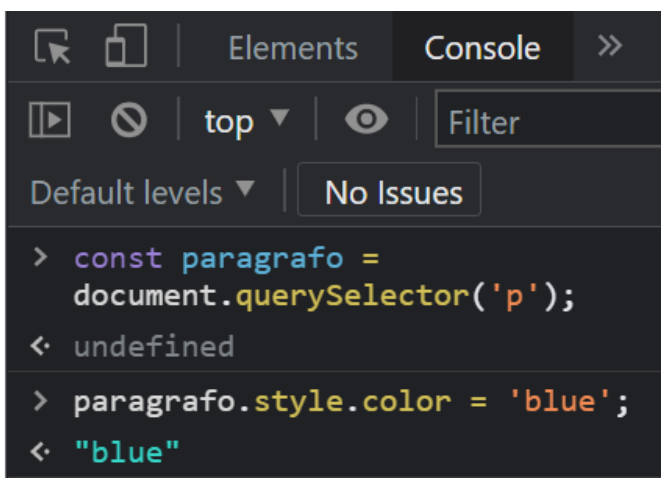
🎖️

Acessando HTML via JavaScript



Acessando a DOM pelo JS

Acessei esse parágrafo

> Para manipular a DOM na maioria das vezes precisamos buscar elementos que estão nela, e para isso, podemos utilizar alguns métodos.

O que é DOM?

Segundo a documentação MDN, o DOM é uma representação orientada a objetos da página da web, que pode ser modificada com uma linguagem de script como JavaScript.

DOM significa 'Document Object Model', e é na verdade a representação da sua página web usada pelo navegador. Podemos alterar sua estrutura, seu estilo e outros aspectos em geral.

Possuímos diversos meios de acessarmos a DOM para criarmos e atribuirmos funcionalidades à nossa página web, como alertas, mensagens, interações, obter dados e etc. Os principais são:

- querySelector;
- getElementById;
- querySelectorAll;

---

querySelector →

Retorna o primeiro elemento que corresponde ao seletor passado na busca. Podemos buscar elementos com base:

- no seu id:

```
<!-- HTML: -->

<p id="idDoParagrafo">
    Parágrafo de exemplo
</p>
```

```
// Javascript:

const paragrafo = document.querySelector('#idDoParagrafo');
```

Perceba que para selecionarmos determinado ID usamos o '#' antes de seu nome, assim como precisaremos usar o '.' no caso de seleção por classe. Um bom jeito de lembrarmos é se comparar ao css, onde selecionamos ID de elementos com # e classes com . .

- na sua classe:

```
<!-- HTML: -->

<p class="classDoParagrafo">
```

```
    Outro parágrafo de exemplo
</p>
```

```
// Javascript:

const paragrafo2 = document.querySelector('.classDoParagrafo');
```

Assim como selecionamos a DOM com base na classe do elemento podemos usar seu id como já dito. Lembre que devemos utilizar o '.' antes de inserirmos o nome da classe em questão.

- no nome da tag do elemento:

```
<!-- HTML: -->

<div>
  <p id="idDoParagrafo">
      Parágrafo de exemplo
  </p>
  <p class="classDoParagrafo">
      Outro parágrafo de exemplo
  </p>
</div>
```

```
// Javascript:

// Por ID:
const paragrafo = document.querySelector('#idDoParagrafo');

// Por CLASSE:
const paragrafo2 = document.querySelector('.classDoParagrafo');

// Por TAG:
const paragrafo3 = document.querySelector('p');
paragrafo3.style.color = 'red';
```

Sempre que usamos a seleção por nome, nos será retornado o primeiro elemento correspondente àquela tag. Assim, no exemplo acima, só será mudado a cor da fonte do primeiro parágrafo.

Diferentemente dos outro métodos, esse não há nenhuma necessidade de colocarmos # nem . antes da tag do elemento.

Um outro método comum de uso é utilizando id, classe ou tag depois de seus elementos pais. Por exemplo:

```
<!-- HTML: -->
<div>
  <p id="idDoParagrafo">
      Parágrafo de exemplo
  </p>
  <p class="classDoParagrafo">
      Outro parágrafo de exemplo
  </p>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
  </ul>
</div>
```

```
// Javascript:

// Com elemento pai e depois o ID:
const paragrafo = document.querySelector('div #idDoParagrafo');

// Com elemento pai e depois a CLASSE:
const paragrafo2 = document.querySelector('div .classDoParagrafo');

// Com elemento pai e depois a TAG:
const paragrafo3 = document.querySelector('div p');

// Com elemento pai e depois a TAG:
const item1 = document.querySelector('div ul li');
```

getElementById →

Outro método que temos para acessarmos a DOM é o getElementById e, como o próprio nome já sugere, ele busca o elemento por seu ID. Comparado com o querySelector, esse oferece mais performance em grandes aplicações, pois quanto mais exato nossa constatação, mais rápida será a execução.

No método anterior tínhamos a necessidade de inserir o '#' antes de qualquer id, já nesse caso não precisa colocar nada além de seu nome. Porém, esse meio permite a busca seja por ID e somente por ID, ou seja, não conseguiremos acessar tags e/ou classes.

```
<!-- HTML: -->

<p id="idDoParagrafo">
  Parágrafo de exemplo
</p>
```

```
// Javascript:

const paragrafo = document.getElementById('idDoParagrafo');
```

querySelectorAll →

Retorna todos os elementos que condizem com o seletor determinado. Como funcionalidade é igual ao querySelector visto anteriormente, porém, com esse método retorna todos os elementos do HTML são selecionados em uma NodeList (Array de seleções da DOM).

```
<!-- HTML: -->

<div>
  <p id="idDoParagrafo" class="corDeFundo">
    Parágrafo de exemplo
  </p>
  <p class="classDoParagrafo corDeFundo">
    Outro parágrafo de exemplo
  </p>
</div>
```

```
// Javascript:

let paragrafos = document.querySelectorAll('.corDeFundo');
paragrafos[0].style.backgroundColor = "gray";
```

Caso teste o exemplo acima, perceberá que somente o primeiro parágrafo ficou com o fundo cinza. Isso ocorre pois não especificamos que todos os elementos retornados deverão possuir aquele determinado funcionamento ou, no nosso caso, determinado estilo.

Para definirmos todos com estilos iguais, usaremos o chamado laço de repetição (for). Perceba que no exemplo a let corDeFundo é acompanhada de [0], isso significa que a posição dela dentro de um array é igual a 0. Fique tranquilo pois ainda não comentamos muito sobre Arrays, mas é bom para ir aprendendo conforme o tempo.

Com o laço de repetição criado, inserimos determinado estilo. Por exemplo:

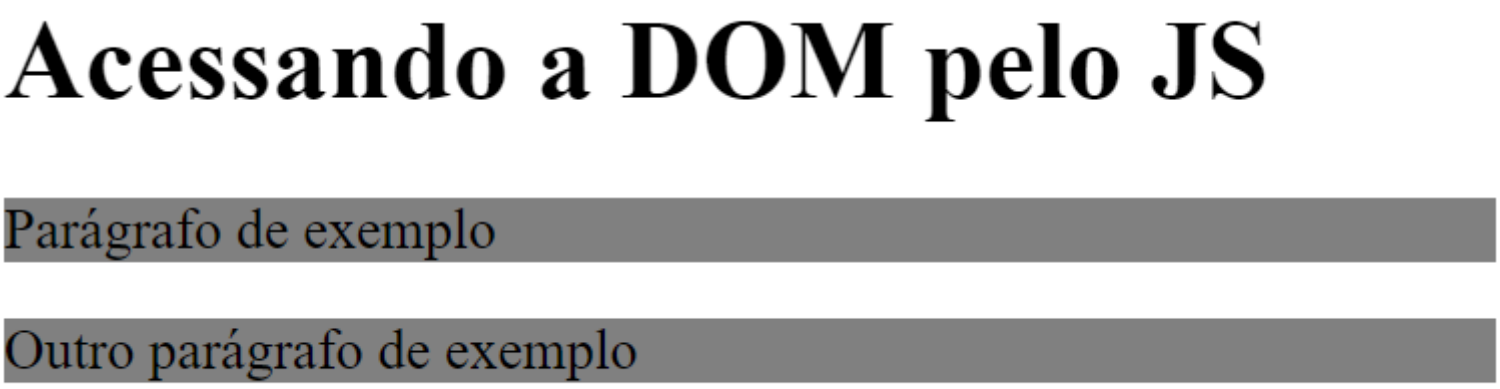
```
// Javascript:

for (i of corDeFundo) {
  i.style.backgroundColor = 'gray'
}
```

Ele define que para todo i de paragrafos, a cor de fundo será cinza. No final, os parágrafos ficam assim:

```
<!-- HTML: -->

<div>
  <p id="idDoParagrafo" class="corDeFundo">
    Parágrafo de exemplo
  </p>
```

```
  <p class="classDoParagrafo corDeFundo">
    Outro parágrafo de exemplo
  </p>
</div>
```

```
// Javascript:

let paragrafos = document.querySelectorAll('.corDeFundo');
paragrafos[0].style.backgroundColor = "gray";

for (i of paragrafos) {
  i.style.backgroundColor = 'gray'
}
```



Formulário →

Quando utilizamos um formulário em nossa página web temos um método mais simples de acesso. Sua sintaxe consiste em:

```
const form = document.forms.nameForm;
```

Desse jeito não precisamos utilizar nenhum dos três métodos aprendidos. Basta usar o `document.forms` e no final colocar o valor do atributo name do formulário desejado. Por exemplo:

```
<!-- HTML: -->

<form name="inscricao">
  <input type="text" name="nome" id="nome">
  <input type="text" name="idade" id="idade">
</form>
```

```
// Javascript:

const form = document.forms.inscricao;
```

Agora que já aprendemos esse novo jeito, podemos utilizá-lo juntamente com outros elementos, como inputs e etc. Por exemplo:

```
const form = document.forms.inscricao;

const { nome } = form;
```

Nesse exemplo, pegamos o formulário vindo do HTML e logo após pegamos o input de dentro do formulário. Basta inserirmos o valor do atributo 'name' do input.

Porém, ao acessarmos por esse meio, ele nos retorna o elemento em si, e não seu valor:

```
const form = document.forms.inscricao;

const { nome } = form;

console.log(nome);
// Retorna: <input type="text" name="nome" id="nome">
```

Então, para pegarmos seu valor podemos colocar o '.value' ao final da const 'nome', ficando parecido com:

```
<!-- HTML: -->

<form name="inscricao">
  <input type="text" name="nome" id="nome" value="Clóvis">
  <input type="text" name="idade" id="idade">
</form>
```

```
// Javascript:

const form = document.forms.inscricao;

const { nome } = form;

console.log(nome.value);
// Retorna: Clóvis
```

Bibliografias:

- https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector;
- https://developer.mozilla.org/en-US/docs/Web/API/Element/querySelectorAll;
- https://developer.mozilla.org/pt-BR/docs/Web/API/Document/getElementById;
- https://developer.mozilla.org/en-US/docs/Web/API/Document/forms;