



Introdução Javascript



Aula Passada

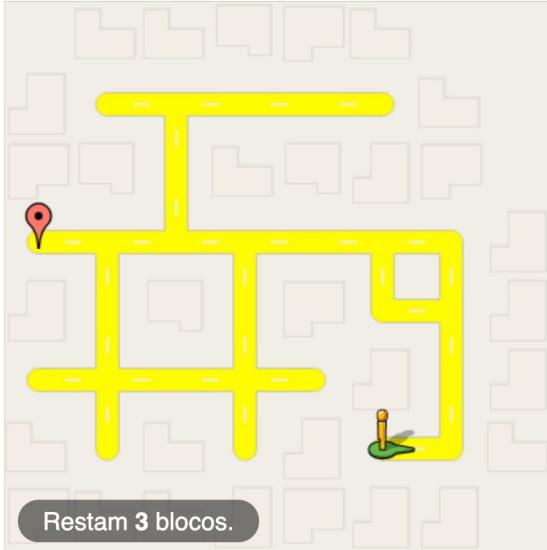
- **Pensamento Computacional:**
 - Decomposição
 - Reconhecimento de padrões
 - Abstração
 - Algoritmos
 - i. Descrição narrativa
 - ii. Fluxograma
 - iii. Pseudocódigo
 - iv. Portugol



Aula Passada - Exercícios

Blockly Games : Maze

Jogos do Blockly : Labirinto



MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INovação

GOVERNO FEDERAL
BRASIL
UNIÃO E RECONSTRUÇÃO

SOFTEX
PERNAMBUCO

 **Softex**



Aula Passada - Exercícios

- Portugol:

- Média escolar

```
1  programa
2  {
3    funcao inicio ()
4    {
5      real nota1, nota2, media
6
7      escreva ("Informe a nota 1: ")
8      leia (nota1)
9      escreva( "Informe a nota 2: ")
10     leia (nota2)
11
12     media = (nota1+nota2) / 2
13
14     se (media >= 7)
15     {
16       escreva ("Aprovado\n")
17     }
18     senao [
19       se(media >= 5){
20         escreva ("Recuperação\n")
21       }
22       senao{
23         escreva ("Reprovado\n")
24       }
25     ]
26   }
```

- Portugol:

- IMC

```
1  programa
2  {
3    funcao inicio ()
4    {
5      real massa, altura, imc
6
7      escreva ("Informe sua massa: ")
8      leia (massa)
9      escreva( "Informe sua altura: ")
10     leia (altura)
11
12     imc = massa / (altura * altura)
13     escreva("Seu IMC é: ", imc)
14     escreva("\n")
15
16     se (imc > 25)
17     {
18       escreva ("IMC ACIMA DO NORMAL\n")
19     }
20     senao [
21       escreva("IMC NORMAL")
22     ]
23   }
```



// Fundamentos de JS



JavaScript é uma **linguagem interpretada** e pode ser executada por navegadores da web, bem como por um ambiente de execução como o Node.

Cada navegador tem sua própria **engine** Javascript

- V8: Google Chrome, Microsoft Edge
- SpiderMonkey: Firefox
- JavaScriptCore: Safari

O ambiente de execução Node.js tem a mesma engine do chrome (V8)

Cada um deles segue o padrão **ECMAScript**, mas podem diferir em qualquer coisa não definida pelo padrão

// Fundamentos de JS

1. ECMAScript

É só a **especificação da linguagem**.

- Um “manual” oficial que descreve como a linguagem deve se comportar: sintaxe, tipos, operadores, funções básicas ([Array](#), [Object](#), etc.).
- Mas ele **não é código**, é apenas teoria/norma.



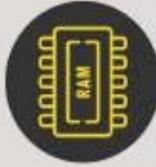
2. JavaScript

É a **implementação mais famosa** do ECMAScript.



- Criado pela Netscape nos anos 90.
- Seguiu (e ainda segue) o padrão ECMAScript.
- Além de ECMAScript, o termo “JavaScript” também virou “marca” para o conjunto da linguagem + APIs que existem nos navegadores.

[Variáveis em JavaScript]



Variáveis são um espaço na memória onde qualquer tipo de dado pode ser salvo e então realizar operações com essas informações em outro lugar do código.



Poderíamos imaginar as variáveis como uma caixa onde os objetos podem ser armazenados. Nesse caso, o nome da variável seria o rótulo daquela caixa, que servirá de referência para identificá-la.



// Tipos de variáveis

Em JavaScript existem três tipos de variáveis:

- `var`
- `let`
- `const`

Para declarar uma variável escrevemos o tipo e o nome que queremos dar à variável:

```
var nome;  
{}  
let contador;  
const url;
```

Vejamos cada parte com mais detalhe...



// Declaração de uma variável

```
let nomeSignificativo;
```

let

A palavra reservada **let** informa ao JavaScript que vamos a **declarar uma variável do tipo let**.

Nome

- Só pode ser composto por letras, números e os símbolos \$ (pesos) e _ (sublinhado)
- Não podem começar com um número
- Não deveriam possuir ñ ou caracteres com acentos



É uma boa prática que os nomes das variáveis usem o formato **Camel case**. *Exemplo* em vez de variávelexemplo ou variavel_exemplo.

**“As boas práticas, embora não sejam
obrigatórias para que o nosso código
funcione, permitirão que este seja mais
fácil de ler e de manter.”**



MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INovaÇÃO



// Declaração de uma variável

```
let miVariable;
```

Não é o mesmo que:

```
let MiVariable;
```



JavaScript é uma linguagem que faz diferença entre MAIÚSCULAS e minúsculas.
Por isso, é bom seguir o padrão na hora de escrever os nomes.

// Asignação de um valor

```
let apelido= 'Hackerman';
```



Nome Asignação

O nome que nos servirá para identificar a nossa variável quando precisamos usá-la.

Diz JavaScript o que queremos salvar? o valor à direita na variável esquerda.

Valor

O que vamos salvar na nossa variável. Neste caso, um texto



// Asignação de um valor

A primeira vez que declaramos uma variável é necessário a palavra reservada **let**.

```
{ } let meuapelido = 'Hackerman';
```

Uma vez declarada a variável, atribuímos-lhe valores sem let.

```
{ } meuapelido = 'El Barto';
```



// Declaração com var

Essas variáveis são declaradas de maneira semelhante, com a diferença de que usamos a **palavra reservada var**.

```
{ } var contador = 1;
```

A principal diferença entre **var** e **let** é que **var** estará acessível global em todo o nosso código e o acesso não será limitado apenas ao bloco do código onde foi declarado como é o caso de **let**. Por convenção e boas práticas recomenda-se o uso de **let**. A palavra reservada **var** está obsoleta. Os blocos de código geralmente são determinados pelas chaves {}.



Vejamos um exemplo:

```
if (true) {  
  var nome = "Juan";  
}  
  
console.log(nome);  
//Ok, mostra 'Juan'"
```

```
if (true) {  
  let nome = "Juan";  
}  
  
console.log(nome);  
// "Erro: nome não existe"
```



Quando usamos **var**, JavaScript ignora os blocos de código e torna nossa variável global.

Isso significa que se houver outra variável de **nome** em nosso código, certamente estaremos pisando em seu valor.

Quando usamos **let**, JavaScript respeita os blocos de código. Isso significa que o nome não pode ser acessado fora do if.

Isso também significa que podemos ter variáveis com o mesmo nome em diferentes blocos do nosso código.

Javascript - var e let

var:

1



```
var apple
```

Podemos declarar uma variável
sem **inicializar**

2



```
apple = maçã vermelha
```

Para atribuir um valor fazemos isso
com o **operador de atribuição** "="

3



```
apple = maçã verde
```

Podemos acessar o valor e modificá-lo
de qualquer lugar do código.

let:

1



```
let apple
```

Podemos declarar uma variável
sem **inicializar**

2



```
apple = maçã vermelha
```

Podemos acessar o valor e modificá-lo
somente dentro do bloco em que foi
declarado.

3



```
apple = maçã verde
```



// Declaração com const

As variáveis **const** são declaradas com a palavra reservada **const**.

```
{} const email = 'meu.email@gmail.com';
```

As variáveis declaradas com **const** funcionam da mesma forma que variáveis **let**,
Eles estarão disponíveis apenas no bloco de código em que foram declarados.
Ao contrário de **let**, uma vez que atribuímos um valor a eles, não podemos
mudar isso.

```
{  
  email = 'meu.otro.email@gmail.com';  
  // Error de asignación, no se puede cambiar el valor  
  // de un const
```



Javascript - const

› const:

1



~~const~~ apple



const apple = 'maçã vermelha'

Não podemos declarar uma variável sem **inicializar**

2



apple = 'maçã verde'



apple.changeValue('maçã verde')

Não podemos modificar seu valor de referência

mas poderíamos pedir que você altere seu valor para si mesmo se você tiver um método para fazer isso



// Tipos de dados em JS

Tipos de dados primitivos:

- **String**: sequências de texto.
- **Number**: valores numéricos.
- **Boolean**: representa uma entidade lógica e pode ter dois valores: *true* e *false*.
- **null**: é um valor atribuído que tem o valor “sem valor”.
- **undefined**: uma variável à qual não foi atribuído um valor tem o valor *undefined*.

Tipos de dados especiais:

- **Array**: lista de dados que podem ser de qualquer tipo, incluindo outros arrays.
- **Object**: coleção de dados armazenados em formato de par chave/valor.



// Memes?



undefined



// Sintaxe dos diferentes tipos de dados

html

css

js

```
> // string  
let str = "Olá mundo"  
//number  
let num = 10  
// boolean  
let isEmpty = false  
// array  
let arr = [str, num, isEmpty]  
// object  
let obj = {string: str, number: num, boolean:  
isEmpty}
```



MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INovação



// Condicionais

Como todas as linguagens de programação, o JavaScript nos dá a possibilidade de adicionar estruturas de controle para que um ou outro bloco de código seja executado de acordo com uma condição.



```
if(idade > 17){  
    console.log('Você é maior de idade')  
} else {  
    console.log('Você é menor de idade')  
}
```

Estrutura condicional

// Condisional simples

Versão mais básica do `if`. Define uma condição e um bloco de código para executar se for verdadeiro.

```
{} if (condição) {  
    // código para executar se a condição é verdadeira  
}
```



// Condicional com bloco else

Igual ao exemplo anterior, mas adiciona um bloco de código para executar caso a condição seja falsa.

É importante observar que o bloco `else` é opcional.

```
if (condição) {  
    // código para executar se a condição é verdadeira  
}  
} else {  
    // código para executar se a condição é falsa  
}
```



// Condicional com bloco else if

Igual ao exemplo anterior mas adiciona um if adicional, ou seja, outra condição que pode ser avaliada caso a primeira seja falsa. Podemos adicionar a quantidade que quisermos de **else if**, mas idéia é que apenas um possa ser avaliado como **verdadeiro**.



```
if (condição) {  
    // Código para executar se a condição é verdadeira  
} else if (outraCondição){  
    // Código para executar se a outra condição é verdadeira  
} else {  
    // Código para executar se todas as condições forem falsas  
}
```

// Operadores Lógicos



Operadores lógicos e relacionais	Descrição	Exemplo
<code>==</code>	É igual	<code>a == b</code>
<code>==</code>	É estritamente igual	<code>a === b</code>
<code>!=</code>	É distinto	<code>a != b</code>
<code>!==</code>	É estritamente distinto	<code>a !== b</code>
<code><, <=, >, >=</code>	Menor, menor ou igual, maior, maior ou igual	<code>a <= b</code>
<code>&&</code>	Operador and (y)	<code>a && b</code>
<code> </code>	Operador or (o)	<code>a b</code>
<code>!</code>	Operador not (no)	<code>!a</code>

// Operadores de Igualdade == e ===



```
1 23 == "23"; //true, "23" será convertido para número
2 1 == true; //true, o valor booleano será convertido em seu equivalente numérico '1'
3 NaN == NaN; //false, '==' não considera NaN igual a si mesmo
```



```
1 23 === "23";      // false
2 1 === true;       // false
3 null === null;    // true
4 false === false;   // true
```



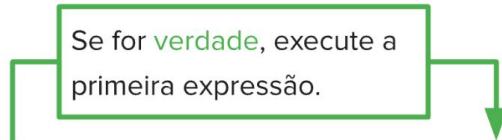
// Operadores Aritméticos



Símbolo	Operação	Exemplo	Descrição
+	Adição	$a + b$	Soma os dois operandos
-	Subtração	$a - b$	Subtrai o segundo operando do primeiro
*	Multiplicação	$a * b$	Multiplica os dois operandos
/	Divisão	a / b	Divide o primeiro operando pelo segundo
**	Potência	$a ** b$	Eleva o primeiro operando à potência do segundo operando
%	Módulo	$a \% b$	Divide o primeiro operando pelo segundo operando e produz a parte restante

// if ternário

Ao contrário de um **if tradicional**, o if ternário é escrito **horizontalmente**. Assim como o if tradicional, tem o mesmo fluxo (se essa condição for verdadeira, faça isso, se não, faça isso), mas neste caso **não é necessário** escrever a palavra **if** ou a palavra **else**.



condição ? primeira expressão : segunda expressão



// if ternário

Para o if ternário se é **obrigatório** colocar código na **segunda expressão**. Se não quisermos que nada aconteça, podemos usar uma string vazia".



```
{ } 4 > 10 ?    4 es mayor ' :      '0 10 es mayor';
```

Condição

Declaramos uma expressão que é avaliada como true ou false.

Primeira expressão

Se a condição for verdadeira, o código após o ponto de interrogação será executado.

Segunda expressão

Se a condição for falsa, o código após os dois pontos será executado. É obrigatório escrevê-lo.

Exercícios



Atividade 4 - Javascript



MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INovaÇÃO





SOFTEX
PERNAMBUCO

 **Softex**

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INovação

GOVERNO FEDERAL

UNIÃO E RECONSTRUÇÃO