

# DOM e métodos de acesso

Daniel de Freitas

# Vinculando JS ao HTML

## O que o browser entende?

Os navegadores (browsers) compreendem as três 'linguagens' a seguir: HTML, CSS e JavaScript. Cada uma delas representa algum aspecto da página:

**HTML**



**Conteúdo**

**CSS**



**Apresentação**

**JS**



**Comportamento**

# JavaScript e ECMAScript

- **JavaScript** (JS) foi criada em 1995 por Brendan Eich, cofundador do *Mozilla*, na época em que trabalhava na Netscape.
- Em 1996 a linguagem foi submetida a **ECMA** (European Computer Manufacturers Association), visando sua padronização para uso por fabricantes de navegadores
- Em 1997 foi publicada a **ECMAScript**, primeira versão ECMA da JavaScript.

Name	Other Names	Release Date	Notes / Version Number
<a href="#">ES 2022</a>	ECMAScript 2022 / ES13		Version 13,
<a href="#">ES 2021</a>	ECMAScript 2021 / ES12	2021 June	Version 12, string.replaceAll
<a href="#">ES 2020</a>	ECMAScript 2020 / ES11	2020 June	Version 11, optional chaining
<a href="#">ES 2019</a>	ECMAScript 2019 / ES10	2019 June	Version 10, string.trimStart / trimEnd
<a href="#">ES 2018</a>	ECMAScript 2018 / ES9	2018 June	Version 9, for-await-of loop
<a href="#">ES 2017</a>	ECMAScript 2017 / ES8	2017 June	Version 8, async and await
<a href="#">ES 2016</a>	ECMAScript 2016 / ES7	2016 June	Version 7, Array.prototype.includes
<a href="#">ES 2015</a>	ECMAScript 2015 / ES6	2015 June	Version 6, let and const
<a href="#">ES 2009</a>	ECMAScript 2009 / ES5	2009 December	Version 5
ES 2008		2008 July	Version 4 was abandoned
ES 1999		1999 December	Version 3
ES 1998		1998 June	Version 2
ES 1997		1997 June	Version 1

Fonte:  
<https://bettersolutions.com/javascript/syntax/versions.htm>

# ECMAScript: desenvolvimento das versões

- O grupo TC39 da ECMA desenvolve as versões de maneira **colaborativa com a comunidade**
- As especificações são desenvolvidas via **GitHub** a partir do repositório <https://github.com/tc39/ecma262>
- É possível acompanhar tudo sobre as próximas versões em: <https://tc39.es/ecma262/#sec-overview>

Draft ECMA-262 / July 7, 2022

## ECMAScript® 2023 Language Specification



### About this Specification

The document at <https://tc39.es/ecma262/> is the most accurate and up-to-date ECMAScript specification. It contains the [finished proposals](#) (those that have reached Stage 4 in the proposal process and thus are implemented in several implementations since that snapshot was taken).

This document is available as a [single page](#) and as [multiple pages](#).

### Contributing to this Specification

This specification is developed on GitHub with the help of the ECMAScript community. There are a number of ways to contribute.

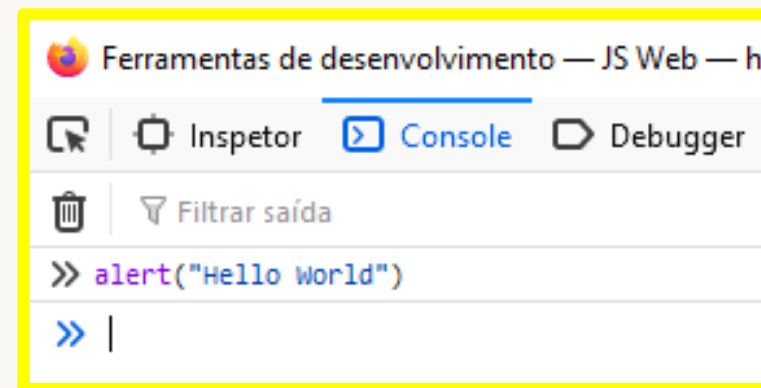
GitHub Repository: <https://github.com/tc39/ecma262>



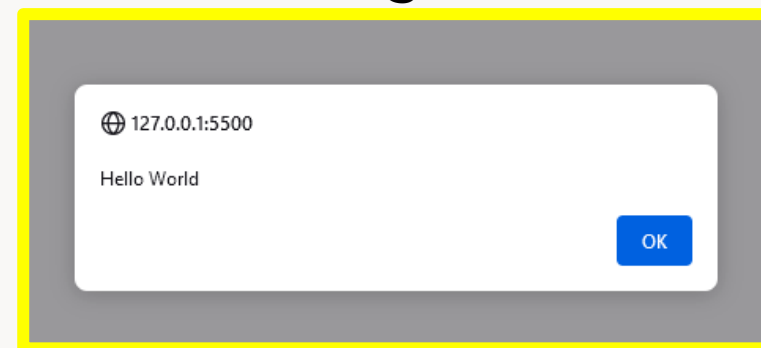
# JavaScript no navegador:

- É possível inserir diversos comandos em JavaScript apenas pelo navegador;
- Para isso basta abrir o navegador e acessar a aba do “**Console**”;
- Com o comando **alert(“Hello World”)** é possível apresentar uma mensagem ao usuário via caixa de diálogo;

## Console



## Navegador

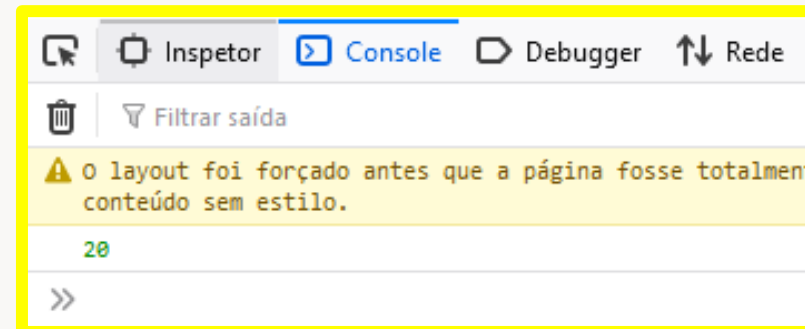


# JavaScript com HTML: inline-script

- Ao vincular o JavaScript com o HTML via tag **<script>** no head do documento, ao abrir o site via live server o script já é executado (**inline-script**);
- Qualquer manipulação de variáveis e valores dentro de um script só será apresentado via console se o desenvolvedor colocar o comando **console.log()**

```
<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <link rel="stylesheet" href="style.css" />
  <title>JS Web</title>
  <script>
    let mensagem = "Hello World";
    if (mensagem == "Hello World") alert(mensagem);

    let numero = 20;
    console.log(numero);
  </script>
</head>
```



# JavaScript com HTML: external-script

- Para criar um **external-script** é necessário criar um documento **.js** no diretório de trabalho e preenche-lo com o código desejado;
- Ele só será executado quando o documento **.js** for vinculado ao documento HTML;



```
VinculandoScripts > JS script.js > ...  
1  let mensagem2 = "Este é um script externo";  
2  alert(mensagem2);  
3
```



# JavaScript com HTML: external-script

- Para vincular o external-script ao HTML, utiliza-se a tag **<script>** com atributo *src*, especificando o documento em questão;

```
<body>  
  <h1>Titulo</h1>  
  <p>  
    Lorem ipsum dolor sit amet conse  
    eveniet aperiarn libero repudiand  
    reprehenderit officia. Voluptati  
  </p>  
  <script src="script.js"></script>  
</body>
```

# Funcionamento do Browser

## Funcionamento do Browser:

Para compreender as diferentes formas de utilização da tag **<script>** é necessário assimilar como o navegador carrega essa página ao longo do tempo. Tem-se que ter em vista os seguintes conceitos:

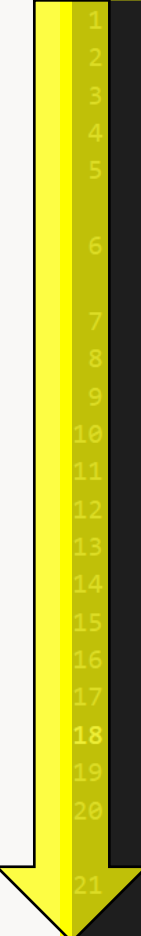
- **Parsing HTML;**
- **Fetch Script;**
- **Execute Script**

# Funcionamento do Browser: Parsing HTML

O termo **Parsing** significa analisar e converter um programa para um formato interno em que ele possa ser executado.

No contexto do HTML, o browser realiza o parse do documento em uma estrutura denominada **DOM tree**, que veremos mais adiante no curso.

O importante é saber que, nesse processo, o documento HTML é lido de cima para baixo, linha a linha. Certas vezes esse processo é interrompido para a requisição e execução de arquivos.

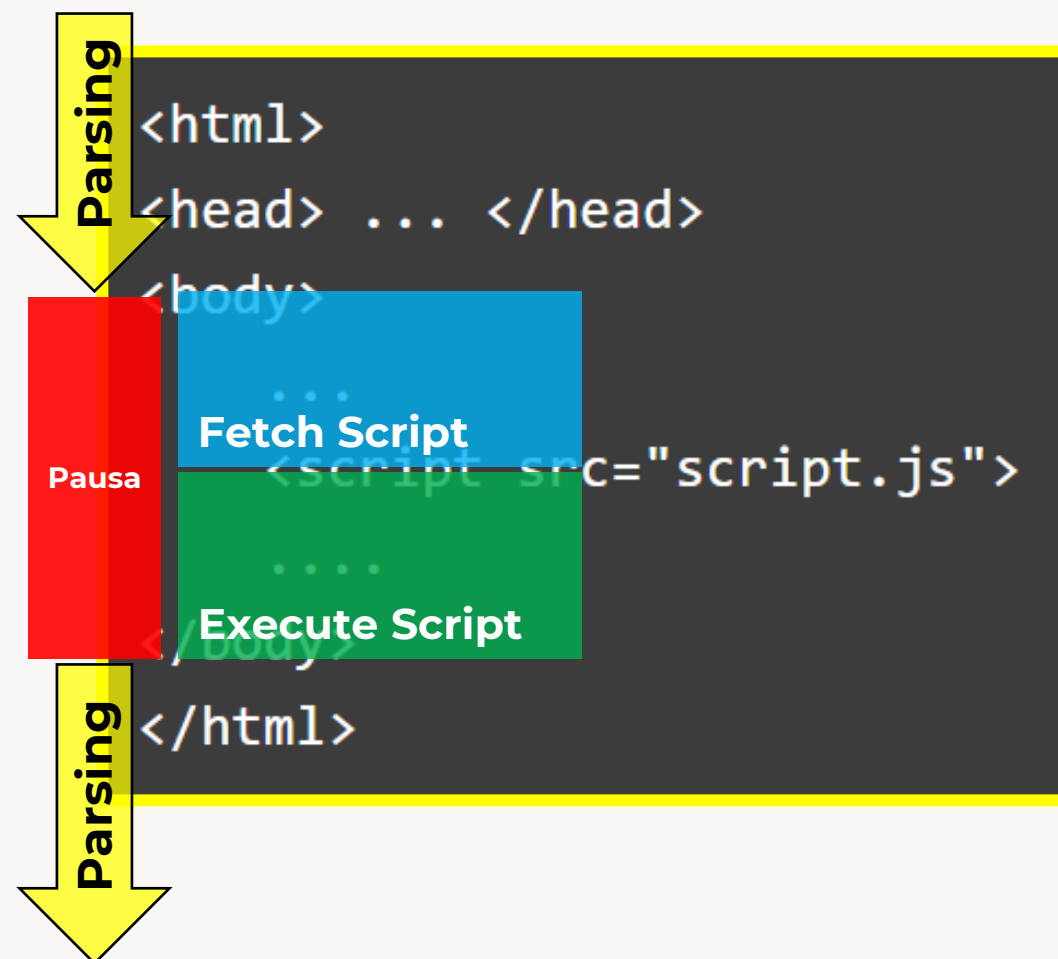


```
1 <!DOCTYPE html>
2 <html lang="pt-br">
3   <head>
4     <meta charset="UTF-8" />
5     <meta http-equiv="X-UA-Compatible"
6       content="IE=edge" />
7     <meta name="viewport" content="width=device-width,
8       initial-scale=1.0" />
9     <link rel="stylesheet" href="style.css">
10    <title>JS Web</title>
11    <script>
12      let mensagem = "Hello World";
13      if (mensagem == "Hello World") alert(mensagem);
14
15      let numero = 20;
16      console.log(numero);
17    </script>
18  </head>
19  <body>
20    <h1>Titulo</h1>
21    <p>
22      Lorem ipsum dolor sit amet consectetur
23      adipisicing elit. Id laborum
24      eveniet aperiam libero repudiandae doloremque
25      harum accusantium
```

# Parsing HTML: script regular

Quando se utiliza o inline-script ou a tag `<script>` apenas com o atributo `src`, o **Parsing do HTML é interrompido** para que no:

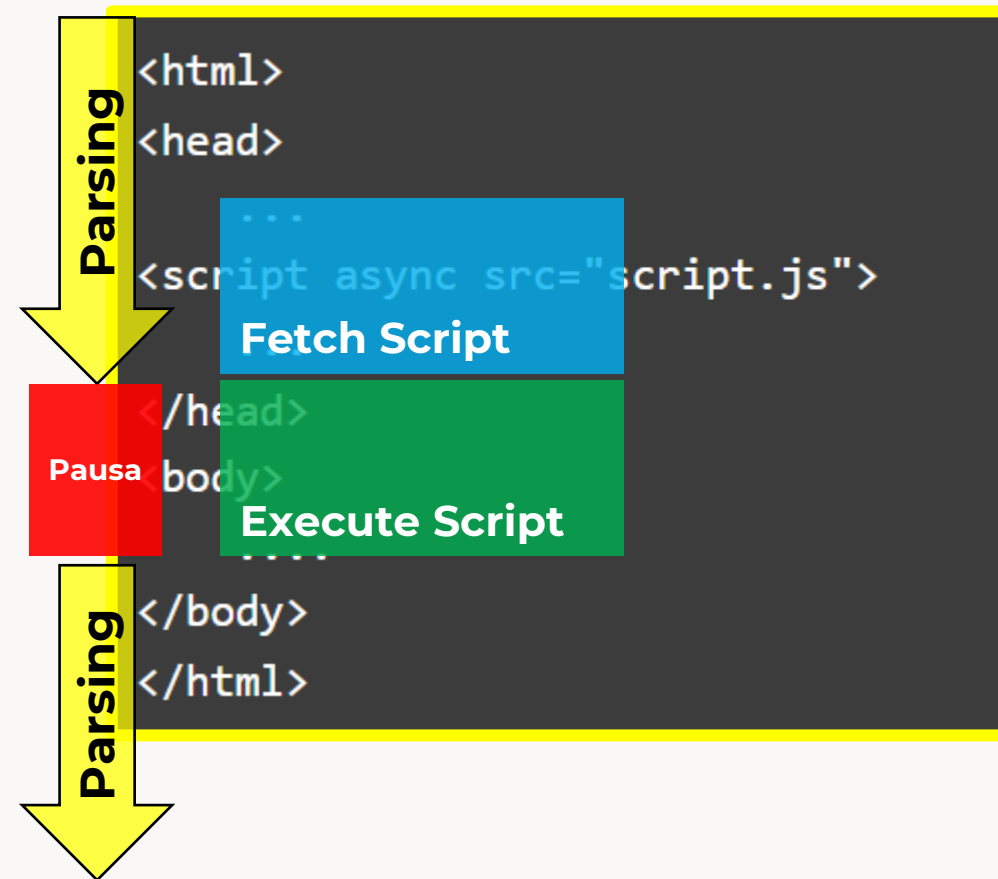
- **Inline-script:** o código seja executado (**Execute Script**);
- **`<script src = "...">`:** o código seja buscado (**Fetch Script**) e, posteriormente, executado (**Execute Script**). Caso seja colocado no final do `<body>`, por exemplo, o script é executado depois do parsing do HTML.





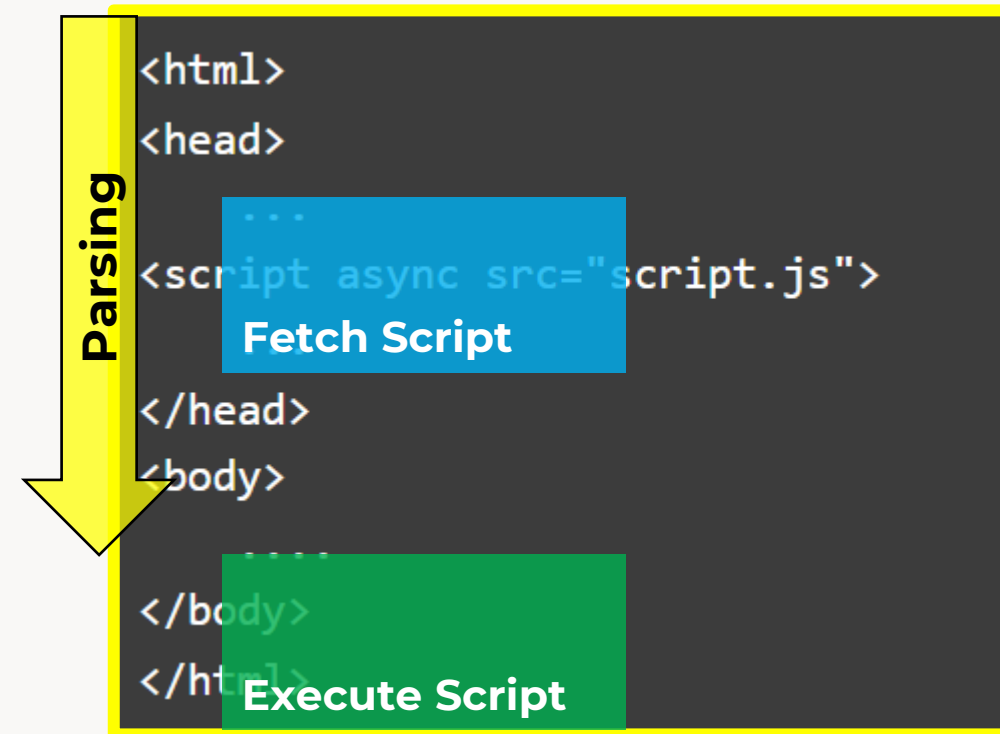
## Parsing HTML: <script async>

- Ao utilizar-se a tag **<script>** no head do documento com o atributo **async** outra dinâmica de Parsing é apresentada, o script será **requisitado assincronamente**.
- A requisição (**Fetch Script**) pelo arquivo é feita paralelamente, enquanto o Parsing continua.
- A execução (**Execute Script**) do arquivo pode acontecer a qualquer momento, uma vez que o arquivo já foi carregado.



## Parsing HTML: <script defer>

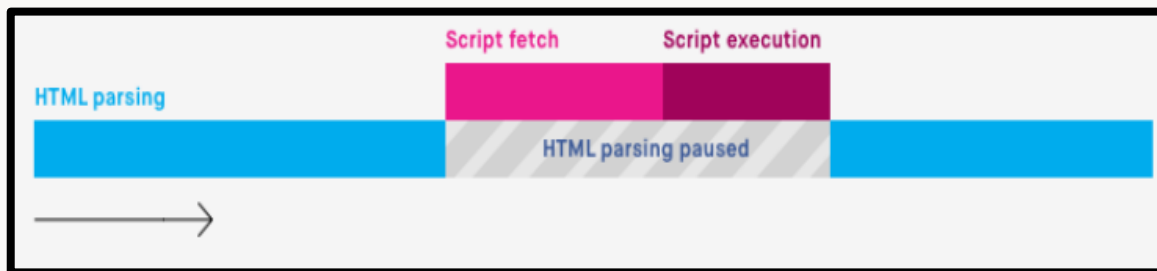
- Ao utilizar-se a tag **<script>** no head do documento com o atributo **defer** outra dinâmica de Parsing é apresentada, o script será **requisitado assincronamente e executado ao final do Parsing**.
- A requisição (**Fetch Script**) pelo arquivo é feita paralelamente, enquanto o Parsing continua.
- A execução (**Execute Script**) do arquivo pode acontecer somente ao final do Parsing, mesmo que o arquivo já tenha sido carregado (**Fetch Script**).



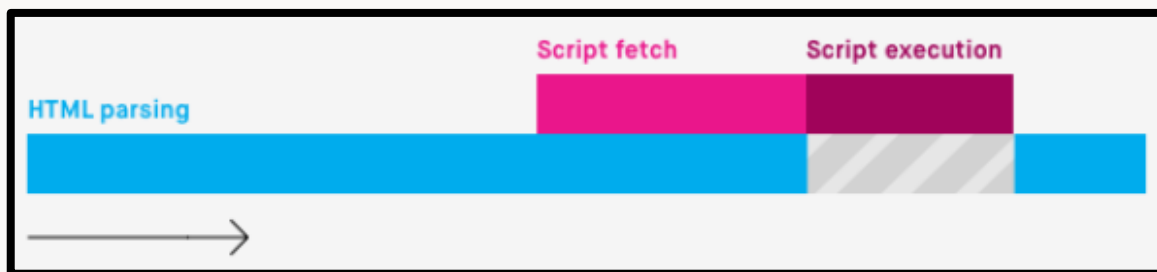
# Parsing HTML: overview

- Nos exemplos de muitos vídeos serão utilizados os **scripts regulares** ao final do body. Portanto na maneira regular. Porém, o Parsing já será encerrado antes do Fetch e Execution;
- No geral, a melhor solução é utilizar o **<script>** no head com o atributo **defer**, pois garante a ordem da execução do script, diferentemente que o async.

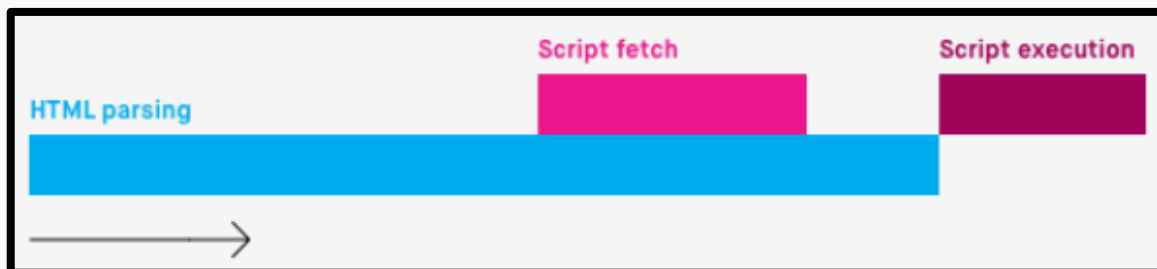
## Regular



## Async



## Defer



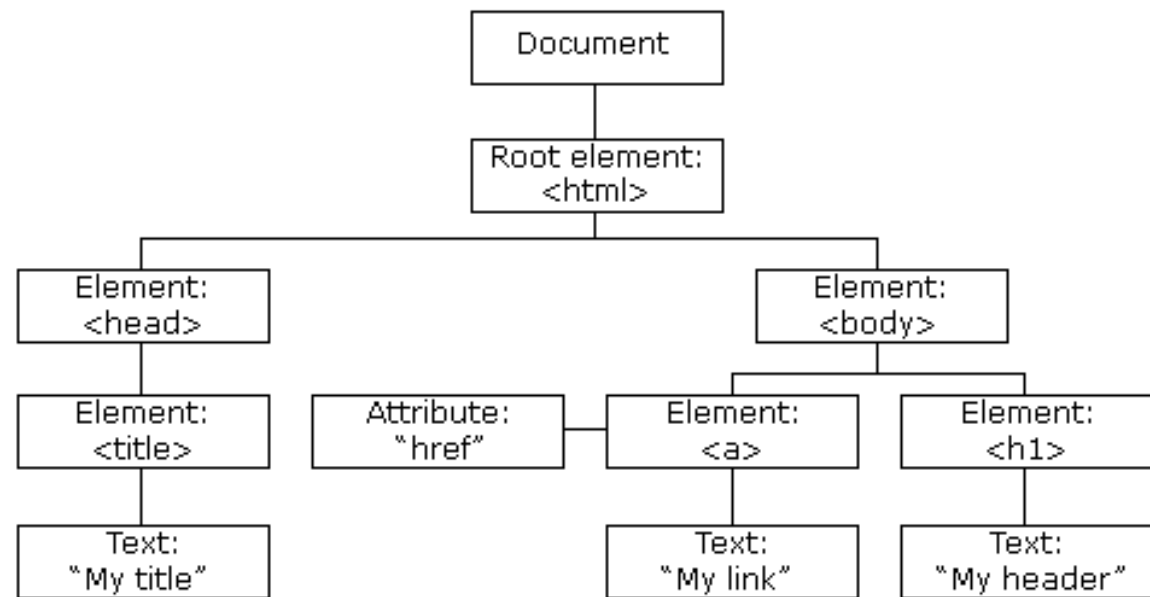


GERMINARE  
Escola de Negócios

DOM

## DOM: definição

- “O Document Object Model (DOM) é uma interface de programação para os documentos HTML. Representa a página de forma que os **programas possam alterar a estrutura do documento**, alterar o estilo e conteúdo. O DOM representa o documento com nós e objetos, dessa forma, as linguagens de programação podem se conectar à página.” – MDN



Fonte: [https://www.w3schools.com/js/js\\_htmlDOM.asp](https://www.w3schools.com/js/js_htmlDOM.asp)

- Assim que a página HTML carrega o DOM é criado automaticamente pelo navegador



# DOM: definição

- **O DOM não pertence ao JavaScript**, mas é uma Web API que pode interagir com ele. Uma lista de Web APIs podem ser acessadas na documentação : <https://developer.mozilla.org/pt-BR/docs/Web/API>
- O DOM é basicamente uma conexão entre o documento HTML e o código JavaScript


## D


- [DOM](#)

## E

- [Encoding API](#)
- [Encrypted Media Extensions \(en-US\)](#)
- [EyeDropper API \(en-US\)](#)

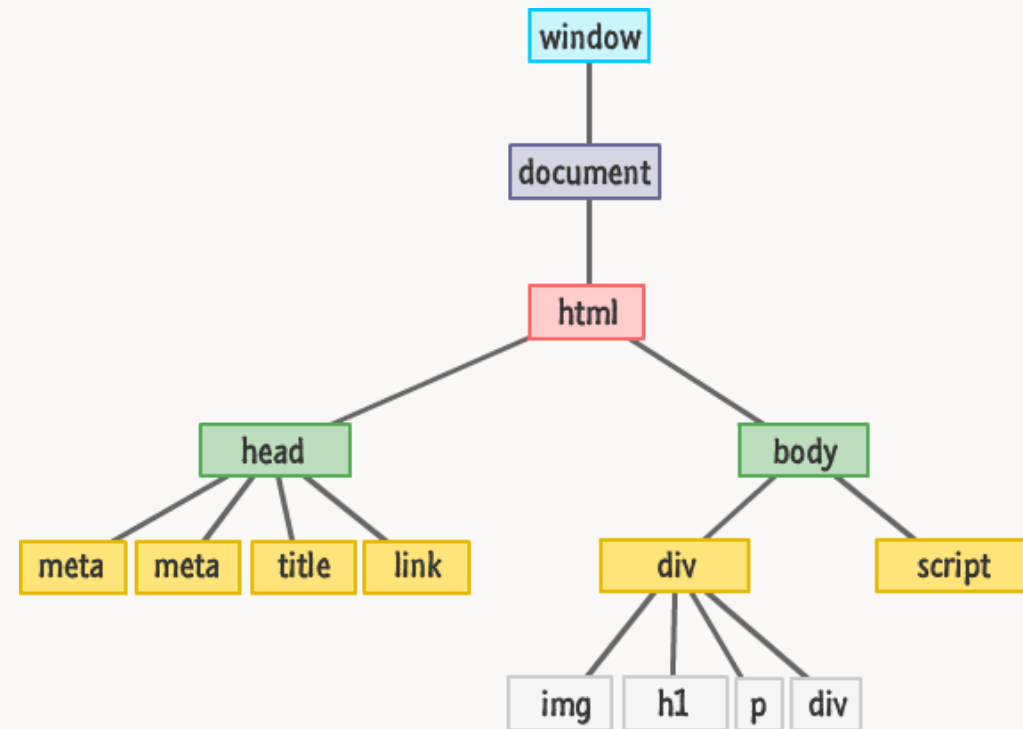
## F

- [Fetch API](#) 
- [File System Access API \(en-US\)](#)
- [File and Directory Entries API \(en-US\)](#)
- [Fullscreen API \(en-US\)](#)

- [Page Visibility API](#)
- [Payment Request API \(en-US\)](#)
- [Performance API](#)
- [Performance Timeline API \(en-US\)](#)
- [Periodic Background Sync \(en-US\)](#)
- [Permissions API \(en-US\)](#)
- [Picture-in-Picture API \(en-US\)](#)
- [Pointer Events \(en-US\)](#)
- [Pointer Lock API \(en-US\)](#)
- [Presentation API \(en-US\)](#)
- [Proximity Events \(en-US\)](#)
- [Push API](#) 

## DOM: definição

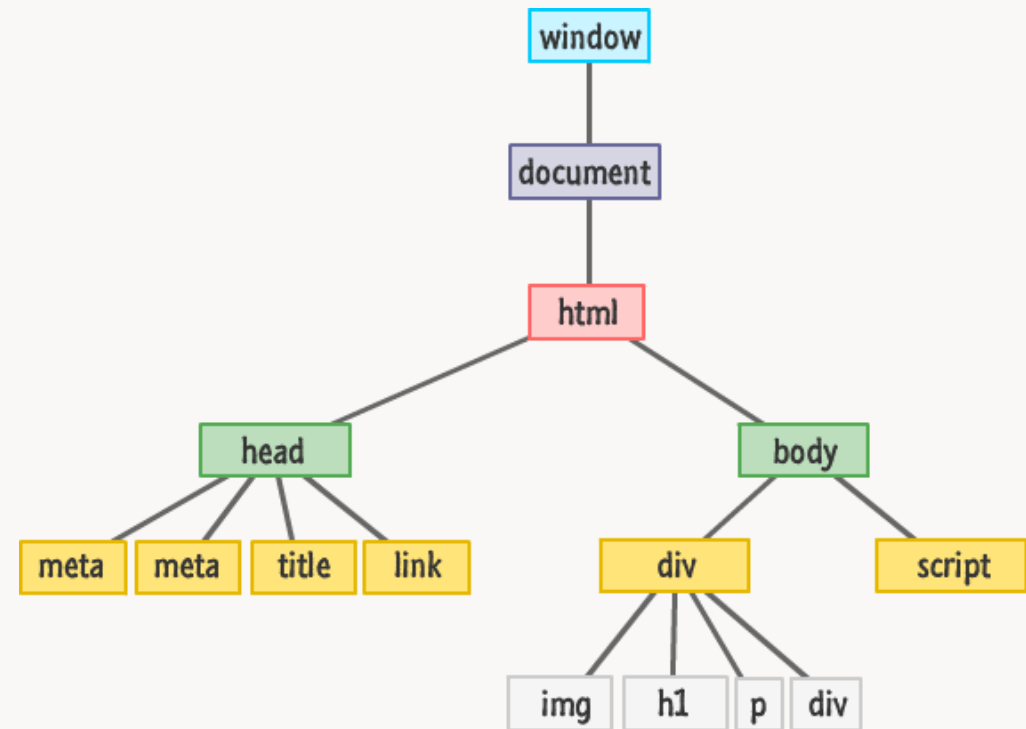
- O objeto **window** representa uma janela que contém o DOM;
- O DOM se inicia com um objeto **document** no topo da árvore. Esse objeto pode ser acessado via JavaScript, ele é o ponto de entrada para o DOM;
- É por meio do **document** que se interage com o DOM;



# Node e HTML Element

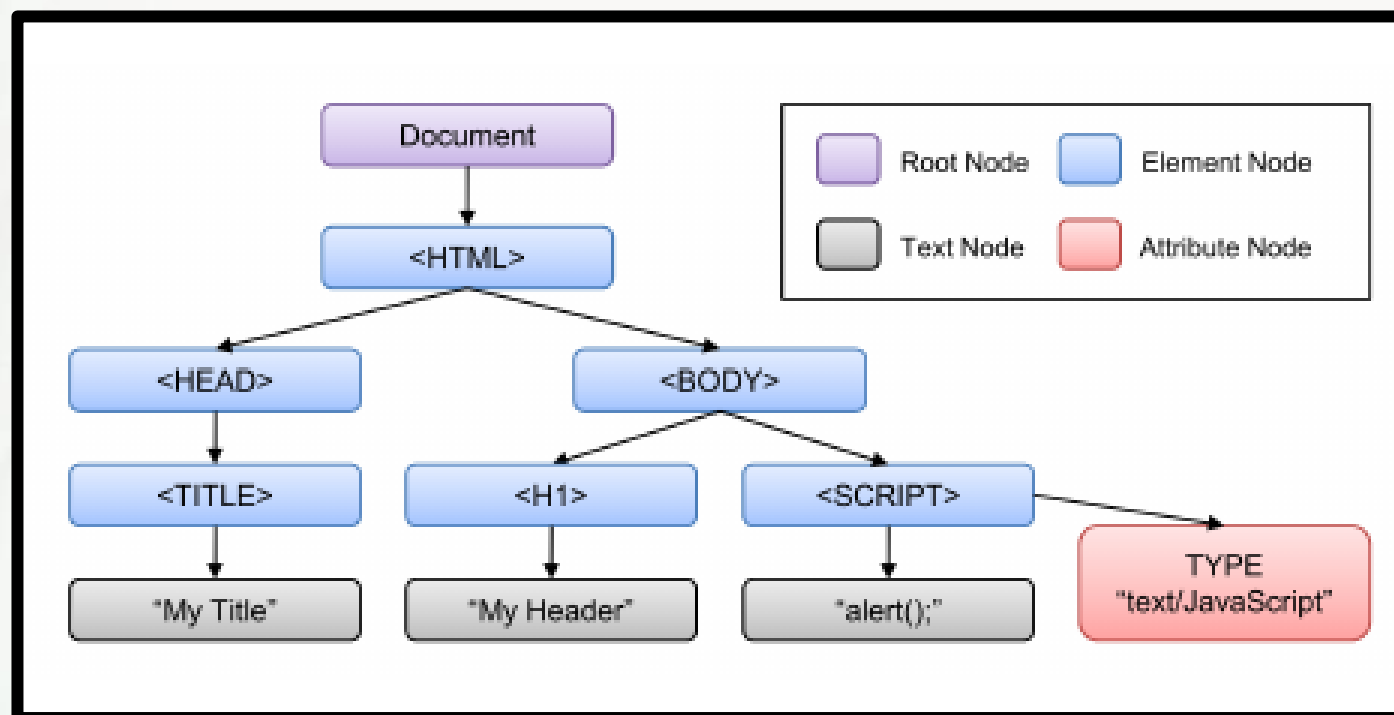
## DOM: Node x Element

- Cada um dos componentes do document é denominado **node** (nó), isso inclui comentários, textos, tags;
- Aqueles que são tags, por sua vez, são denominados **HTML elements**, isso inclui <html>, <body>, e afins.
- Todo **HTML element** é um **node**, mas o contrário não é verdade. Isso implica que **as propriedades** de um node valem para um HTML element, mas o contrário não.



# DOM: Node

Abaixo alguns exemplos de nodes, tendo em vista a visão geral do DOM:



Fonte: <https://medium.com/@ralph1786/accessing-dom-elements-with-javascript-7962f73c59be>



# DOM: Node

Caso queira verificar o tipo de nó, basta utilizar o comando `nodeType` que é possível verificar, considerando a seguinte tabela:

Type	Description	Children
1 Element	Represents an element	Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
2 Attr	Represents an attribute	Text, EntityReference
3 Text	Represents textual content in an element or attribute	None
4 CDATASection	Represents a CDATA section in a document (text that will NOT be parsed by a parser)	None
5 EntityReference	Represents an entity reference	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
6 Entity	Represents an entity	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
7 ProcessingInstruction	Represents a processing instruction	None
8 Comment	Represents a comment	None
9 Document	Represents the entire document (the root-node of the DOM tree)	Element, ProcessingInstruction, Comment, DocumentType
10 DocumentType	Provides an interface to the entities defined for the document	None
11 DocumentFragment	Represents a "lightweight" Document object, which can hold a portion of a document	Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
12 Notation	Represents a notation declared in the DTD	None

Fonte: [https://www.w3schools.com/jsref/prop\\_node\\_nodetype.asp](https://www.w3schools.com/jsref/prop_node_nodetype.asp)

As propriedades e métodos dos **nodes** podem ser acessadas em <https://developer.mozilla.org/en-US/docs/Web/API/Node>. É importante dizer que cada um dos tipos de nós possuem seus respectivos conjuntos de propriedades e métodos.

# DOM: Acessando os elementos

# DOM: métodos

Podemos acessar os elementos do DOM por meio de diversos **métodos** do *node* document:

- **document.getElementById(id):** seleciona o primeiro elemento com o determinado ID;
- **document.getElementsByClassName(className):** seleciona todos os elementos que contem a determinada classe;
- **document.getElementsByTagName(tag):** seleciona todos os elementos correspondente a uma determinada tag;
- **document.querySelector(seletor):** seleciona o primeiro elemento correspondente ao seletor determinado;
- **document.querySelectorAll(seletor):** seleciona todos os elementos correspondentes ao seletor determinado;

## DOM: getElementById – exemplo

```
// titulo principal capturado
console.log("Titulo principal capturado");
console.log(document.getElementById("titulo-principal"));

// primeiro paragrafo capturado
console.log("Primeiro parágrafo capturado");
console.log(document.getElementById("p1"));

// link capturado
console.log("Link capturado");
console.log(document.getElementById("link"));
```

## DOM: getElementsByClassName – exemplo

```
// classe txt capturada - retorna colecao de objetos
console.log("Classe txt capturada");
console.log(document.getElementsByClassName("txt"));

// elementos com classe txt E white capturados - retorna colecao de objetos
console.log("Classe txt E white capturadas");
console.log(document.getElementsByClassName("txt white"));

// id p1 capturado, dentro dele capturou aquele elemento que tem classe destaque
console.log("classe destaque capturada");
console.log(document.getElementById("p1").getElementsByClassName("destaque"));
```



## DOM: getElementByTagName – exemplo

```
// body capturado
console.log("Body capturado");
console.log(document.getElementsByTagName("body"));

// h2 capturados
console.log("h2 capturados");
console.log(document.getElementsByTagName("h2"));

// p capturados
console.log("p capturados");
console.log(document.getElementsByTagName("p"));
```

## DOM: querySelector – exemplo

```
// body capturado - seletor de elemento
console.log("body capturado");
console.log(document.querySelector("body"));

// id p1 capturado - seletor de id (retorna apenas o primeiro)
console.log("id p1 capturado");
console.log(document.querySelector("#p1"));

// classe txt capturada - seletor de classe (retorna apenas o primeiro)
console.log("um elemento com classe txt capturado");
console.log(document.querySelector(".txt"));

// classe txt e white capturadas - seletor de classe (retorna apenas o primeiro)
console.log("um elemento com classe txt e white capturado");
console.log(document.querySelector(".txt.white"));

// id link dentro de classe txt capturado - seletor de classe (retorna apenas o primeiro)
console.log("um elemento id link filho direto de .txt capturado");
console.log(document.querySelector(".txt>#link"));
```

## DOM: querySelectorAll – exemplo

```
// classe txt capturada - seletor de classe
console.log("todos .txt capturados");
console.log(document.querySelectorAll(".txt"));

// classe destaque capturada - seletor de classe
console.log("todos .destaque descendentes de .txt capturados");
console.log(document.querySelectorAll(".txt .destaque"));

// classes heading e txt capturadas - seletor de classe
console.log("todos .txt e .heading capturados");
console.log(document.querySelectorAll(".heading, .txt"));
```

# DOM: manipulando conteúdo

# DOM: acessando o conteúdo

Podemos acessar os conteúdos dos elementos do DOM por meio de diversos métodos:

- **textContent:** todo o texto contido por um elemento e seus descendentes.  
**OBS: propriedade de um node;**
- **innerHTML:** todo o conteúdo do elemento HTML, permitindo a inserção de tags via JavaScript; (**OBS: Não confundir com . textContent() que só permite alteração no conteúdo textual**)
- **innerText:** todo o texto contido por um elemento e seus descendentes; **OBS: definido apenas para *HTML element objects***
- **value:** todo o conteúdo contido em um text field;

## DOM: textContent – acessando

```
// acessando conteúdo textual (getElementById)
console.log("capturando o box3");
console.log(document.getElementById("box3"));
console.log("conteúdo textual do elemento");
console.log(document.getElementById("box3").textContent);
```

```
console.log("capturando os elementos h2");
const elementos = document.getElementsByTagName("h2"); // armazenando os elementos
console.log(elementos);

// acessando o conteúdo dos elementos
for (let i = 0; i < elementos.length; i++) {
  console.log("Elemento " + i + " do vetor");
  console.log(elementos[i].textContent);
}
```



## DOM: textContent – alterando

```
// alterando o conteudo das caixas verdes
const verdes = document.querySelectorAll(".cor2");
for (let i = 0; i < verdes.length; i++) {
  verdes[i].textContent = "Verde";
}
```

```
// alterando o conteudo dos retangulos brancos
const brancos = document.querySelectorAll(".cor1");
for (let i = 0; i < brancos.length; i++) {
  brancos[i].textContent = "Branco";
}
```

## DOM: innerHTML – acessando e alterando

```
// acessando e alterando o paragrafo #p2
console.log("acessando o HTML do elemento de id p2");
console.log(document.querySelector("#p2").innerHTML);
document.querySelector("#p2").innerHTML =
  "Este é um texto adicionado via JS com o <strong class='destaque'>innerHTML</strong>";
```

Este é um texto adicionado via JS com o **innerHTML**

## DOM: innerText – acessando e alterando

```
// acessando e alterando o paragrafo #p2  
console.log("acessando o HTML do elemento de id p2");  
document.getElementById("p1").innerText =  
    "Este texto foi escrito via JavaScript com o textContent";  
// -----
```

## DOM: innerText – acessando e alterando

```
// acessando campo de texto
// inicialmente vazio
console.log("acessando o valor de um campo de texto");
console.log(document.querySelector(".campo-texto").value);

// alterando o valor de um input
console.log("alterando o valor de um campo de texto");
document.querySelector(".campo-texto").value = "Nome Completo";
// -----
```

# DOM: manipulando atributos

Uma vez que conseguimos acessar os elementos do DOM e seus respectivos conteúdos, podemos acessar, alterar e remover atributos das tags. Para isso utilizamos:

- **getAttribute("<nome\_propriedade>")**: acessa o valor da propriedade presente na tag;
- **setAttribute("<nome\_propriedade>", "<valor>")**: altera o valor de uma propriedade para uma determinada tag;
- **removeAttribute("<nome\_propriedade>")**: remove um determinado atributo da tag

## Exemplo

```
// acessando o valor do href
console.log("acessando o atributo da tag");
console.log(document.querySelector(".link").getAttribute("href"));

// alterando o valor do href
console.log("alterando o atributo href da tag");
console.log(
  document
    .querySelector(".link")
    .setAttribute("href", "https://developer.mozilla.org/pt-BR/")
);

// criando atributo target e atribuindo um valor
console.log("criando o atributo target da tag");
console.log(document.querySelector(".link").setAttribute("target", "_blank"));

// removendo um atributo
document.querySelector(".link").removeAttribute("target");
```

## DOM: `classList`, `className` e `style`

Algumas propriedades dos *Elements* podem ser utilizadas para a manipulação de classes e estilo-inline:

- **classList:** Propriedade que apresenta o conjunto de classes de um determinado elemento.
  - **.remove():** remove uma determinada classe da lista;
  - **.add():** adiciona uma determinada classe na lista;
- **style:** Propriedade que contém todas as listas de propriedades que um elemento pode ter. Para acessar as propriedades deve-se utilizar o nome delas do CSS considerando o *CamelCase*:
  - **backgroundColor;**
  - **color;**
- **className:** Propriedade que permite a inserção de uma classe eliminando todas as outras;



## DOM: classList, className e style

```
// acessando as classes de um elemento
const caixa2 = document.querySelector("#box2");
console.log(caixa2.classList);

// removendo a classe de um elemento
caixa2.classList.remove("cor2");

// adicionando uma classe ao elemento
caixa2.classList.add("cor2");

// atribuindo um estilo (style)
document.querySelector("#box2").style.backgroundColor = "#426dff";
document.querySelector("#box2").textContent = "Azul";

// adicionando uma classe com className
// ele retira todas as outras
console.log(document.querySelector("#box3").classList);
document.querySelector("#box3").classList.remove("ret");
document.querySelector("#box3").className = "ret";
```

# Executando o JS sem o HTML

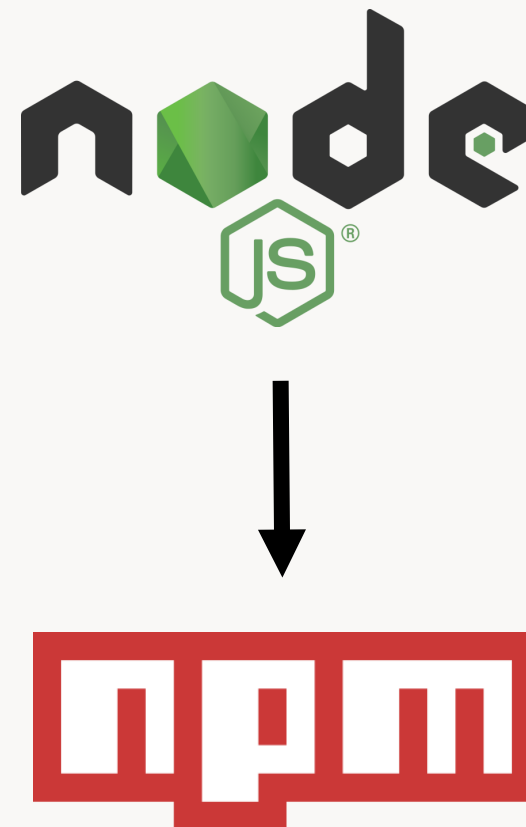
# Node.js

- Para rodar os scripts fora do navegador, sem estar vinculado ao HTML é necessário a instalação do Node.js
- Software livre baseado no interpretador, ou engine, v8 utilizado pelo Google Chrome;
- Permite escrever e 'rodar' aplicações javascript no servidor;
- Permite escrever ferramentas para auxiliar o desenvolvimento web local



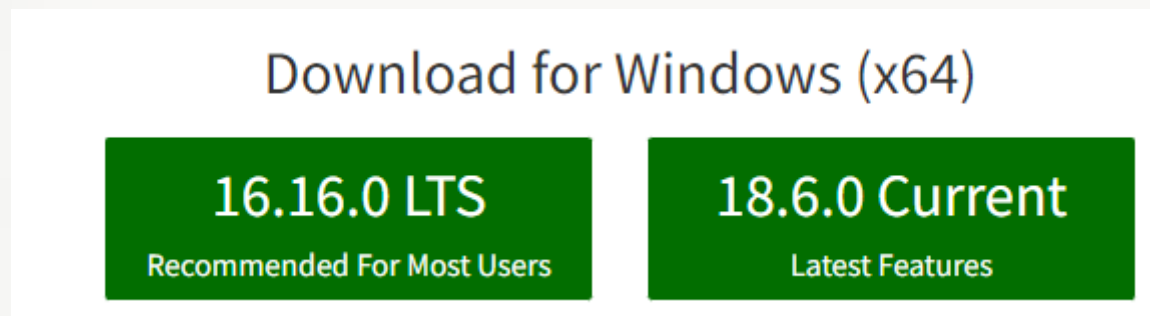
## Node.js: npm e npx

- **npm:** refere-se ao **N**ode **P**ackage **M**anager, que é um **gerenciador de pacotes** do Node. É uma interface por linha de comando (CLI) que permite a instalação e manuseio de pacotes;
- **npx:** refere-se ao **N**ode **P**ackage **E**Xecute, que é um executor de pacotes Node.js. A instalação do npm já garante a instalação do npx.



# Node.js: Como instalar?

Para realizar a instalação do Node.js com o npm basta acessar <https://nodejs.org/> e selecionar a opção 16.16.0 LTS:



Após a instalação verifique a versão dos softwares instalados com os comandos abaixo:

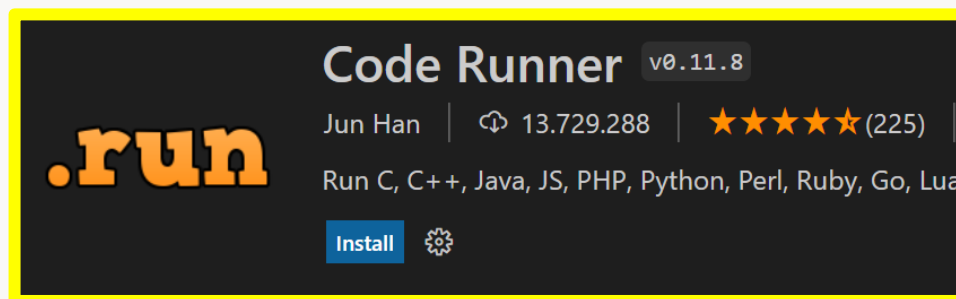
```
C:\Users\Pichau\OneDrive\Área de Trabalho>node -v
v16.13.2

C:\Users\Pichau\OneDrive\Área de Trabalho>npm -v
8.1.2

C:\Users\Pichau\OneDrive\Área de Trabalho>npx -v
8.1.2
```

# Node.js: Configurando o VS CODE

Para que seja possível rodar os scripts via VS Code, após a instalação do Node.js deve-se instalar a extensão **Code Runner**:





# Node.js: Rodando um script

Após a instalação da extensão basta criar um arquivo .js de preferência e, neste arquivo, apertar “**CTRL + ALT + N**” para a execução do código:

## Código

```
JS script-puro.js
1  console.log("Hello World");
2
```

## Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
[Running] node "c:\Users\Pichau\OneDrive\Área de Trabalho\script-puro.js"
Hello World

[Done] exited with code=0 in 0.157 seconds
```

# Live Server com Node.js

# Node.js: instalando o live server

- Após a instalação do Node.js, podemos realizar a instalação do *live server*;
- Para isso utiliza-se o gerenciador de pacotes npm com o seguinte comando no terminal “**npm install live-server -g**”;
- -g especifica que o live server deverá ser instalado globalmente;
- Com o comando live-server a página é carregada;

## Terminal

```
p\Js Web> npm install live-server -g
```

## Terminal

```
p\Js Web> live-server
```

## Node.js: instalando o live server

Caso haja um erro de autorização, a politica de execução deve ser alterada acessando o Power Shell como admin, definindo o seguinte comando “**Set-ExecutionPolicy Unrestricted**”:

**Power Shell (admin)**

```
PS C:\Windows\system32> Set-ExecutionPolicy Unrestricted
```