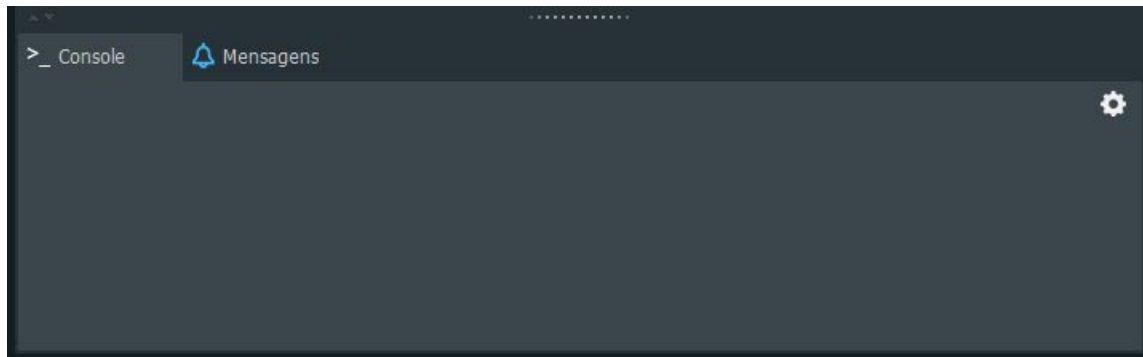


# Exibindo informações no console

@CESAR 2019 | Todos os Direitos Reservados

Quando um programa executa algum procedimento, é comum que ele mostre alguma saída para o usuário.

No Portugal, usaremos a saída padrão chamada de console:



O console serve tanto para saída, como para entrada de dados.



# Exibindo informações no console

O comando usado para exibir informações no console é o:

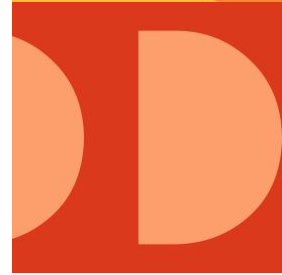
```
escreva ("Olá mundo!")
```

É possível exibir o valor de uma variável no console passando-a como parâmetro:

```
inteiro idade = 30  
escreva(idade)
```

Ainda é possível passar vários valores, separados por vírgula:

```
inteiro idade = 30  
escreva("Sua idade é ", idade)
```





# Recebendo informações no console

O comando usado para receber informações no console é o:

```
leia(nome_variavel)
```

Com esse comando, é possível receber um valor do usuário e atribuí-lo a uma variável:

```
inteiro idade  
leia(idade)  
escreva("Sua idade é ", idade)
```





# Resumo:

Tipos de variáveis:

- **inteiro, real, caracter, cadeia, lógico**

Declaração de uma variável:

- **[tipo] [nome]**

Atribuição:

- **[nome] = [valor]**

Declaração com atribuição:

- **[tipo] [nome] = [valor]**

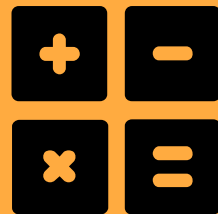
Variável x Constante:

- **Variável:** o valor pode mudar ao decorrer da execução do código
- **Constante:** o valor não muda depois que foi declarado

Exibir e receber informações:

- **escreva()**
- **leia()**





# Operadores

# É possível realizar operações sobre as variáveis e sobre algumas constantes:

- **Atribuição:** guardar valor em uma variável

=

- **Aritmética:** soma, subtração, multiplicação, divisão, resto da divisão

+

-

\*

/

%

- **Incremento:** incrementa o valor de uma variável

++

--

+=

-=

\*=

- **Relacional:** maior, menor, igual, diferente, maior ou igual, menor ou igual

>

<

>=

<=

==

!=

- **Lógica:** e, ou, não

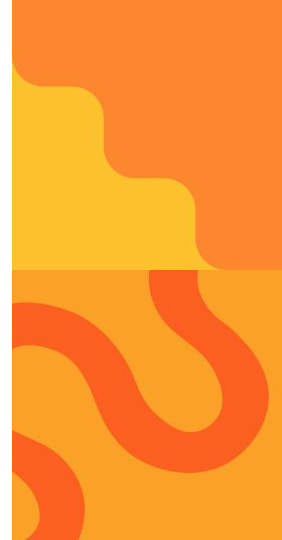
e

ou

nao



C.E.S.A.R.





## Operador de Atribuição

- Símbolo (recebe): =
- Sintaxe:
  - `[variável] = [valor]`
  - `[variável] = [expressão]`
- Primeiro avalia o lado direito do operador
- O valor de `[valor]` ou `[expressão]` é guardado na posição de memória endereçada por `[variável]`



```
29 programa
30 {
31     funcao inicio ()
32     {
33         cadeia nome, sobrenome
34         inteiro idade
35         real altura
36
37         nome = "Carol"
38         sobrenome = "Melo"
39         idade = 26
40         altura = 1.76
41
42     }
43 }
44
```

# Operadores Aritméticos

- Para construir algoritmos que realizam cálculos matemáticos, precisamos utilizar os operadores aritméticos;
- As expressões aritméticas devem ser linearizadas:

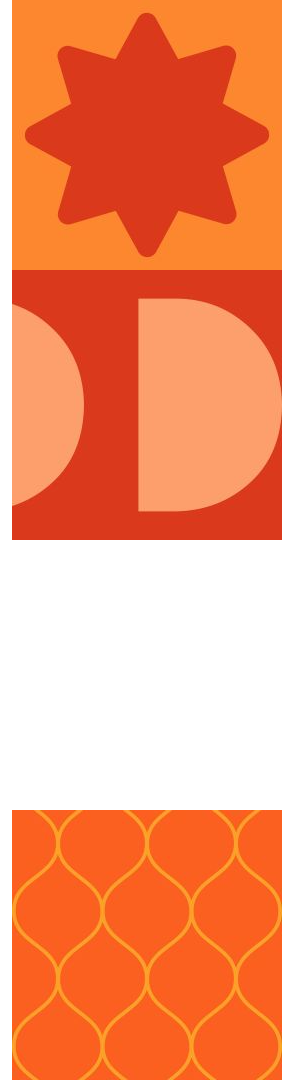
## Tradicional

$$\left\{ \left[ \frac{7}{3} - (7 + 2) \right] + 5 \right\} . 2$$

## Computacional

$$((7/3 - (7 + 2) + 5)) * 2$$

- Deve ser feito o mapeamento dos operadores utilizados tradicionalmente para o português estruturado, que é utilizado no Portugol Studio.



# Operadores Aritméticos

Operadores Aritméticos	Portugol Studio	Descrição
Adição	<b>+</b>	Operador tradicional de adição
Subtração	<b>-</b>	Operador tradicional de subtração
Multiplicação	<b>*</b>	Operador tradicional de multiplicação
Divisão	<b>/</b>	Operador tradicional de divisão
Módulo (Resto da Divisão)	<b>%</b>	Resto da divisão inteira ( $9\%2=1$ )
Incremento	<b>a++</b>	Acrescenta 1 ao valor da variável
Decremento	<b>a--</b>	Diminui 1 do valor da variável



# Operadores de Incremento

- Símbolos: `++` `--` `+=` `-=` `*=`

- Sintaxe:

- `[variável]++`
- `[variável] += [valor]`

- Servem para escrevermos algumas operações de forma mais simples:

```
a++
```

é o mesmo que

```
a = a + 1
```

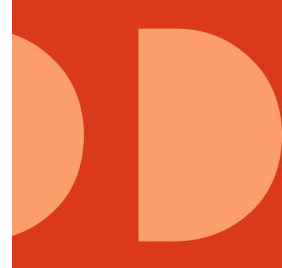
```
b += 4
```

é o mesmo que

```
b = b + 4
```



C.E.S.A.R



# Operadores Relacionais

Realizam comparações e retornam valores lógicos:  
**verdadeiro** ou **falso**;

Operadores Relacionais	Matemática	Portugol Studio
Maior	$>$	$>$
Menor	$<$	$<$
Maior ou igual	$\geq$	$>=$
Menor ou igual	$\leq$	$<=$
Igual	$=$	$==$
Diferente	$\neq$	$!=$

# Operadores Lógicos

Servem para combinar resultados de expressões retornando se o resultado final é VERDADEIRO ou FALSO.

Operadores Lógicos	Portugal Studio	Significado
Conjunção	<b>e</b>	O resultado será <b>verdadeiro</b> se uma parte <b>e</b> a outra parte forem verdadeiras
Disjunção	<b>ou</b>	O resultado será <b>verdadeiro</b> se uma parte <b>ou</b> a outra parte forem verdadeiras
Não Lógico	<b>nao</b>	O resultado será a inversão do valor lógico. Se for <b>verdadeiro</b> , torna-se <b>falso</b> .

# Operadores - Lógicos

- Vamos praticar? E no Portugol, como funciona?

programa

{

funcao inicio()

{

escreva(verdadeiro e verdadeiro, "\n")

escreva(verdadeiro e falso, "\n")

escreva(falso e falso, "\n")

}

}







# **Estruturas de Decisão**

**(se-senao)**

# Introdução

**Até agora, todos os algoritmos vistos eram sequenciais**

- Instruções são executadas uma após a outra
- De cima para baixo

**Porém, problemas reais, em sua maioria, exigem uma tomada de decisão no algoritmo, onde há comandos que desviam o fluxo de execução**

- Algumas instruções podem ser ignoradas
- Depende da condição dada



## Quando usar as instruções de decisão?

- Quando queremos que uma condição seja analisada;
- Dependem de uma condição;
- Resultado da condição deve retornar VERDADEIRO ou FALSO;
- Caso esta condição seja verdadeira, um comando será executado;
- Caso esta condição seja falsa, outro comando será executado.



# Instrução de Decisão Simples

- Utiliza a seguinte sintaxe:

```
se(condição) {  
    // código a ser executado  
}
```

- A expressão da *condição* é avaliada;
- Se o resultado da avaliação é **verdadeiro**
- Os comandos dentro do bloco **indentado** são executados



# Instrução de Decisão Simples

- Se o resultado da avaliação é FALSO, tudo dentro do escopo do "se" é ignorado
- Execução segue normal depois do "se"

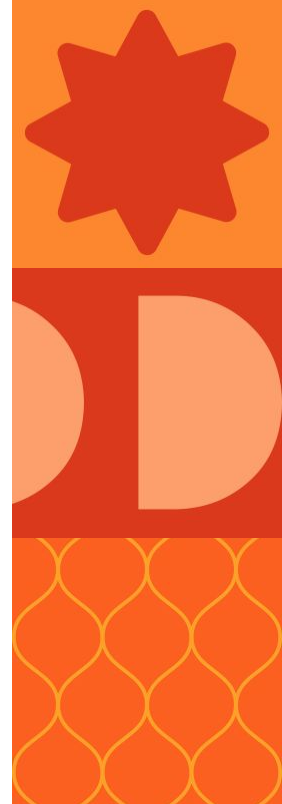
```
se (num > 10) {  
    escreva (num, " é maior que 10")  
}
```



# Instrução de Decisão Composta

Estrutura condicional “se-senao”:

- Após definir o parâmetro do “se” o que não ocorrer dentro desse parâmetro podemos colocar no “senao” e declarar novas instruções para esse parâmetro.



# Instrução de Decisão Composta

Sintaxe:

```
se(condição) {  
    // Código a ser executado  
    // se a condição for verdadeira  
} senao {  
    // Código a ser executado  
    // se a condição NÃO verdadeira  
}
```



# Exemplos

(Estruturas de Decisão)



## A 3x3 grid of nine squares, each containing a different geometric pattern or shape in various shades of orange and red. The patterns include a star, a circle, a diamond, a chevron, and a repeating scale pattern.

A 3x3 grid of nine squares, each containing a different geometric pattern or shape in various shades of orange and red. The patterns include a star, a circle, a diamond, a chevron, and a repeating scale pattern.

## A 3x3 grid of nine squares, each containing a different geometric pattern or shape in various shades of orange and red. The patterns include a star, a circle, a diamond, a chevron, and a repeating scale pattern.

A 3x3 grid of nine squares, each containing a different geometric pattern or shape in various shades of orange and red. The patterns include a star, a circle, a diamond, a chevron, and a wavy line.

- 
- A 3x3 grid of nine squares, each containing a different geometric pattern or shape in various shades of orange and red. The patterns include a star, a circle, a diamond, a chevron, and a wavy line.

# EXEMPLO 4

```
programa
{
    funcao inicio()
    {
        real n1, n2, total

        leia(n1)
        leia(n2)

        total = n1 + n2

        se(total >= 10){
            escreva(total)
        }
    }
}
```



# EXEMPLO 5

```
funcao inicio()
{
    inteiro x, y

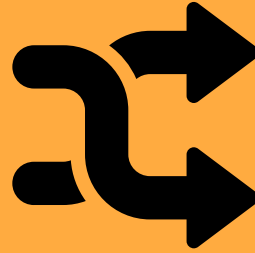
    leia(x)
    leia(y)

    se(x == y){
        escreva("São iguais")
    }

    se(x > y){
        escreva(x, " é maior que ", y)
    }

    se(x < y){
        escreva(x, " é menor que ", y)
    }
}
```





# **Estruturas de Decisão**

**(se-senao se)**

# Instrução de Decisão Aninhada

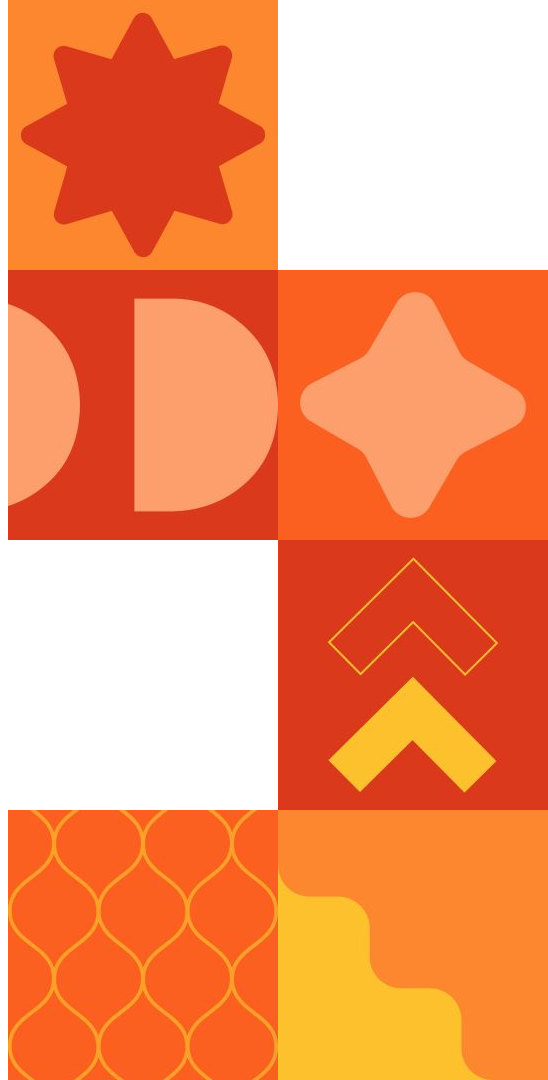
- Estrutura condicional “se – senao se”:
  - É utilizada para verificar se uma condição é verdadeira, caso contrário, verifica se a próxima condição é verdadeira e assim por diante.
- Uma determinada ação não poderá ser executada se uma condição anterior for satisfeita;
- É utilizada uma instrução de decisão dentro de uma instrução de decisão.



# Instrução de Decisão Aninhada

Sintaxe:

```
se(condição) {  
    // Código a ser executado  
} senao se(condição) {  
    // Código a ser executado  
} senao se(condição) {  
    // Código a ser executado  
} senao se(condição) {  
    // Código a ser executado  
} senao se(condição) {  
    // Código a ser executado  
} senao {  
    // Código a ser executado caso  
    // nenhum outro caso tenha sido  
    // atendido  
}
```



# EXEMPLO 6

Se o valor da mesada for maior ou igual a R\$20,00, vamos ao cinema. Menor que R\$ 20,00 e maior ou igual a R\$ 10,00, vamos comprar pipoca e Netflix. Menor que R\$10,00, vamos apenas assistir Netflix.



# EXEMPLO 6

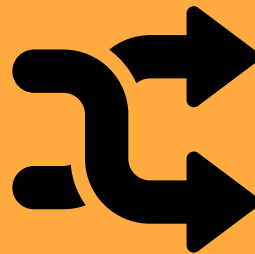
```
funcao inicio()
{
    real mesada

    escreva("Insira o valor da mesada: ")
    leia(mesada)

    se(mesada >= 20) {
        escreva("Cineminha \\o/")
    } senao se(mesada >= 10 e mesada < 20) {
        escreva("Pipoca e Netflix")
    } senao {
        escreva("Só Netflix, mesmo")
    }
}
```







# **Estruturas de Decisão**

**(escolha-caso)**

## Instrução de Decisão Composta

Em determinadas situações, temos um número predeterminado de opções de entrada. Para executar um código específico para cada um dos casos, é possível usar a estrutura de decisão se-senão. Contudo, se forem muitos casos, o código pode ficar extenso e complexo.

Para esses casos, podemos usar o comando escolha-caso.

- Com ele não é possível usar os operadores lógicos. Internamente o operador de igualdade é utilizado.
- Esta estrutura requer o uso do comando **pare** ao final de cada instrução, caso contrário o comando executará todos os comandos existentes.



# Instrução de Decisão Composta

Sintaxe:

**!** A estrutura escolha é compatível apenas com os tipos **inteiro** e **caracter**!

```
escolha(variável) {  
    caso valor1:  
        // Código a ser executado  
    pare  
    caso valor2:  
        // Código a ser executado  
    pare  
    caso valor3:  
        // Código a ser executado  
    pare  
    caso contrario:  
        // Código a ser executado  
}
```



# EXEMPLO 7

O programa deve receber a entrada de um número e verificar se é igual ou não aos números 0, 1 e 2. Caso seja diferente, imprime: "O valor não é igual a 0, 1 e 2".

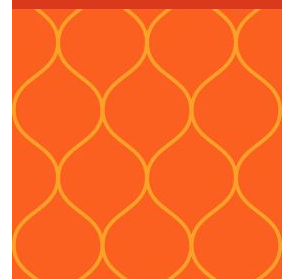


# EXEMPLO 7

```
funcao inicio()
{
    inteiro num
    leia(num)

    escolha(num){

        caso 0: escreva("O valor é igual a 0")
        pare
        caso 1: escreva("O valor é igual a 1")
        pare
        caso 2: escreva("O valor é igual a 2")
        pare
        caso contrario: escreva("O valor não é igual a 0, 1 e 2")
        pare
    }
}
```



# EXEMPLO 8

Escreva um programa que receba um operador de soma ou subtração, e dois números inteiros. Mostre o resultado da operação em seguida.

Exemplo de entrada	Exemplo de saída
+ 3 2	5
- 8 10	-2

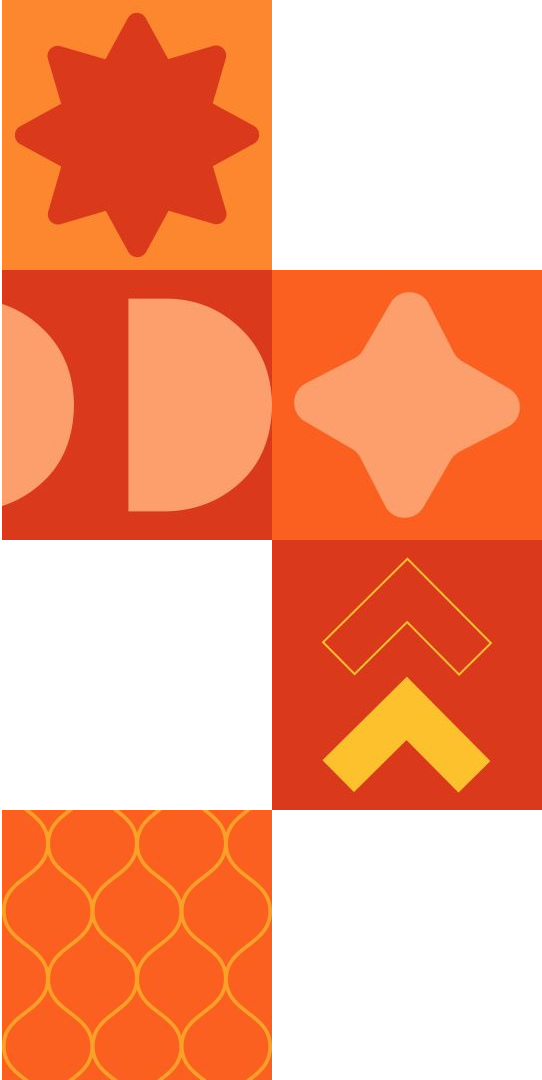


# EXEMPLO 8

```
funcao inicio()
{
    cadeia operador
    inteiro num1, num2

    leia(operador)
    leia(num1, num2)

    escolha(operador) {
        caso '+':
            escreva(num1 + num2)
            pare
        caso '-':
            escreva(num1 - num2)
            pare
    }
}
```



# Breakout Time!

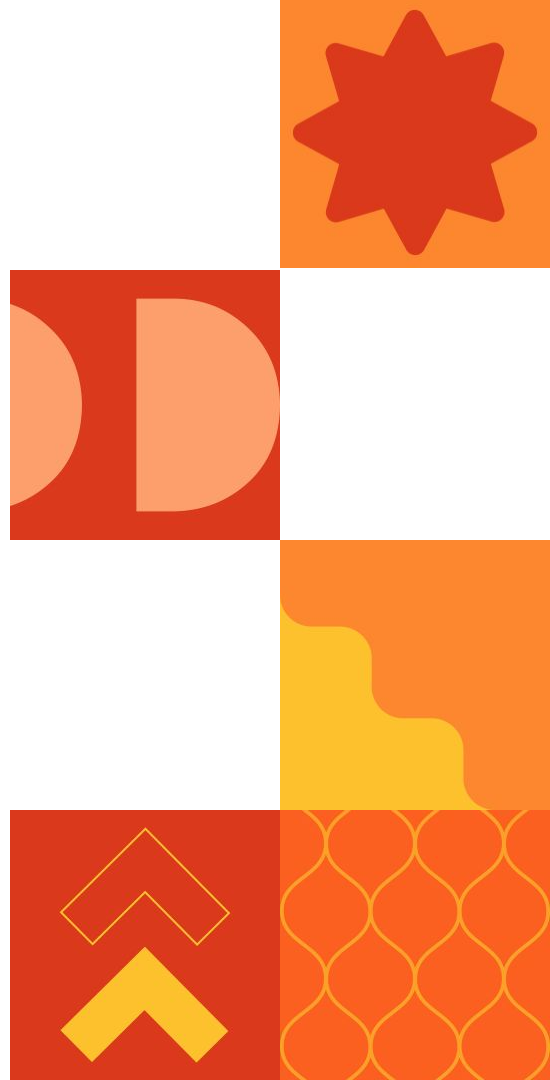
Resolva os desafios da lista de exercícios com sua sala no breakout room.



A lista possui exercícios em duas categorias:

- Exercícios fundamentais;
- Exercícios de aprofundamento.

Se precisar de ajuda, chame uma das pessoas monitoras ou professoras.





Pessoas impulsionando inovação.  
Inovação impulsionando negócios.



**Taty Calixto**

e a melhor equipe de monitores da CESAR School 

Copyright © CESAR School 2023 | Todos os direitos reservados.

Este material, ou qualquer parte dele, não pode ser reproduzido, divulgado ou usado de forma alguma sem autorização escrita.