


# DOM

## DOCUMENT OBJECT MODEL (DOM)

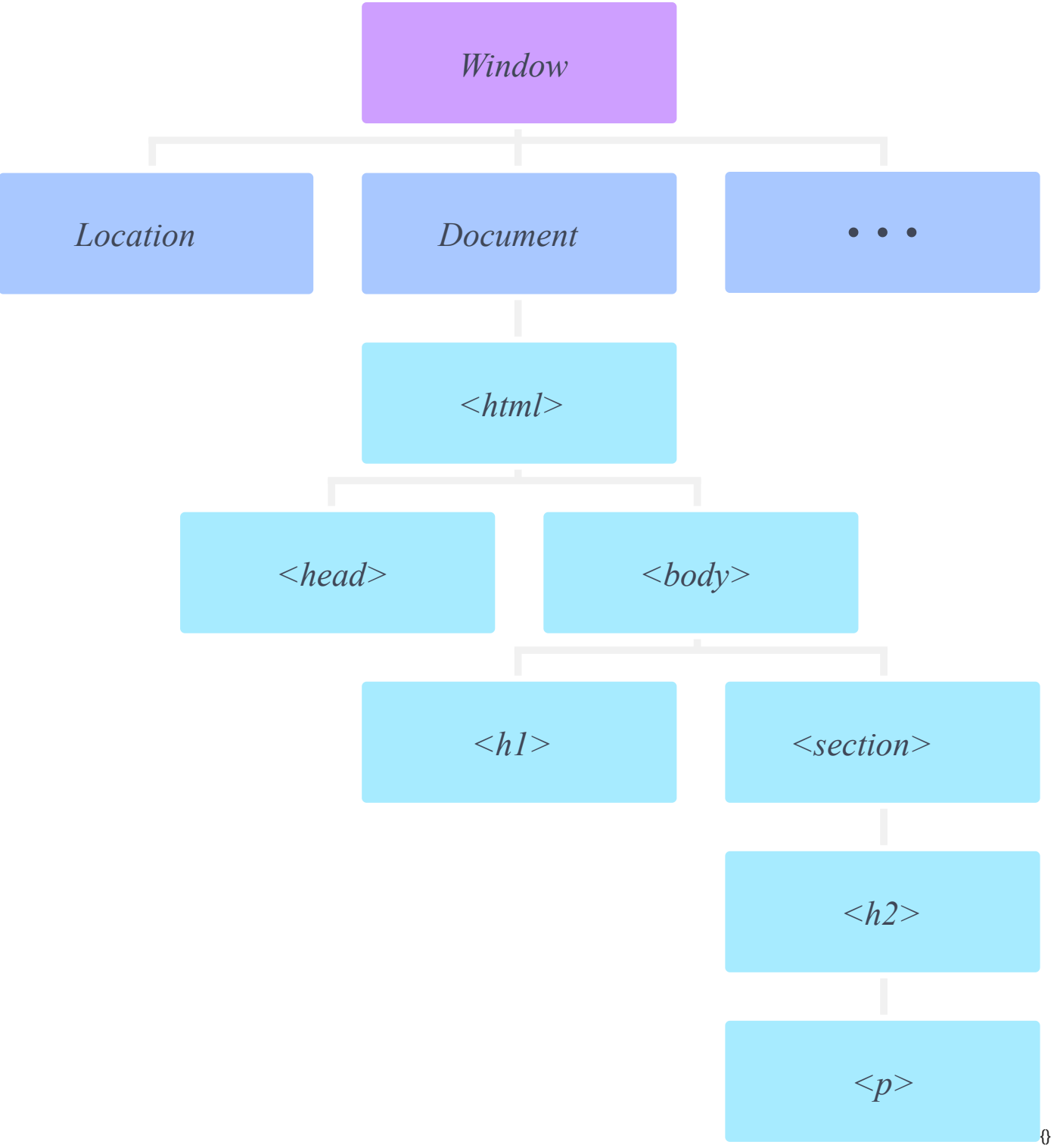
É uma interface que representa documentos HTML e XML através de objetos. Com ela é possível manipular a estrutura, estilo e conteúdo destes documentos.

DOM

```
1 console.log(window);
2 // window é o objeto global do browser
3 // possui diferentes métodos e propriedades
4
5 window.innerHeight; // retorna a altura do browser
```

 DOM

Ao inspecionar elemento com o Chrome, você está vendo a representação oficial do DOM.




## WINDOW E DOCUMENT

São os objetos principais do DOM, boa parte da manipulação é feita através dos seus métodos e propriedades.

windows

```
1 window.alert('Isso é um alerta');
2 alert('Isso é um alerta'); // Funciona
3
4 document.querySelector('h1'); // Seleciona o primeiro h1
5 document.body; // Retorna o body
```

 **window é o objeto global, por isso não precisamos chamar ele na frente dos seus métodos e propriedades.**

## NODE

Toda tag html é representada pelo objeto Element e por isso herda os seus métodos e propriedades. Element é um tipo de objeto Node.

```
node

1  const titulo = document.querySelector('h1');
2
3  titulo.innerText; // retorna o texto;
4  titulo.classList; // retorna as classes;
5  titulo.id; // retorna o id;
6  titulo.offsetHeight; // retorna a altura do elemento;
7
8  titulo.addEventListener('click', callback);
9  // ativa a função callback ao click no titulo
```

## Seleção de Elementos

### ID

**getElementById** seleciona e retorna elementos do DOM

```
id

1  // Seleciona pelo ID
2  const animaisSection = document.getElementById('animais');
3  const contatoSection = document.getElementById('contato');
4
5  // Retorna null caso não exista
6  const naoExiste = document.getElementById('teste');
```

### CLASSE E TAG

**getElementsByClassName** e **getElementsByTagName** selecionam e retornam uma lista de elementos do DOM. A lista retornada está ao vivo, significa que se elementos forem adicionados, ela será automaticamente atualizada.

```
tag

1  // Seleciona pela classe, retorna uma HTMLCollection
2  const gridSection = document.getElementsByClassName('grid-section');
3  const contato = document.getElementsByClassName('grid-section contato');
4
5  // Seleciona todas as UL's, retorna uma HTMLCollection
6  const ul = document.getElementsByTagName('ul');
7
8  // Retorna o primeiro elemento
9  console.log(gridSection[0]);
```

### SELETOR GERAL ÚNICO

**querySelector** retorna o primeiro elemento que combinar com o seu seletor CSS.

```
querySelector

1  const animais = document.querySelector('.animais');
2  const contato = document.querySelector('#contato');
3  const ultimoItem = document.querySelector('.animais-lista li:last-child');
4  const linkCSS = document.querySelector('[href^="https://"]');
5  const primeiroUl = document.querySelector('ul');
6
7  // Busca dentro do Ul apenas
8  const navItem = primeiroUl.querySelector('li');
```

### SELETOR GERAL LISTA

**querySelectorAll** retorna todos os elementos compatíveis com o seletor CSS em uma NodeList

```
querySelectorAll

1  const gridSection = document.querySelectorAll('.grid-section');
2  const listas = document.querySelectorAll('ul');
3  const titulos = document.querySelectorAll('.titulo');
4  const fotosAnimais = document.querySelectorAll('.animais-lista img');
5
6  // Retorna o segundo elemento
7  console.log(gridSection[1]);
```

 Diferente do `getElementsByClassName`, a lista aqui é estática

## HTMLCOLLECTION VS NODELIST

A diferença está nos métodos e propriedades de ambas. Além disso a `NodeList` retornada com `querySelectorAll` é estática.

nodeList


```
1  const titulo = document.querySelector('.titulo');
2  const gridSectionHTML = document.getElementsByClassName('grid-section');
3  const gridSectionNode = document.querySelectorAll('.grid-section');
4
5  titulo.classList.add('grid-section');
6
7  console.log(gridSectionHTML); // 4 itens
8  console.log(gridSectionNode); // 3 itens
```

## ARRAY-LIKE

`HTMLCollection` e `NodeList` são array-like, parecem uma array mas não são. O método de Array `forEach()` por exemplo, existe apenas em `NodeList`.

array-like

```
1  const gridSection = document.querySelectorAll('.grid-section');
2
3  gridSection.forEach(function(gridItem, index, array) {
4    gridItem.classList.add('azul');
5    console.log(index) // index do item na array
6    console.log(array) // a array completa
7  });
```

 É possível transformar array-like em uma Array real

Basta utilizar o método `Array.from(gridSection)"`

### Exercício

- Retorne no console todas as imagens de um site
- Selecione o primeiro h2
- Selecione o último p do site
- Repositório para os exercícios: <https://github.com/gleonel/TWAcode.git>

## FOREACH

Constantemente vamos selecionar uma lista de itens do dom. A melhor forma para interagirmos com os mesmos é utilizando o método `forEach`.

foreach

```
1  const imgs = document.querySelectorAll('img');
2
3  imgs.forEach(function(item){
4    console.log(item);
5  });
```

### PARÂMETROS DO FOREACH

O primeiro parâmetro é o callback, ou seja, a função que será ativada a cada item. Esse função pode receber três parâmetros: `valorAtual`, `index` e `array`;

foreach

```
1  const imgs = document.querySelectorAll('img');
2
3  imgs.forEach(function(valorAtual, index, array){
4    console.log(item); // o item atual no loop
5    console.log(index); // o número do index
6    console.log(array); // a Array completa
7  });
```

## FOREACH E ARRAY

`forEach` é um método de Array, alguns objetos array-like possuem este método. Caso não possua, o ideal é transformá-los em uma array.

array-like

```
1  const titulos = document.getElementsByClassName('titulo');
2  const titulosArray = Array.from(titulos);
3
```

```
4  títulosArray.forEach(function(item){
5      console.log(item);
6  });
```

ARROW FUNCTION

Sintaxe curta em relação a `function expression`. Basta remover a palavra chave function e adicionar a fat arrow `=>` após os argumentos.

```
arrow

1  const imgs = document.querySelectorAll('img');
2
3  imgs.forEach((item) => {
4      console.log(item);
5  });
```

PARÂMETROS E PARÊNTESES

```
1  const imgs = document.querySelectorAll('img');
2
3  // parâmetro único não precisa de parênteses
4  imgs.forEach(item => {
5      console.log(item);
6  });
7
8  // multiplos parâmetros precisam de parênteses
9  imgs.forEach((item, index) => {
10     console.log(item, index);
11 });
12
13 // sem parâmetro precisa dos parênteses, mesmo vazio
14 let i = 0;
15 imgs.forEach(() => {
16     console.log(i++);
17 });
```

RETURN

É possível omitir as chaves {} para uma função que retorna uma linha.

```
=>

1  const imgs = document.querySelectorAll('img');
2
3  imgs.forEach(item =>
4      console.log(item)
5      // Não é permitido fechar a linha com ;
6  );
7
8
9  imgs.forEach(item => console.log(item));
```

Exercício

- Mostre no console cada parágrafo do site
- Mostre o texto dos parágrafos no console
- Como corrigir os erros abaixo:

ex3

```
1  const imgs = document.querySelectorAll('img');
2
3  imgs.forEach(item, index => {
4      console.log(item, index);
5  });
6
7  let i = 0;
8  imgs.forEach( => {
9      console.log(i++);
10 });
11
12 imgs.forEach(() => i++);
```

CLASSLIST

Retorna uma lista com as classes do elemento. Permite adicionar, remover e verificar se contém.

```
classList

1  const menu = document.querySelector('.menu');
2
3  menu.className; // string
```

```
4 menu.classList; // lista de classes
5 menu.classList.add('ativo');
6 menu.classList.add('ativo', 'mobile'); // duas classes
7 menu.classList.remove('ativo');
8 menu.classList.toggle('ativo'); // adiciona/remove a classe
9 menu.classList.contains('ativo'); // true ou false
10 menu.classList.replace('ativo', 'inativo');
```

## ATTRIBUTES

Retorna uma array-like com os atributos do elemento.

```
attributes

1 const animais = document.querySelector('.animais');
2
3 animais.attributes; // retorna todos os atributos
4 animais.attributes[0]; // retorna o primeiro atributo
```

## GETATTRIBUTE E SETATTRIBUTE

Métodos que retornam ou definem de acordo com o atributo selecionado

```
get_set

1 const img = document.querySelector('img');
2
3 img.getAttribute('src'); // valor do src
4 img.setAttribute('alt', 'Texto Alternativo'); // muda o alt
5 img.hasAttribute('id'); // true / false
6 img.removeAttribute('alt'); // remove o alt
7
8 img.hasAttributes(); // true / false se tem algum atributo
```

## READ ONLY VS WRITABLE

Existem propriedades que não permitem a mudança de seus valores, essas são considerados Read Only, ou seja, apenas leitura.

```
read only

1 const animais = document.querySelector('.animais');
2
3 animais.className; // string com o nome das classes
4 animais.className = 'azul'; // substitui completamente a string
5 animais.className += ' vermelho'; // adiciona vermelho à string
6
7 animais.attributes = 'class="ativo"'; // não funciona, read-only
```

Exercício

- Adicione a classe ativo a todos os itens do menu
- Remove a classe ativo de todos os itens do menu e mantenha apenas no primeiro
- Verifique se as imagens possuem o atributo alt
- Modifique o href do link externo no menu

## ADDEVENTLISTENER

Adiciona uma função ao elemento, esta chamada de `callback`, que será ativada assim que certo evento ocorrer neste elemento.

```
event

1 const img = document.querySelector('img');
2
3 // elemento.addEventListener(event, callback, options)
4 img.addEventListener('click', function() {
5   console.log('Clicou');
6 })
```

## CALLBACK

Quando irá reutiliza-la é boa prática separar a função de callback do addEventListener, ou seja, declarar uma função ao invés de passar diretamente uma função anônima

```
callback

1 const img = document.querySelector('img');
2 function callback() {
```

```
3     console.log('Clicou');
4 }
5
6 img.addEventListener('click', callback); // 🚩
7 img.addEventListener('click', callback()); // undefined
8 img.addEventListener('click', function() {
9     console.log('Clicou');
10 });
11 img.addEventListener('click', () => {
12     console.log('Clicou');
13 });
```

EVENT

O primeiro parâmetro do callback é referente ao evento que ocorreu.

event	
1	const img = document.querySelector('img');
2	
3	function callback(event) {
4	console.log(event);
5	}
6	
7	img.addEventListener('click', callback);

Geralmente utilizam e como nome do parâmetro

PROPRIEDADES DO EVENT

title	
1	const animaisLista = document.querySelector('.animais-lista');
2	
3	function executarCallback(event) {
4	const currentTarget = event.currentTarget; // this
5	const target = event.target; // onde o clique ocorreu
6	const type = event.type; // tipo de evento
7	const path = event.path;
8	console.log(currentTarget, target, type, path);
9	}
10	
11	animaisLista.addEventListener('click', executarCallback);

EVENT.PREVENTDEFAULT()

Previne o comportamento padrão do evento no browser. No caso de um link externo, por exemplo, irá prevenir que o link seja ativado.

prevent default	
1	const linkExterno = document.querySelector('a[href^="http"]');
2	
3	function clickNoLink(event) {
4	event.preventDefault();
5	console.log(event.currentTarget.href);
6	}
7	
8	linkExterno.addEventListener('click', clickNoLink);

THIS

A palavra chave `this` é uma palavra especial de JavaScript, que pode fazer referência a diferentes objetos dependendo do contexto. No caso de eventos, ela fará referência ao elemento em que `addEventListener` foi adicionado.


this	
1	const img = document.querySelector('img');
2	
3	function callback(event) {
4	console.log(this); // retorna a imagem
5	console.log(this.getAttribute('src'));
6	}
7	
8	img.addEventListener('click', callback);

DIFERENTES EVENTOS

Existem diversos eventos como `click`, `scroll`, `resize`, `keydown`, `keyup`, `mouseenter` e mais. Eventos podem ser adicionados a diferentes elementos, como o `window` e `document` também.

events	
1	

```
2  const h1 = document.querySelector('h1');
3
4  function callback(event) {
5      console.log(event.type, event);
6  }
7
8  h1.addEventListener('click', callback);
9  h1.addEventListener('mouseenter', callback);
10 window.addEventListener('scroll', callback);
11 window.addEventListener('resize', callback);
    window.addEventListener('keydown', callback);
```

 **Eventos**

<https://developer.mozilla.org/en-US/docs/Web/Events>

KEYBOARD

Você pode adicionar atalhos para facilitar a navegação no seu site, através de eventos do `keyboard`.

keyboard


```
1  function handleKeyboard(event) {
2      if(event.key === 'a')
3          document.body.classList.toggle('azul');
4      else if(event.key === 'v')
5          document.body.classList.toggle('vermelho');
6  }
7
8  window.addEventListener('keydown', handleKeyboard);
```

FOREACH E EVENTOS

O método `addEventListener` é adicionado à um único elemento, então é necessário um loop entre elementos de uma lista, para adicionarmos à cada um deles.

foeach

```
1  const imgs = document.querySelectorAll('img');
2
3  function imgSrc(event) {
4      const src = event.currentTarget.getAttribute('src');
5      console.log(src);
6  }
7
8  imgs.forEach((img) => {
9      img.addEventListener('click', imgSrc);
10 });
```

 **Exercício**

- Quando o usuário clicar nos links internos do site, adicione a classe ativo ao item clicado e remova dos demais itens caso eles possuam a mesma. Previna o comportamento padrão desses links
- Selecione todos os elementos do site começando a partir do body, ao clique mostre exatamente quais elementos estão sendo clicados
- Utilizando o código anterior, ao invés de mostrar no console, remova o elemento que está sendo clicado, o método `remove()` remove um elemento

OUTERHTML, INNERHTML E INNERTEXT

Propriedades que retornam uma string contendo o html ou texto. É possível atribuir um novo valor para as mesmas `element.innerText = 'Novo Texto'`.

text

```
1  const menu = document.querySelector('.menu');
2
3  menu.outerHTML; // todo o html do elemento
4  menu.innerHTML; // html interno
5  menu.innerText; // texto, sem tags
6
7  menu.innerText = '<p>Texto</p>'; // a tag vai como texto
8  menu.innerHTML = '<p>Texto</p>'; // a tag é renderizada
```

TRASVERSING

Como navegar pelo DOM, utilizando suas propriedades e métodos.

transversing

```
1  const lista = document.querySelector('.animais-lista');
2
3  lista.parentElement; // pai
```

```
4 lista.parentElement.parentElement; // pai do pai
5 lista.previousElementSibling; // elemento acima
6 lista.nextElementSibling; // elemento abaixo
7
8 lista.children; // HTMLCollection com os filhos
9 lista.children[0]; // primeiro filho
10 lista.children[--lista.children.length]; // último filho
11
12 lista.querySelectorAll('li'); // todas as LI's
13 lista.querySelector('li:last-child'); // último filho
```

ELEMENT VS NODE

Element's represetam um elemento html, ou seja, uma tag. Node representa um nó, e pode ser um elemento (Element), texto, comentário, quebra de linha e mais.

```
element

1 const lista = document.querySelector('.animais-lista');
2
3 lista.previousElementSibling; // elemento acima
4 lista.previousSibling; // node acima
5
6 lista.firstChild; // primeiro node child
7 lista.childNodes; // todos os node child
```

MANIPULANDO ELEMENTOS

É possível mover elementos no dom com métodos de Node.

```
elements

1 const lista = document.querySelector('.animais-lista');
2 const contato = document.querySelector('.contato');
3 const titulo = contato.querySelector('.titulo');
4
5 contato.appendChild(lista); // move lista para o final de contato
6 contato.insertBefore(lista, titulo); // insere a lista antes de titulo
7 contato.removeChild(titulo); // remove titulo de contato
8 contato.replaceChild(lista, titulo); // substitui titulo por lista
```

NOVOS ELEMENTOS

Podemos criar novos elementos com o método `createElement()`

```
createElement

1 const animais = document.querySelector('.animais');
2
3 const novoH1 = document.createElement('h1');
4 novoH1.innerText = 'Novo Título';
5 novoH1.classList.add('titulo');
6
7 animais.appendChild(novoH1);
```

CLONAR ELEMENTOS

Todo elemento selecionado é único. Para criarmos um novo elemento baseado no anterior, é necessário utilizar o método `cloneNode()`

```
clone

1 const titulo = document.querySelector('h1');
2 const titulo2 = document.querySelector('h1');
3 const novoTitulo = titulo;
4 // titulo, titulo2 e novoTitulo são iguais
5
6 const cloneTitulo = titulo.cloneNode(true); //true sinaliza para incluir os filhos
7 const contato = document.querySelector('.contato');
8 contato.appendChild(cloneTitulo);
```

1

2

Exercício

- Duplicue o menu e adicione ele em copy
- Selecione o primeiro DT da dl de Faq
- Selecione o DD referente ao primeiro DT
- Substitua o conteúdo html de .faq pelo de .animais