





BRUNA VIEIRA

Desenvolvedora

Experiência com Front-End e Back-end.
Principais linguagens: JavaScript, Angular,
React, Java.

<h1>

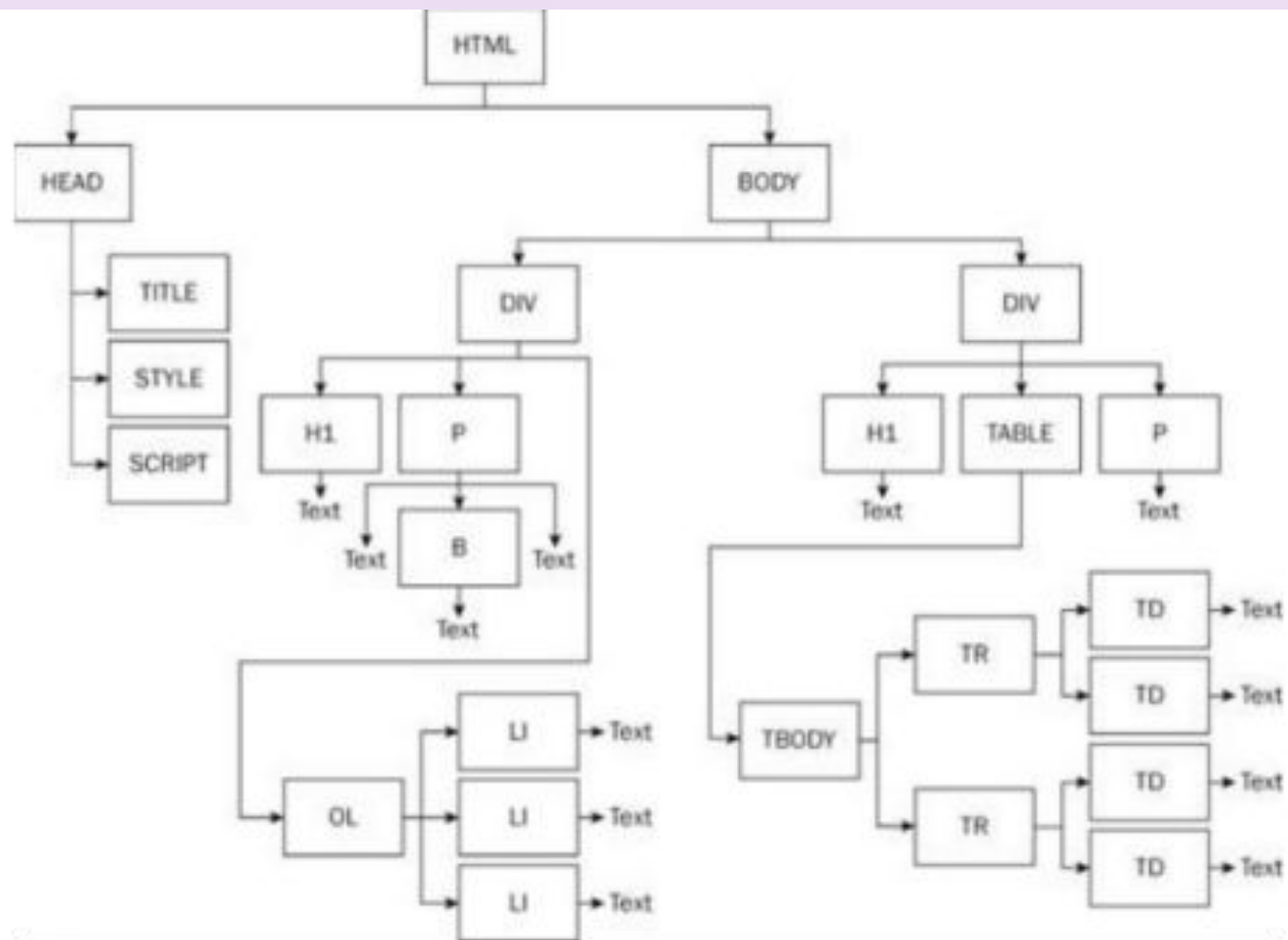
DOM - Document Object Model Com JavaScript

</h1>

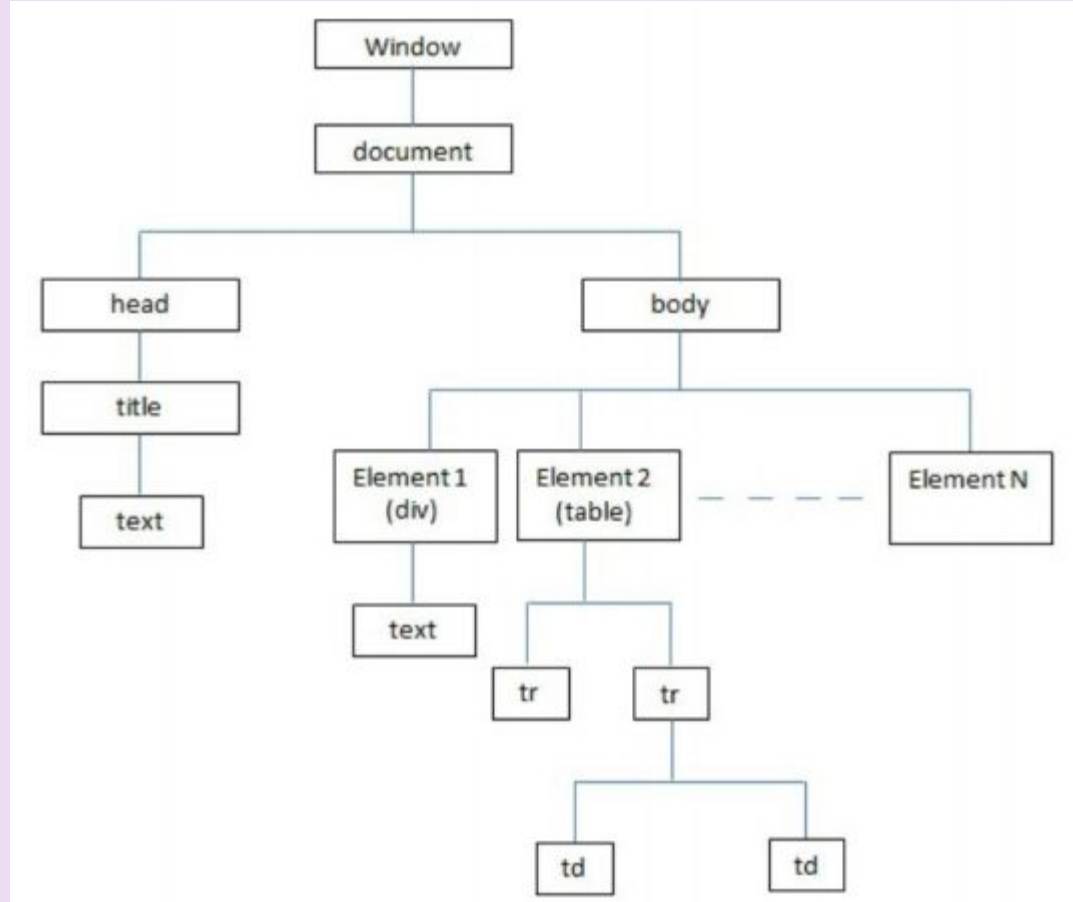
<html>

- DOM – Document Object Model ou Modelo de Objeto de Documento, é uma interface que representa o documento HTML da página web.
- É uma interface de programação para manipulação de documentos HTML, XML e SVG e está disponível para várias linguagens.
- O DOM fornece essa interface através de nós e objetos para que possamos manipular toda nossa página através de métodos e propriedades.

</html>



- A hierarquia começa pelo objeto window:
- window.document.body
OU APENAS
- document.body



Os principais métodos para o entendimento na criação e manipulação de elementos HTML através do DOM:

- `getElementById()`
- `getElementsByClassName()`
- `getElementsByTagName()`
- `querySelector()`
- `querySelectorAll()`
- `createElement()`
- `appendChild()`
- `removeChild()`
- `parentNode()`

- Lembrando que JavaScript é case sensitive isso quer dizer que os métodos do DOM são escritos dessa forma: `getElementById` e não `getelementbyid`.



- Os nomes desses métodos foram assinados com o padrão Camel Case a partir da segunda palavra.

- **getElementById():** pega um elemento através do id e retorna o seu objeto.

```
<form class="form">
  <label for="name">Nome</label>
  <input type="text" id="nome" value="Bruna">
  <label for="email">Nome</label>
  <input type="text" id="email" value="bruna.vieira.t@hotmail.com">
</form>

</body>

<script>
  var elemento = document.getElementById('email');
  console.log(elemento);
  console.log(elemento.value);
</script>
```

```
<input type="text" id="email" value="bruna.vieira.t@hotmail.com">
```

bruna.vieira.t@hotmail.com

- **getElementsByClassName():** Seleciona uma coleção de elementos através do nome da classe. Perceba o plural **"Elements"**

```
<body>
  <ul>
    <li class="test">item 1</li>
    <li class="test">item 2</li>
    <li class="test">item 3</li>
    <li class="test">item 4</li>
  </ul>
</body>

<script>
  var itens = document.getElementsByClassName('test');
  console.log(itens);
  console.log(itens[1]);
</script>
```

```
▼ HTMLCollection(4) ⓘ
  ▶ 0: li.test
  ▶ 1: li.test
  ▶ 2: li.test
  ▶ 3: li.test
  length: 4
  ▶ __proto__: HTMLCollection

  <li class="test">item 2</li>
```

- **getElementsByTagName():** retorna os elementos, ou seja, uma coleção, através do nome de uma tag:

```
<form class="form">
  <label for="name">Nome</label>
  <input type="text" id="nome" value="Bruna">
  <label for="email">Nome</label>
  <input type="text" id="email" value="bruna.vieira.t@hotmail.com">
</form>
</body>

<script>
  var elemento = document.getElementsByTagName('input');
  console.log(elemento);
</script>
```

```
▼ HTMLCollection(2) ⓘ
  ▶ 0: input#nome
  ▶ 1: input#email
    length: 2
  ▶ email: input#email
  ▶ nome: input#nome
  ▶ __proto__: HTMLCollection
```

```
var elementos = document.getElementsByTagName('input');

for(var pos = 0; pos < elementos.length; pos++){
    console.log("valor do input " + elementos[pos].id.toUpperCase());
    console.log(elementos[pos].value);
}
```

valor do input NOME

Bruna

valor do input EMAIL

bruna.vieira.t@hotmail.com

- **querySelector():** retorna um elemento, da mesma forma que o `getElementById()` porém você utiliza um seletor CSS ao invés do nome do ID.

```
var inputLastChild = document.querySelector('input:last-child');  
console.log('last child ' + inputLastChild);  
  
<input class="form__input" type="text" id="email" value="bruna.vieira.t@hotmail.com">
```

```
var formInput = document.querySelector('.form__input');  
console.log(formInput);  
  
<input class="form__input" type="text" id="nome" value="Bruna">
```

```
var formInputLastChild = document.querySelector('.form__input:last-child');  
console.log(formInputLastChild);  
  
<input class="form__input" type="text" id="email" value="bruna.vieira.t@hotmail.com">
```

- **querySelectorAll()**: funciona da mesma forma que o **querySelector** porém retorna um **Array NodeList**.

```
<script>  
  var elementos = document.querySelectorAll('.form__input');  
  console.log(elementos);  
</script>
```

```
querySe  
▼ NodeList(2) [input#nome.form__input,  
  input#email.form__input] ⓘ  
  ▶ 0: input#nome.form__input  
  ▶ 1: input#email.form__input  
    length: 2  
  ▶ __proto__: NodeList
```

createElement e appendChild

- Esses dois métodos da interface DOM servem para criar elementos e adicioná-los ao DOM.
- É como se fossemos utilizar o innerHTML, porém aqui a ideia é criar cada elemento como um objeto separadamente.
- Imagine que você precisa criar um elemento div qualquer e adicioná-lo em sua página. Com innerHTML fizemos algo como na imagem abaixo:

```
var div = '<div></div>';  
document.body.innerHTML = div;
```


- Com create element você cria um fragmento do elemento em memória e logo em seguida utiliza o método appendChild para inserir o elemento de fato no HTML. Exemplo:

```
var div = document.createElement('div');  
document.body.appendChild(div);
```


A photograph of a group of students sitting at desks in a classroom or computer lab, viewed from the side. The image is dimly lit with a purple tint. The students are looking towards the right side of the frame. A text overlay is centered over the image.

Vamos aos exercícios!

A large group of young women, likely a school or club group, are posing for a photo in a hallway. They are arranged in several rows, with some standing and some sitting or kneeling in the front. Many of the women are smiling and making hand gestures, such as peace signs and thumbs up. The image has a purple tint. Overlaid on the image is the text "<the end>" in a light purple font at the top left, "Thanks \o/" in a large white font in the center, and "</the end>" in a light purple font at the bottom left.

<the end>

Thanks \o/

</the end>