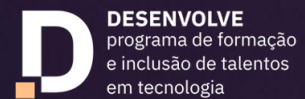



KORU





VARIÁVEIS, OPERADORES, CONDICIONAIS E LOOPS

O QUE VAMOS APRENDER HOJE?

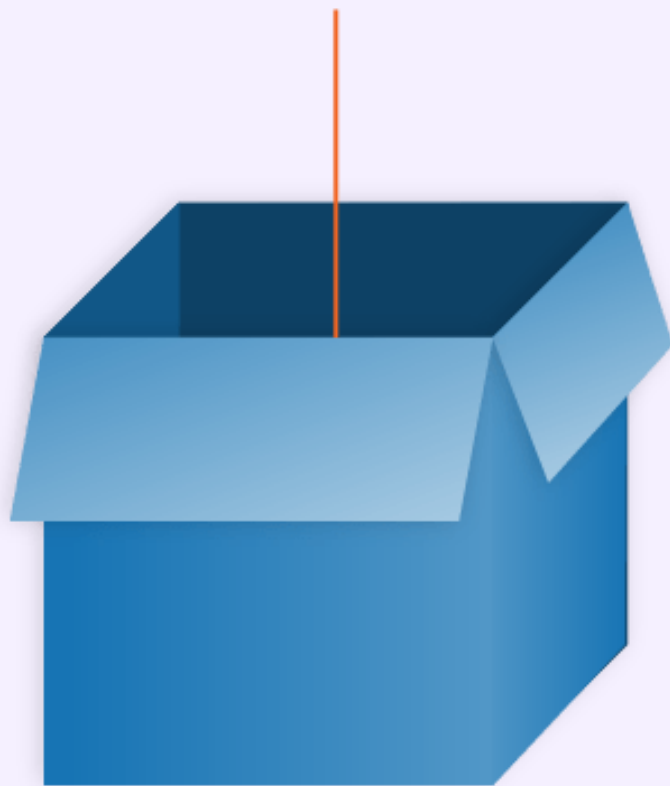
- Lógica de programação!
- Variáveis e Tipos de Dados (**guardando informação**)
- Operadores (**cálculos**)
- Condicionais (**tomada de decisão**)
- Loops (**repetindo tarefas**)



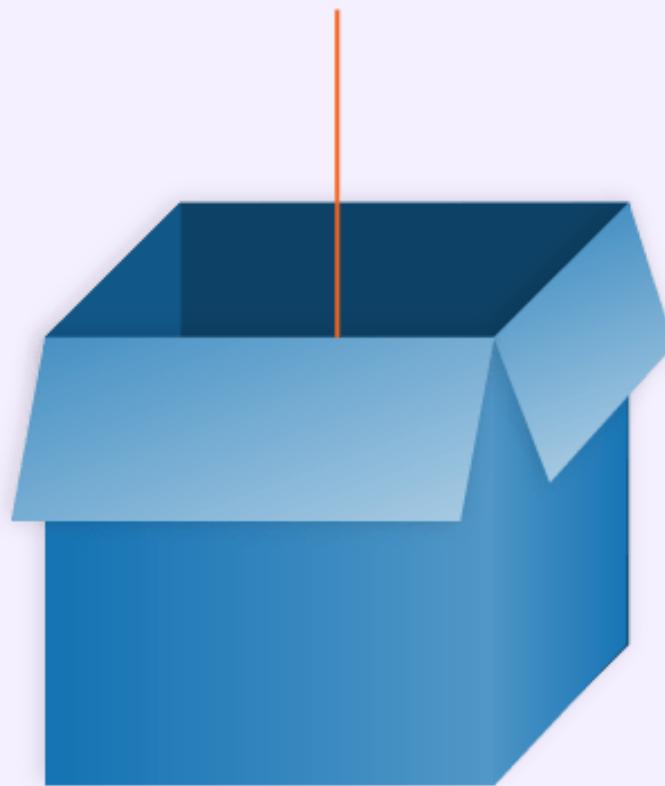
VARIÁVEIS

VARIÁVEIS SÃO COMO "CAIXAS" ROTULADAS

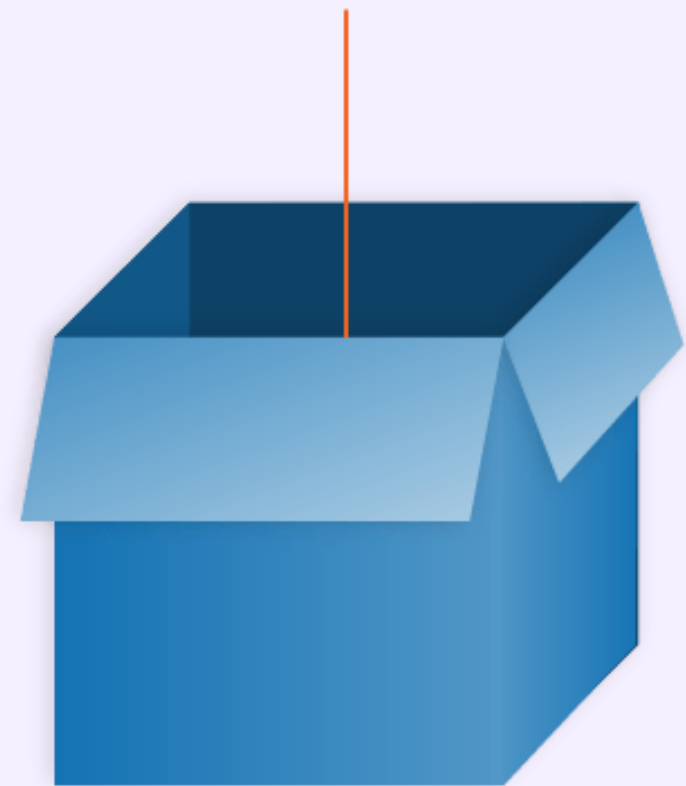
"Bob"



true



35



VARIÁVEIS SÃO COMO "CAIXAS" ROTULADAS

```
// Criando uma caixa que guardará o valor da idade de alguém
```

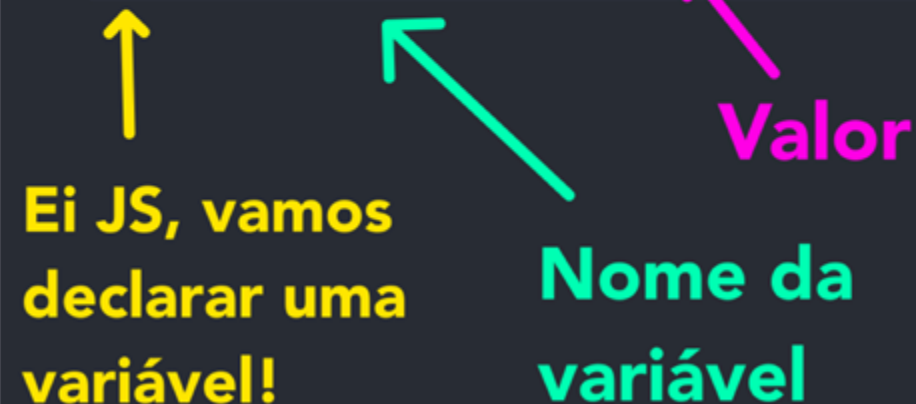
```
let age = 30
```

```
// Criando uma caixa que guardará o nome de alguém
```

```
let name = 'Benjamin'
```

```
// Estrutura de variáveis
```

```
let address = 'Rua X'
```



Ei JS, vamos declarar uma variável!

Nome da variável

Valor

VARIÁVEIS SÃO COMO "CAIXAS" ROTULADAS

Se você rodar esse código, o que vai acontecer?

```
// Criando uma caixa que guardará o valor da idade de alguém
let age = 30

// Criando uma caixa que guardará o nome de alguém
let name = 'Benjamin'

// Estrutura de variáveis
let address = 'Rua X'
```

The diagram illustrates the components of a variable declaration in JavaScript. It features three colored arrows pointing to parts of the line `let address = 'Rua X'`:

- A yellow arrow points to the keyword `let`, with the text "Ei JS, vamos declarar uma variável!" (Hey JS, let's declare a variable!) below it.
- A green arrow points to the variable name `address`, with the text "Nome da variável" (Variable name) below it.
- A pink arrow points to the value `'Rua X'`, with the text "Valor" (Value) next to it.

VARIÁVEIS SÃO COMO "CAIXAS" ROTULADAS

E agora?

```
// Criando uma caixa que guardará o valor da idade de alguém
let age = 30

// Criando uma caixa que guardará o nome de alguém
let name = 'Benjamin'

// Estrutura de variáveis
let address = 'Rua X'

console.log(age);
console.log(name);
console.log(address)
```


DECLARANDO VARIÁVEIS: LET VS CONST

Se você rodar esses códigos, o que vai acontecer?

```
let age = 30  
age = 10  
console.log(age)
```

```
let age = 30  
console.log(age)  
age = 10
```

DECLARANDO VARIÁVEIS: LET VS CONST

E agora?

```
const age = 30  
  
age = 10  
  
console.log(age)
```

DECLARANDO COM `LET` E COM `CONST`

- **let**: para variáveis cujo valor pode mudar ao longo do tempo
- **const**: para variáveis cujo valor não pode mudar após a primeira atribuição.

⚠ Para arrays e objetos (vamos ver essas estruturas mais tarde) o conteúdo pode ser alterado mesmo usando const

E **var**? Não se preocupe - e não use!

DICA - NOME DE VARIÁVEIS

- Escolha **nomes descritivos** que expliquem o que a variável guarda. Vamos usar **nomes em inglês** no curso.
- Use a convenção **camelCase** em Javascript: *myVariable, userAge*.
- **Regras:** Podem conter letras, números, \$, _.
 - Não podem começar com número.
 - Não podem ser palavras reservadas (if, for, etc.).

CURIOSIDADE - TIPOS DE "CASES"

Nome do Padrão	Descrição	Exemplo(s)	Usos Comuns
camelCase	Primeira palavra minúscula, restantes iniciam com maiúscula.	minhaVariavel	JS, Java, Swift (vars, funções)
PascalCase	Cada palavra começa com maiúscula.	MinhaClasse	C#, Java, Swift (classes, métodos, propriedades)
snake_case	Todas minúsculas, separadas por <code>_</code> .	minha_variavel	Python, Ruby (vars, funções), Bancos de Dados
kebab-case	Todas minúsculas, separadas por <code>-</code> .	meu-componente	CSS, HTML, URLs, nomes de arquivos
SCREAMING_SNAKE_CASE	Todas maiúsculas, separadas por <code>_</code> .	VALOR_MAXIMO	Constantes
flatcase	Todas minúsculas, sem separadores.	minhavariavel	Pouco frequente para nomes com várias palavras



TIPOS DE DADOS (PRIMITIVOS)

STRING

String é a mesma coisa que **texto**.

```
const name = "Fulano de Tal";  
const message = "Olá mundo!"  
const anotherMessage = `Olá ${name}`
```

NUMBER

Number pode ser números inteiros ou decimais

```
// Números podem ser inteiros (integer)  
// ou decimais (floats / double)  
  
const age = 33;  
  
const price = 99.99
```

BOOLEAN

Boolean → verdadeiro/falso.
Usado para condições

```
// Booleans representam V/F  
let isActive = true;  
let isAdmin = false;
```

NULL E UNDEFINED

Null → representa a ausência *intencional* de qualquer valor

Undefined → alguma variável foi declarada, mas não recebeu um valor.

```
// aqui sabemos (e queremos) que o user
// seja nulo.
let user = null;

// aqui address será undefined.
let address;
```



TIPOS DE DADOS (OBJECT)

OBJETOS

Objetos são usados para representar entidades mais complexas com propriedades (pares chave: valor).

Analogia: *ficha de cadastro*

LOGO	FICHA DE CADASTRO DE EMPRESA		
DADOS CADASTRAIS			
Razão Social:			
Nome Fantasia:		CNPJ:	
Inscrição Estadual:		Inscrição Municipal:	
Atividade Principal:			
Endereço:			
CEP:		Cidade/Estado:	
Telefone:		Fax:	
E-mail:			
COMPOSIÇÃO CAPITAL			
Nome dos Sócios	Cargo	% Participação	Telefone
REFERÊNCIAS COMERCIAIS			
Fornecedores / Endereço		Telefone	E-mail
REFERÊNCIAS BANCÁRIAS			
Banco	Agência	Conta	Telefone
CONTATOS NA EMPRESA			
Financeiro:		Telefone:	
Recursos Humanos:		Telefone:	
Administração:		Telefone:	
Vendas:		Telefone:	

ARRAYS

Arrays são um tipo especial de Objeto em Javascript.

Usados para guardar uma **coleção de valores ordenados**.

Cada valor tem um índice numérico, começando do 0.

```
let frutas = ['Maçã', 'Banana', 'Laranja'];

// Acessando elementos pelo índice:
console.log(frutas[0]); // Saída: "Maçã"
console.log(frutas[1]); // Saída: "Banana"

// Um array pode conter valores de tipos diferentes:
let misturado = [1, 'texto', true, { nome: 'Obj' }];
```

OBJETOS

```
let carro = {  
  marca: 'Toyota',  
  modelo: 'Corolla',  
  ano: 2022,  
  cor: 'prata',  
};  
  
// Acessando as propriedades do objeto:  
console.log(carro.marca); // Saída: Toyota  
console.log(carro.ano); // Saída: 2022
```



OPERADORES

OPERADORES ARITMÉTICOS

- Utilizamos **operadores** para trabalhar com valores de variáveis
- **Aritméticos**
 - Adição (+)
 - Subtração (-)
 - Multiplicação (*)
 - Divisão (/)
 - Módulo / Resto (%)
 - Potência (**)

```
let a = 10;  
let b = 5;  
let soma = a + b; // 15  
let resto = a % b; // 0  
let potencia = b ** 2; // 25
```

OPERADORES DE ATRIBUIÇÃO

Usados para **atribuir ou atualizar** o valor de uma variável.

- Atribuição simples (=)
- Somar e atribuir (+=)
- Subtrair e atribuir (-=)
- Também há multiplicar/dividir e atribuir.

```
let contador = 10;  
contador += 5; // contador agora é 15  
contador *= 2; // contador agora é 30
```

OPERADORES DE COMPARAÇÃO

Para comparar valores. O **resultado é sempre um Boolean** (true ou false).

```
console.log(5 == "5"); // Saída: true (Perigo!)
console.log(5 === "5"); // Saída: false (Correto!)
console.log(10 > 5); // Saída: true
console.log(10 <= 10); // Saída: true
console.log("a" < "b"); // Saída: true (Comparação de strings alfabética)
```

- **==** Igual a (Compara **valor**, ignora tipo - EVITAR!)
- **===** Estritamente Igual a (Compara **valor** E **tipo** - PREFIRA!)
- **!=** Diferente de (Compara **valor**, ignora tipo - EVITAR!)
- **!==** Estritamente Diferente de (Compara **valor** E **tipo** - PREFIRA!)
- **>** Maior que
- **<** Menor que
- **>=** Maior ou igual a
- **<=** Menor ou igual a

OPERADORES LÓGICOS

Para **combinar** ou **negar** **condições** booleanas.

- **&&** (AND / E): Verdadeiro se ambas as condições forem verdadeiras.

- **||** (OR / OU): Verdadeiro se pelo menos uma condição for verdadeira.

- **!** (NOT / NÃO): Inverte o valor booleano.

```
let isLegalAge = age >= 18;
let hasCNH = true;
console.log(isLegalAge && hasCNH); // true se for maior E tiver CNH

let isWeekend = weekDay === "Sábado" || weekDay === "Domingo"; // true se for Sábado OU Domingo
```

OPERADOR TERNÁRIO

Uma forma curta de **if/else** para atribuição ou retorno simples.

Sintaxe:

- condicao ?
valor_se_verdadeiro :
valor_se_falso;

```
let age = 20;  
let status = age >= 18 ? "Adulto" : "Menor";  
console.log(status); // Saída: "Adulto"  
  
let isMember = false;  
let price = isMember ? 100 : 150; // Preço é 100 se for membro, 150 caso  
contrário  
console.log(price); // Saída: 150
```

TRUTHY E FALSY

Em contextos booleanos (como **if**, **&&**, **||**), valores não-booleanos são interpretados (coercidos) como true ou false.

Falsy Values (interpretados como false):


- false
- 0 (o número zero)
- "" (string vazia)
- null
- undefined
- NaN (Not a Number)

Truthy Values (interpretados como true):

- Qualquer outra coisa!
(Strings com conteúdo, números diferentes de zero, objetos, arrays, etc.)

TRUTHY E FALSY

```
if (0) {  
  console.log("Entrou aqui?"); // Não, 0 é Falsy  
}  
  
if ("texto") {  
  console.log("Entrou aqui??"); // Sim, string não vazia é Truthy  
}  
  
let myVariable;  
if (myVariable) {  
  console.log("Entrou aqui??"); // Não, undefined é Falsy  
}
```



ESTRUTURAS CONDICIONAIS (IF)

ESTRUTURAS CONDICIONAIS: TOMANDO DECISÕES

Permitem que o código execute blocos diferentes com base em uma condição ser true ou false.

- **if**: Executa o bloco de código SE a condição for verdadeira.
- **if...else**: Executa um bloco SE a condição for verdadeira, e outro bloco CASO CONTRÁRIO.

```
let hours = 10;  
if (hours < 12) {  
    console.log("Bom dia!");  
}
```

```
let hora = 14;  
if (hora < 12) {  
    console.log("Bom dia!");  
} else {  
    console.log("Boa tarde!");  
}
```


ESTRUTURAS CONDICIONAIS: TOMANDO DECISÕES

- **else if:** Para verificar múltiplas condições em sequência.

```
let grade = 75;

if (grade >= 90) {
  console.log('Aprovado com Excelência!');
} else if (grade >= 70) {
  // Executa se a primeira for falsa E esta for verdadeira
  console.log('Aprovado!');
} else if (grade >= 50) {
  // Executa se as anteriores forem falsas E esta for verdadeira
  console.log('Recuperação.');
} else {
  // Executa se NENHUMA das condições acima for verdadeira
  console.log('Reprovado.');
}
```



ESTRUTURAS DE REPETIÇÃO (LOOPS)

ESTRUTURAS DE REPETIÇÃO / LOOPS

- Permitem executar um bloco de código várias vezes. Essencial para tarefas repetitivas
 - Existem diversas formas de executar loops no JavaScript.
 - As formas mais básicas são:
 - **for** loops
 - **while** loops e **do...while** loops
- ⚠ **Não vamos ver while loops pois são pouco utilizados.**

FOR LOOPS

- Você deverá saber (ou poder determinar) quantas vezes quer repetir

(let i = 0; Inicialização - executa 1 vez)

(i < 5; Condição - verifica antes de CADA repetição)

(i++; Incremento - executa DEPOIS de CADA repetição)

The diagram shows a JavaScript for loop with three annotations above it: 'Variável "i"' (Variable "i") in pink, 'condição' (condition) in orange, and 'executa após repetição' (executes after repetition) in yellow. Arrows point from each annotation to its corresponding part of the loop syntax. The code is:

```
for (let i = 0; i < 5; i++) {  
  console.log("Número: " + i);  
}
```

 The parts of the code are underlined and color-coded: 'let i = 0' is pink, 'i < 5' is orange, and 'i++' is yellow.

```
for (let i = 0; i < 5; i++) {  
  console.log("Número: " + i);  
}
```

BREAK E CONTINUE

break e continue são controles do loop e modificam o fluxo normal dentro dele.

- **break:** Sai imediatamente do loop atual.
- **continue:** Pula a execução do resto do código nesta iteração e vai para a próxima.

```
for (let i = 0; i < 10; i++) {  
  if (i === 5) {  
    break; // Quando i for 5, sai do loop  
  }  
  console.log(i); // Imprime de 0 a 4  
}
```

```
for (let i = 0; i < 5; i++) {  
  if (i === 2) {  
    continue; // Quando i for 2, pula para a próxima  
               iteração (i=3)  
  }  
  console.log(i); // Imprime 0, 1, 3, 4  
}
```


A group of diverse young people, including men and women of various ethnicities, are smiling and waving. The image is overlaid with a purple gradient. The word "RESUMO" is written in large, white, bold, sans-serif capital letters on the right side of the image.

RESUMO

RESUMO DO DIA

- **Variáveis** (let, const) armazenam valores de diferentes Tipos (String, Number, Boolean, Object, Array...).
- **Operadores** realizam cálculos (+, -), comparações (===, >), e combinam lógicas (&&, ||).
- **Condicionais** (if/else, switch) controlam o fluxo do código com base em condições (true/false).
- **Loops** (for, while, do...while) repetem a execução de blocos de código.
- **Truthy/Falsy** são comportamentos específicos do JS em contextos booleanos/de operação.



Vamos
avaliar o
encontro?

KORU

