

# CURSO DE FRONT-END

Aprendendo Front-end desenvolvendo jogos para web



**Professor/autor:**  
Leonardo S. Souza





# MÓDULOS 3 – JAVASCRIPT

A yellow square containing the letters 'JS' in a bold, black, sans-serif font, representing the JavaScript logo.

**JS**



# SUMÁRIO

1- O QUE É O JAVASCRIPT

2- FORMAS DE UTILIZAR O JAVASCRIPT

3- ATRIBUTO DEFER

5- O QUE É UMA VARIÁVEL

4- COMENTÁRIOS

6- SINTAXE DAS VARIÁVEIS NO JAVASCRIPT

7- DIFERENÇA ENTRE VAR E LET

8- REGRAS PARA NOMEAR UMA VARIÁVEL

9- CONSTANTE EM JAVASCRIPT

10- TIPOS DE VARIÁVEIS

11- OPERADORES

12- LAÇOS CONDICIONAIS

13- LAÇOS DE REPETIÇÃO

14- FUNÇÃO

15- ARRAY

16- MATRIZ

17- MANIPULAÇÃO DO DOM COM JAVASCRIPT

18- FUNÇÃO

19- DIFERENÇA ENTRE VARIÁVEIS GLOBAIS E LOCAIS

20- MANIPULAÇÃO DO DOM

21- TIPOS DE EVENTOS

22- ARRAY

23- FUNÇÕES DE UM ARRAY

24- MATRIZ

# O QUE É O JAVASCRIPT?

O JavaScript é uma **linguagem de programação responsável pela interatividade de uma página web**. Junto com as linguagens HTML e CSS, o JavaScript é uma das três principais tecnologias da World Wide Web. Ele **possui uma tipagem dinâmica**, ou seja, **não exige a declaração** do tipo de dado; o tipo da variável é determinado em tempo de execução. A tecnologia pode ser utilizada tanto para o front-end – parte visual que interage com o usuário – quanto para o back-end – parte não visual que lida com o servidor – sendo uma linguagem muito vasta, com diversas funcionalidades.

# FORMAS DE UTILIZAR O JAVASCRIPT

Existem três formas principais de utilizar o JavaScript em uma página web: **inline**, **interno** e **externo**. Cada uma delas possui características específicas e é utilizada conforme a necessidade do projeto.

# INLINE

O JavaScript é **inserido diretamente no elemento HTML**, através de atributos como `onClick=""` e `onChange=""`. Essa abordagem é útil para pequenas interações, mas não é recomendada para códigos mais complexos, pois pode dificultar a organização do código.

```
<BUTTON ONCLICK="LET EXEMPLO = 'SCRIPT INLINE';"></BUTTON>
```

# INTERNO

O código **JavaScript** é escrito dentro de uma tag `<script>` que pode ser inserida no `<head>` ou no `<body>` do documento HTML. Essa abordagem permite agrupar o código dentro da própria página, facilitando a manipulação de elementos HTML sem precisar de arquivos externos.

```
<SCRIPT DEFER>
```

```
    LET EXEMPLO = 'SCRIPT INTERNO';
```

```
</SCRIPT>
```

# EXTERNO

O JavaScript é **armazenado em um arquivo separado** com a extensão **.js**. Esse arquivo é vinculado ao documento HTML através da tag **<script>**, utilizando o atributo **src**. Essa prática é a mais recomendada para projetos maiores, pois melhora a organização, **facilita a reutilização do código e otimiza o carregamento da página**.

```
<SCRIPT SRC="NOME_DO_ARQUIVO.JS" DEFER></SCRIPT>
```



# ATRIBUTO DEFER

O atributo **defer** é utilizado na tag **<script>** para indicar que o JavaScript deve ser executado somente após o carregamento completo do conteúdo HTML da página, ou seja, ele adia a execução do script até que o HTML esteja totalmente processado.

# COMENTARIOS

Os comentários no JavaScript são declarados da seguinte maneira:

```
// COMENTARIO
```

## ATALHO NO VS CODE

CTRL

+

;

# O QUE É UMA VARIÁVEL

Uma variável é um espaço na memória reservado para armazenar um valor. Esse valor pode ser de diferentes tipos, como:

- **String:** Variáveis do tipo texto.
- **Número real:** Valores numéricos com casas decimais.
- **Número inteiro:** Valores numéricos sem casas decimais.
- **Booleano:** Variáveis do tipo lógico, que podem ter os valores verdadeiro ou falso (ou, em termos numéricos, 1 ou 0).

# SINTAXE DAS VARIÁVEIS JAVASCRIPT

No JavaScript, existem duas formas de declarar uma variável:

- **var:** Uma palavra reservada que cria uma variável.

```
VAR NOME_VARIAVEL = 'VALOR';
```

- **let:** Outra palavra reservada para declarar uma variável, mas com uma característica importante.

```
LET NOME_VARIAVEL = 'VALOR';
```

# DIFERENÇA ENTRE VAR E LET

Ambas as palavras reservadas criam variáveis, mas a principal diferença entre elas é que, ao usar **let**, o JavaScript **emite um erro se houver a declaração de duas variáveis com o mesmo nome** dentro do mesmo escopo, evitando a redundância e ajudando a prevenir possíveis problemas durante a depuração do código.

Por isso, **a palavra reservada let** é considerada uma boa prática entre os programadores JavaScript, pois **traz mais segurança e clareza ao código, evitando bugs difíceis de identificar**.

# REGRAS PARA NOMEAR UMA VARIÁVEL

- Devem começar com uma letra;
- Não podem começar com números;
- Não podem conter espaços ou caracteres especiais (exceto `_` e `-`);
- São case-sensitive (`minhaVariavel`  $\neq$  `MinhaVariavel`);
- Não podem ser palavras reservadas (como `let`, `const`, `if`, `return`, etc.).

# CONSTANTE EM JAVASCRIPT

Uma **constante** é uma **variável cujo valor não pode ser alterado** após sua definição. No JavaScript, as constantes são declaradas utilizando a palavra-chave **const**. Ao contrário das variáveis declaradas com **var** ou **let**, que podem ter seus valores alterados durante a execução.

```
CONST PI = 3,14;
```

# TIPOS DE VARIÁVEIS (INTEIRO, REAL, STRING, BOOLEAN)

Como mencionado anteriormente, as variáveis são responsáveis por armazenar valores, e esses valores podem ser de diferentes tipos. No JavaScript, não precisamos especificar explicitamente o tipo de uma variável, pois a linguagem possui uma **tipagem dinâmica**. Isso significa que o JavaScript permite o armazenamento de qualquer tipo de valor em uma variável, **esem a necessidade de declaração prévia**, o tipo de variável é determinada ao decorrer da execução do programa. Contudo, **é importante entender em quais situações cada tipo de dado se encaixa**.



# STRING

Uma **string** é um tipo de dado que armazena uma sequência de caracteres, como letras, números e símbolos especiais. É muito usada para representar dados textuais. Exemplos de situações em que usamos strings incluem:

- Nomes;
- Textos (mensagens, frases);
- URLs (endereço de sites).

# NÚMERO INTEIRO

O tipo **número inteiro** armazena valores numéricos inteiros, ou seja, números sem casas decimais. Eles podem ser tanto positivos quanto negativos. Exemplos de casos em que usamos números inteiros:

- Quantidade de pessoas;
- Quantidade de objetos;
- Idade.

# NÚMERO REAL

O tipo **número real** é utilizado para armazenar números que possuem casas decimais. Este tipo é muito útil para representar valores que requerem precisão com frações. Exemplos de casos onde se usa números reais:

- Dinheiro (ex: 19.99);
- Altura (ex: 1.75 metros);
- Temperatura (ex: 36.6°C).

# BOOLEANO

O tipo **booleano** é um tipo de dado que só pode ter dois valores possíveis:

- Verdadeiro (true) ou Falso (false);
- Ou, no formato numérico, 1 (verdadeiro) ou 0 (falso);
- Esse tipo é comumente utilizado em condições e testes lógicos, por exemplo, para verificar se algo é verdadeiro ou falso.

# OPERADORES

Os operadores **são símbolos utilizados para realizar operações** sobre valores e variáveis. Eles podem ser classificados em diferentes categorias conforme sua função. São eles:

- Operadores de **atribuição**;
- Operadores **aritméticos**;
- Operadores de **incremento e decremento**;
- Operadores **relacionais (de comparação)**;
- Operadores **lógicos**.

# OPERADORES DE ATRIBUIÇÃO

Os operadores de atribuição **servem para definir ou modificar** o valor de uma variável.

- O símbolo `=` atribui um valor a uma variável;
- O operador `+=` adiciona um valor ao já existente na variável;
- O operador `-=` subtrai um valor do já armazenado na variável.

**Ex.:** Se uma variável tem o valor 10 e utilizamos `+= 5`, ela passará a ter o valor 15. Da mesma forma, se aplicarmos `-= 2`, o valor será reduzido para 13.

# OPERADORES ARITMÉTICOS

Esses operadores são **usados para realizar cálculos matemáticos**:

- **+** (adição) soma dois valores;
- **-** (subtração) reduz um valor pelo outro;
- **\*** (multiplicação) multiplica dois valores;
- **/** (divisão) divide um valor pelo outro;
- **%** (módulo) retorna o resto da divisão entre dois números;
- **\*\*** (exponenciação) eleva um número à potência de outro.

**Ex.:** Ao **multiplicar** dois números, obtemos um produto; ao **dividir** um número por outro, podemos ter um valor exato ou um número decimal, dependendo do caso.

# OPERADORES DE INCREMENTO E DECREMENTO

Os operadores de incremento e decremento, ao serem aplicados a uma variável, **aumentam ou reduzem seu valor em 1 unidade.**

- **++n, n++** Incremento;
- **--n, n--** Decremento.

**Ex.:** Se uma variável tem o valor 3 e aplicamos um **incremento**, ela passará a valer 4. Com o **decremento**, retornará a 3.

# OPERADORES RELACIONAIS (DE COMPARAÇÃO)

Esses operadores são **usados para comparar valores** e retornam um resultado verdadeiro ou falso.

- **=** verifica se dois valores são **iguais**, independentemente do tipo;
- **===** verifica se dois valores e seus tipos são **idênticos**;
- **!=** verifica se dois valores são **diferentes**;
- **!==** verifica se os **valores e seus tipos são diferentes**;
- **>** verifica se um valor é **maior** que outro;
- **<** verifica se um valor é **menor** que outro;
- **>=** verifica se um valor é **maior ou igual** a outro;
- **<=** verifica se um valor é **menor ou igual** a outro.

**Ex.:** Ao comparar dois números, podemos verificar se um é **maior**, **menor** ou **igual** ao outro. A diferença entre **=** e **===** está no fato de que **===** também verifica se os valores possuem o mesmo tipo.

# OPERADORES LÓGICOS

Os operadores lógicos **permitem combinar diferentes expressões booleanas** para criar condições mais complexas.

- **&&** (**E lógico**) retorna verdadeiro se ambas as condições forem verdadeiras;
- **||** (**OU lógico**) retorna verdadeiro se pelo menos uma das condições for verdadeira;
- **!** (**NÃO lógico**) inverte o valor de uma expressão lógica.

**Ex.:** Ao **avaliar duas condições ao mesmo tempo**, podemos exigir que ambas sejam verdadeiras (**&&**) ou permitir que apenas uma seja suficiente (**||**). Já o operador **!** **serve para inverter o resultado de uma condição**, tornando verdadeiro o que era falso e vice-versa.



# LAÇOS CONDICIONAIS

Um laço condicional é uma estrutura de código utilizada na programação, permitindo que o programa **tome diferentes caminhos**. Esses caminhos são acessados pela definição de determinadas condições. Os tipos de laços condicionais que veremos são:

- IF/Else
- Switch/case

# IF / ELSE

O laço condicional **IF** **avalia a condição fornecida**. Se a condição for verdadeira, o bloco de código dentro do **IF** será executado. Caso contrário, se não houver um **else**, nada será feito. Se houver um **else**, o bloco de código dentro do **else** será executado.

## Estrutura:

```
IF (CONDITION_1) { // ← CONDIÇÃO 1
    // BLOCO EXECUTADO CASO A PRIMEIRA CONDIÇÃO SEJA A VERDADEIRA
}
ELSE IF (CONDITION_2){ // ← CONDIÇÃO 2
    //BLOCO EXECUTADO CASO A SEGUNDA CONDIÇÃO SEJA A VERDADEIRA
}
ELSE {
    //BLOCO EXECUTADO CASO TODAS AS CONDIÇÃO FOREM FALSA
}
```

# SWITCH / CASE

O **switch / case** é um laço condicional que utiliza uma variável (**chave**) **para comparar com os valores definidos nos casos** (**case**) dentro do **switch**. Se um dos valores for igual ao da chave, o código correspondente será executado. O **switch case** é útil quando existem várias opções e pode ser mais organizado do que múltiplos **if**.

## Estrutura:

```
SWITCH (KEY) { // ← CHAVE PARA COMPARAÇÃO DOS VALORES
```

```
    CASE VALUE: // ← VALOR PARA SER COMPARADO COM A CHAVE
```

```
        //BLOCO DE CÓDIGO A SER EXECUTADO CASO O VALOR DO CASE FOR  
        IGUAL O DA KEY
```

```
    BREAK; // ← INTERROMPE A EXECUÇÃO
```

```
    DEFAULT: // ← OUTRA ALTERNATIVA DE RESPOSTA PARA CASO
```

```
        //BLOCO DE CÓDIGO A SER EXECUTADO CASO NENHUM DOS VALORES  
        SEJAM IGUAIS
```

```
    BREAK; // ← INTERROMPE A EXECUÇÃO
```

```
}
```

# LAÇOS DE REPETIÇÃO

Um laço de repetição (também chamado de loop) é uma estrutura usada em programação para repetir um bloco de código várias vezes, até que uma determinada condição seja satisfeita (ou deixe de ser).

É útil quando você quer executar a mesma tarefa várias vezes sem ter que escrever o mesmo código repetidamente. Os tipos de laços condicionais que veremos são:

- While
- Do / While
- For
- ForEach

# WHILE

O laço **while** repete um bloco de código enquanto a condição fornecida **for verdadeira**. Ou seja, ele verifica a condição antes de cada execução parando de repetir apenas quando a condição for falsa.

## Estrutura:

```
WHILE (CONDITION) { // ← CONDIÇÃO  
    // CÓDIGO A SER EXECUTADO ENQUANTO A CONDIÇÃO FOR VERDADEIRA  
}
```

# DO / WHILE

O laço **do/while** é semelhante ao **while**, mas a principal diferença é que no **do/while**, ele executa o bloco de código **pelo menos uma vez**, e depois continua executando enquanto a condição for **verdadeira**. Em outras palavras, no **do/while**, a condição é verificada **após** a execução do bloco de código, o que garante que o código será executado pelo menos uma vez, independentemente da condição ser verdadeira ou falsa no início.

## Estrutura:

```
DO {  
    // BLOCO DE CÓDIGO A SER EXECUTADO  
} WHILE (CONDITION); // ← CONDIÇÃO A SER VERIFICADA AO FINAL DA EXECUÇÃO
```

# FOR

O laço de repetição **for** é **utilizado quando sabemos o número de iterações que o laço deve realizar**. Ele usa uma variável, chamada de índice ou contador (geralmente representada como i ou ind), que é iniciada com um valor e, a cada iteração, é atualizada até atingir um valor final. O laço pode ser configurado para ser crescente ou decrescente, dependendo de como o índice é manipulado.

## Estrutura:

```
FOR (LET IND = 0; IND < 10; I++) {  
    //EXEMPLO INDO DE 0 A 10  
    // OU SEJA ESSE CÓDIGO VAI SE REPETIR 10 VEZES  
}
```



# FOREACH

O **forEach** é um **método utilizado para percorrer elementos de um array ou objeto**. Ele executa uma função para cada item do **array** ou **objeto**, facilitando a iteração.

## Estrutura:

```
CONST ARR = [1, 2, 3, 4];  
ARR.FOREACH(FUNCTION(ELEMENT) {  
    // CÓDIGO A SER EXECUTADO PARA CADA ELEMENTO  
    CONSOLE.LOG(ELEMENT);  
});
```


# FUNÇÃO

**Uma função é uma receita pronta** que pode ser usada e reutilizada sempre que necessário. Em programação, funções são **blocos de código reutilizáveis que executam uma tarefa específica**. Elas ajudam a organizar o código, evitando repetições e facilitando a manutenção do programa, já que o código pode ser reutilizado em várias partes do programa sem ser reescrito.



## O QUE É UM PARÂMETRO:

Os **parâmetros** de uma função são **valores** ou **informações** que são **passadas para** ela no momento em que a **função** é chamada. Esses valores podem ser utilizados dentro da função para realizar operações ou retornar resultados. Os parâmetros permitem que a função seja mais flexível, permitindo que ela execute a mesma tarefa com diferentes entradas.



## Estrutura:

Estrutura para **construir uma função**:

```
FUNCTION NOME_DA_FUNÇÃO(RECEBENDO_PARAMETRO1, RECEBENDO_PARAMETRO2){  
  // BLOCO DE CÓDIGO DENTRO DA FUNÇÃO  
}
```

Estrutura para **chamar a função** construída:

```
NOME_DA_FUNÇÃO(PASSANDO_PARAMETRO1, PASSANDO_PARAMETRO2);
```

Quando chamamos uma **função**, **atribuímos determinados valores como parâmetros**. **Esses valores vão ser passados para função** em que desejamos utilizar. **A função recebe então os valores passados**, e esses poderão ser utilizados ao decorrer da execução dos códigos presentes na função.

# VARIÁVEIS GLOBAIS E LOCAIS – DIFERENÇA

- **Variáveis globais** são aquelas definidas fora de qualquer função e podem ser acessadas e modificadas de qualquer parte do código, inclusive dentro de funções. Elas têm um **escopo global**.
- **Variáveis locais**, por outro lado, são aquelas definidas dentro de uma função e só podem ser acessadas e utilizadas dentro daquela função. Elas têm um **escopo local**, ou seja, seu valor não pode ser acessado fora da função onde foi definida.

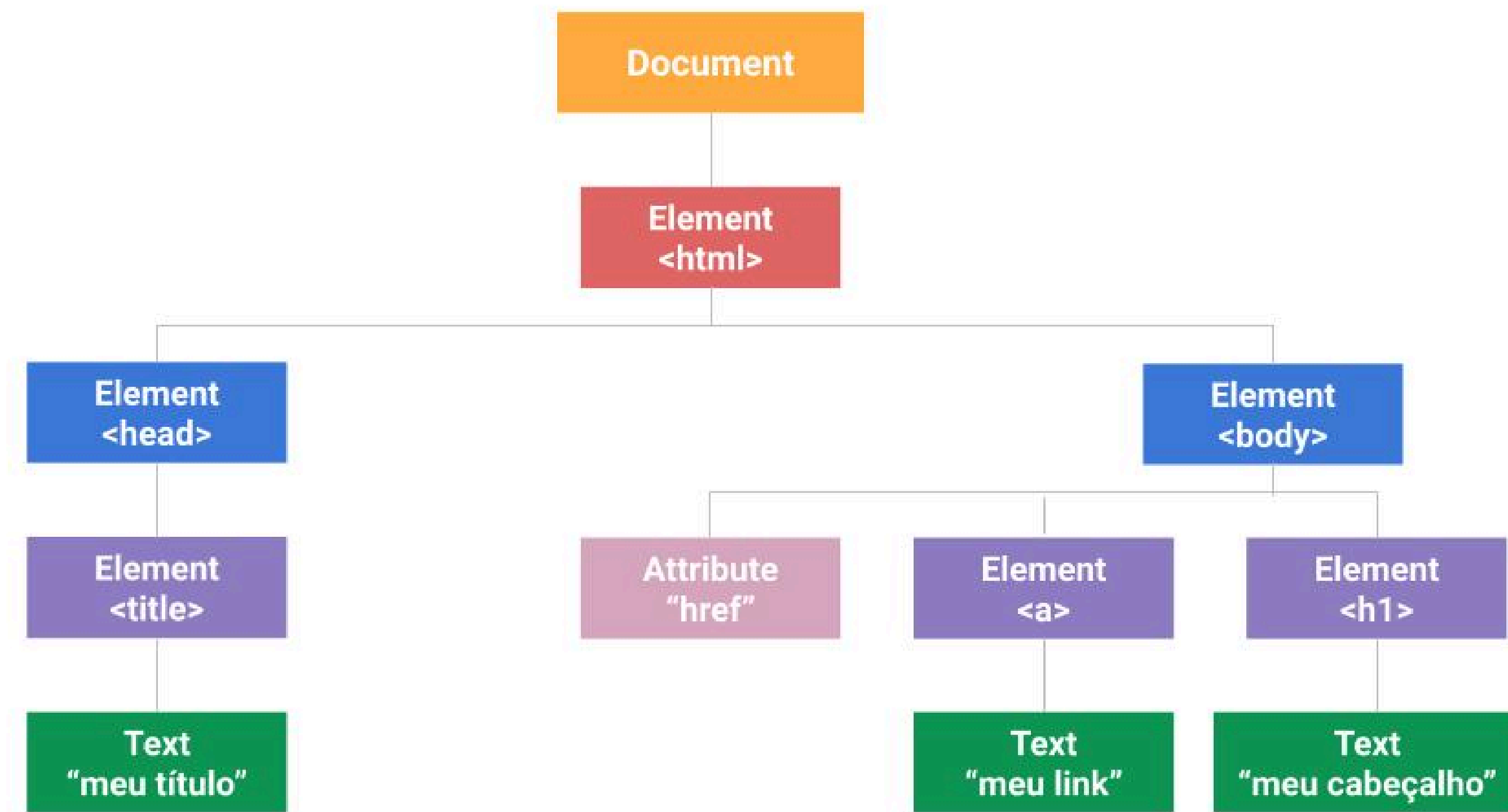
# MANIPULAÇÃO DO DOM

## O QUE É O DOM:

**DOM** significa "**Document Object Model**", ou, em português, **Modelo de Objeto do Documento**. Ele é a forma como o navegador **representa a estrutura de uma página web**.

Podemos imaginar o **DOM** como uma **árvore de elementos**, onde **cada tag HTML vira um objeto que pode ser acessado e modificado com JavaScript**.

# ARVORE DE ELEMENTOS:



## O QUE É A MANIPULAÇÃO DO DOM:

**Manipular o DOM** significa **adicionar, remover ou alterar elementos da página** — como **trocar um texto, esconder uma imagem ou mudar a cor de um botão**, por exemplo.

**Essa manipulação é feita usando JavaScript**, que interage com o HTML e, indiretamente, pode influenciar também o CSS.



# 1 - SELECIONAR ELEMENTOS:

- `document.getElementById("nome_id")`:

O método `document.getElementById("nome_id")` seleciona um elemento HTML pelo seu **ID**.

```
DOCUMENT.GETELEMENTBYID("NOME_ID");
```

- `document.querySelector("seletor")`:

O método `document.querySelector("seletor")` **busca um elemento HTML utilizando um seletor CSS**. Isso significa que ele pode selecionar um elemento por classe (definido por `."`), ID (definido por `"#"`), ou até mesmo pelo nome da tag igual fazemos no CSS.

```
DOCUMENT.QUERYSELECTOR("SELETOR");
```

- `document.querySelectorAll("seletor")`:

O método `document.querySelectorAll("seletor")` funciona da mesma forma que `querySelector`, porém, **ao invés de retornar apenas um elemento, ele retorna todos os elementos que correspondem ao seletor especificado**.

```
DOCUMENT.QUERYSELECTORALL("SELETOR");
```

## 2 - CRIAR ELEMENTOS E CLASSES:

- **document.createElement("tag"):**

Cria um novo elemento HTML.

```
DOCUMENT.CREATEELEMENT("TAG");
```

- **element.appendChild(novo\_elemento):**

Adiciona um novo elemento como filho.

```
ELEMENT.APPENDCHILD(NOVO_ELEMENTO);
```

- **element.classList.add("classe"):**

Adiciona uma classe ao elemento.

```
ELEMENT.CLASSLIST.ADD("NOME_DA_CLASSE");
```

- **element.classList.remove("classe"):**

Remove uma classe do elemento.

```
ELEMENT.CLASSLIST.REMOVE("NOME_DA_CLASSE");
```

### 3 - ADICIONA CONTEÚDOS E ESTILOS:

- `element.innerHTML = ""`:

A propriedade `.innerHTML` permite definir ou obter o conteúdo HTML de um elemento.

```
ELEMENT.INSERTHTML = ""
```

- `element.style.property = ""`:

A propriedade `.style` é utilizada para alterar dinamicamente o estilo CSS de um elemento selecionado via JavaScript.

```
ELEMENT.STYLE.PROPERTY = ""
```

## 4 - MANIPULAÇÃO DE EVENTOS:

- `element.addEventListener("evento", funcao):`

**Adiciona um ouvinte de evento** a um elemento (exemplo: `click`, `mouseover`).

```
ELEMENT.ADDEVENTLISTENER("EVENTO", FUNCAO)
```

- `element.removeEventListener("evento", funcao):`

**Remove um evento** previamente adicionado.

```
ELEMENT.REMOVEEVENTLISTENER("EVENTO", FUNCAO)
```

# TIPOS DE EVENTOS:

## EVENTOS DE MOUSE

- **click** – Ocorre quando um **elemento é clicado**.
- **dblclick** – Ocorre quando um **elemento é clicado duas vezes**.
- **mouseover** – Ocorre quando o **cursor do mouse passa sobre um elemento**.
- **mouseout** – Ocorre quando o **cursor do mouse sai de um elemento**.
- **mousemove** – Ocorre quando o **mouse se move dentro de um elemento**.

## EVENTOS DE TECLADO

- **keydown** – Ocorre quando uma **tecla é pressionada**.
- **keyup** – Ocorre quando uma **tecla é solta**.
- **keypress** – Ocorre quando uma **tecla é pressionada e solta** (obsoleto nas versões modernas do JavaScript).

# TIPOS DE EVENTOS:

## EVENTOS DE FORMULÁRIO

- **submit** – Ocorre quando um **formulário é enviado**.
- **change** – Ocorre quando um **elemento de entrada muda de valor**.
- **focus** – Ocorre quando um **elemento recebe o foco**.
- **blur** – Ocorre quando um **elemento perde o foco**.

## EVENTOS DE CARREGAMENTO

- **load** – Ocorre quando um **recurso é completamente carregado**.
- **DOMContentLoaded** – Ocorre quando o **HTML é completamente carregado e analisado**.
- **unload** – Ocorre quando a **página está sendo descarregada**.

# TIPOS DE EVENTOS:

## EVENTOS DE MOUSE

- **touchstart** – Ocorre quando um **toque é detectado na tela.**
- **touchmove** – Ocorre quando um **toque é movido na tela.**
- **touchend** – Ocorre quando um **toque é finalizado.**

# ARRAY

Um **array** é uma estrutura de dados que permite armazenar uma **lista de valores** dentro de uma única variável. Os valores são acessados através de um **índice**, **que representa a posição do elemento dentro do array**. Os índices de uma array começam a partir do número 0, ou seja:

- **Índice[ 0 ]** → Primeiro elemento;
- **Índice[ 1 ]** → Segundo elemento;
- **Índice[ 2 ]** → Terceiro elemento, e assim por diante.



## Estrutura:

Os elementos de um array podem ser de **qualquer tipo**, como números, strings e valores booleanos.

```
// IND.0 IND.1 IND.2 IND.3  
LET ARRAY = ['RONI FERREIRA', 'LEONORA', 345, FALSE];
```

No exemplo acima:

- **array[ 0 ]** retorna "Roni Ferreira" (**string**);
- **array[ 1 ]** retorna "Leonora" (**string**);
- **array[ 2 ]** retorna 345 (**número**);
- **array[ 3 ]** retorna false (**booleano**).

# FUNÇÕES DE UM ARRAY

Os arrays possuem várias funções embutidas para **facilitar sua manipulação**, como:

- **push(valor)** → Adiciona um elemento no final do array;
- **pop()** → Remove o último elemento do array;
- **shift()** → Remove o primeiro elemento do array;
- **unshift(valor)** → Adiciona um elemento no início do array;
- **length** → Retorna o tamanho do array (**quantidade de elementos**);
- **indexOf(valor)** → Retorna o índice de um valor dentro do array;
- **includes(valor)** → Verifica se o valor existe no array (**retorna true ou false**).

# EXEMPLOS DAS FUNÇÕES DO ARRAY:

```
LET FRUTAS = ["MAÇÃ", "BANANA", "UVA"];
```

```
FRUTAS.PUSH("MANGA"); // ADICIONA "MANGA" AO FINAL
```

```
FRUTAS.POP(); // REMOVE O ÚLTIMO ELEMENTO
```

```
FRUTAS.UNSHIFT("LARANJA"); // ADICIONA "LARANJA" NO INÍCIO
```

```
FRUTAS.SHIFT(); // REMOVE O PRIMEIRO ELEMENTO
```

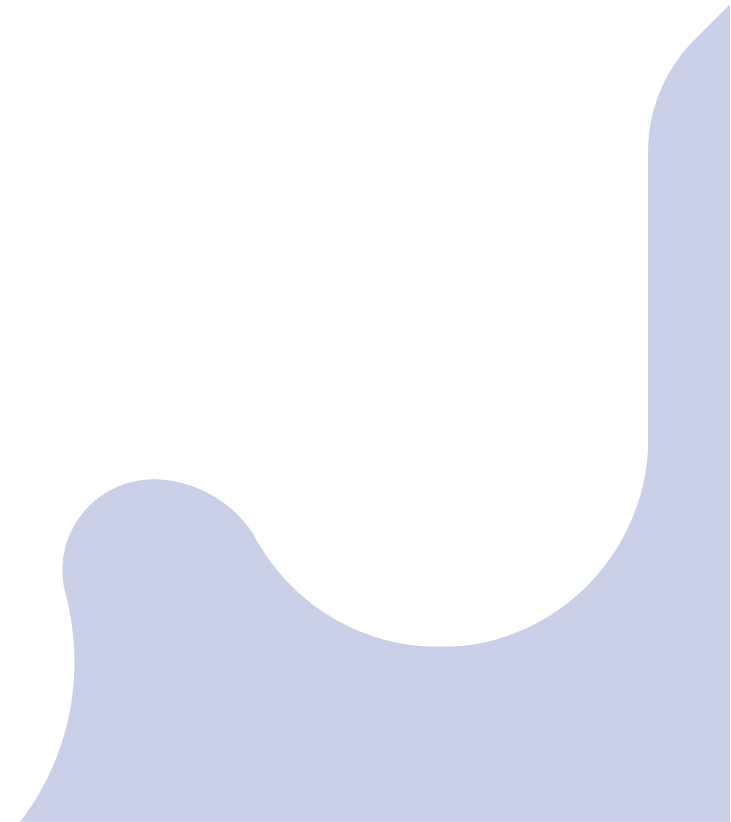

```
CONSOLE.LOG(FRUTAS.LENGTH); // RETORNA O TAMANHO DO ARRAY
```

```
CONSOLE.LOG(FRUTAS.INDEXOF("BANANA")); // RETORNA A POSIÇÃO  
DA "BANANA"
```



# MATRIZ

Uma matriz é um tipo especial de **array bidimensional**, onde os elementos são organizados em linhas e colunas. Essa estrutura é formada por **arrays dentro de um array**, ou seja, **cada elemento da matriz é um array individual**, permitindo a organização de dados em tabelas.



## Estrutura:

Visualização em linha, como um array:

```
LET MATRIZ = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ];
```

Visualização em linhas e colunas, como uma matriz:

```
LET MATRIZ = [  
  [1, 2, 3],  
  [4, 5, 6],  
  [7, 8, 9]  
];
```

# SINTAXE DA MATRIX:

Em uma **matriz**, utilizamos dois índices **[linha][coluna]** para acessar um elemento específico. No entanto, se omitirmos o segundo índice, estaremos tratando a linha inteira como um array. Isso significa que estaremos acessando o array que representa a linha da matriz, ao invés de um único elemento.

## Acessando um elemento específico:

- **matriz[ 0 ][ 0 ]** retorna o valor **[ 1 ]** (**primeira linha**)(**primeira coluna**);
- **matriz[ 0 ][ 1 ]** retorna o valor **[ 2 ]** (**primeira linha**)(**segunda coluna**);
- **matriz[ 1 ][ 0 ]** representa **[ 4 ]** (**segunda linha**)(**primeiro elemento**).

## Omitindo o índice da coluna e acessando a linha da matriz:

- **matriz[ 0 ]** representa **[ 1, 2, 3 ]** (**primeira linha**);
- **matriz[ 1 ]** representa **[ 4, 5, 6 ]** (**segunda linha**);
- **matriz[ 2 ]** representa **[ 7, 8, 9 ]** (**terceira linha**).

# FIM DO MÓDULO !

“A tecnologia é um mecanismo de liberação de recursos. Ela pode tornar abundante o que antes era escasso.”

Peter Diamandis