**O que é o DOM ?**

**DOM** é um acrônimo para Document Object Model (Modelo de Objetos de Documento) ele representa os objetos de uma página HTML em forma de uma árvore. O **DOM** é importante pois é através dele que acessamos os elementos dentro de uma página HTML. Em outra palavras, obtemos acesso ao documento HTML através do **DOM**.

Quando um documento HTML é baixado no navegador, esse navegador precisa transformar em uma página Web o que é basicamente uma longa string de caracteres. Para tanto, o navegador decide quais partes são parágrafos, quais são cabeçalhos, quais são texto e assim por diante. Para evitar que os pobres programadores de JavaScript tenham de fazer exatamente o mesmo trabalho, o navegador armazena sua interpretação do código HTML como um estrutura de objetos, chamada **Document Object Model** ou **DOM**. Como resultado da maneira pela qual o código HTML é escrito - como uma hierarquia de elementos aninhados, marcados com tags de início e fim - o **DOM** cria um objeto diferente para cada elemento, mas vincula cada objeto de elemento ao seu elemento circundante (ou elemento pai). Isso cria uma relação pai-filho (parent-children) explícita entre os elementos e torna mais imediata a visualização do **DOM** como uma estrutura em árvore.

## O objeto window

O **objeto Window** é o principal ponto de entrada para todos os recursos e APIs de Javascript do lado do cliente. Ele representa uma janela ou quadro de navegador Web e pode ser referenciado através do identificador window. Em Javascript do lado do cliente, o objeto Window também é o objeto Global. Isso significa que o objeto Window está no topo do encadeamento de escopo e que suas propriedades e métodos são efetivamente variáveis globais e funções globais.

# Document

Para cada página carregada no browser, existe um objeto **Document**. A interface Document serve como um ponto de entrada para o conteúdo da Página ( a árvore DOM, incluindo elementos como [<body>](https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/body) e [<header>](https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/table)) e provê funcionalidades globais ao documento (como obter a URL da página e criar novos elementos no documento).

Um objeto document pode ser obtido por meio de várias APIs:

* A mais comum; você trabalha com o script do documento que está rodando o script com o objeto document. (O mesmo document também pode ser referenciado como [window.document](https://developer.mozilla.org/pt-BR/docs/Web/API/Window/document).)
* O documento de um iframe através da propriedade [contentDocument](https://developer.mozilla.org/en-US/docs/Web/API/HTMLIFrameElement#properties).
* O [responseXML de um objeto XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest#responsexml).
* O documento ao qual um determinado node ou elemento pertence pode ser recuperado usando a propriedade do node [ownerDocument (en-US)](https://developer.mozilla.org/en-US/docs/Web/API/Node/ownerDocument)

# Element

A interface **Element** é a classe base mais geral da qual todos os objetos em um [Document](https://developer.mozilla.org/pt-BR/docs/Web/API/Document) herda. Ela somente tem métodos e propriedades comuns para todos os tipos de elementos. Mais classes específicas herdam de Element. Por exemplo, a interface [HTMLElement](https://developer.mozilla.org/pt-BR/docs/Web/API/HTMLElement) é a interface base para elementos HTML, enquanto a interface [SVGElement (en-US)](https://developer.mozilla.org/en-US/docs/Web/API/SVGElement) é a base para todos os elementos SVG. A maioria das funcionalidades é especificada mais abaixo da hierarquia de classes.

O innerHTML pode ser usado para inserção de tags, textos e imagens em uma página web, gerando um risco de segurança. Parecido com um cross-site scripting, mas inofensivo, pois o HTML5 especifica que uma tag <script> inserida via innerHTML em uma página web não deve ser executada. Entretanto, existem formas de executar JavaScript sem usar <script>, portanto, ainda há um risco de segurança ao utilizar o innerHTML. Por esta razão, recomenda-se não utilizar o innerHTML para inserção de texto puro.

#### **innerText:**

Já o innerText retorna apenas o texto. Ele ignora todas as tags HTML que estão dentro do elemento, mas ainda assim, “entende” o css do elemento, retornando apenas textos visíveis. Utiliza-se quando queremos buscar apenas o texto visível disponível no elemento.

#### **textContent:**

O textContent funciona de forma muito semelhante ao innerText, porém, retornando todo o conteúdo de texto, incluindo o texto oculto pelo css, bem como as quebras de linha (\n). Utilizamos o textContent quando queremos buscar todo o texto disponível no elemento.

NodeListpode parecer um array, mas, na realidade, ambos são duas coisas completamente diferentes. Um NodeListobjeto é basicamente uma coleção de nós [DOM](https://attacomsian.com/blog/getting-dom-elements-javascript) extraídos do documento HTML. Um array é um tipo de dado especial em JavaScript, que pode armazenar uma coleção de elementos arbitrários.

A função createElement() irá criar um elemento HTML para ser, posteriormente, inserido em um documento HTML.

Veja a sintaxe da função:

var element = document.createElement(tagName);

tagName é uma string que especifica o tipo do elemento, é uma tag HTML.

element é o objeto criado e retornado pela função

Aparentemente, a função não surte efeito algum, precisamos aplicar o método appendChild() para que o elemento seja, efetivamente, inserido ao documento HTML e fique visível ao usuário. Além disso, precisamos definir (buscar) um elemento pai, ou seja, o elemento HTML que receberá nosso elemento (recém criado) como filho.

// Buscar elemento pai

var elemento\_pai = document.body;

// Criar elemento

var titulo = document.createElement('h1');

// Inserir (anexar) o elemento filho (titulo) ao elemento pai (body)

elemento\_pai.appendChild(titulo);