

DOM PARA INICIANTES

O que é o DOM

Document Object Model (DOM)

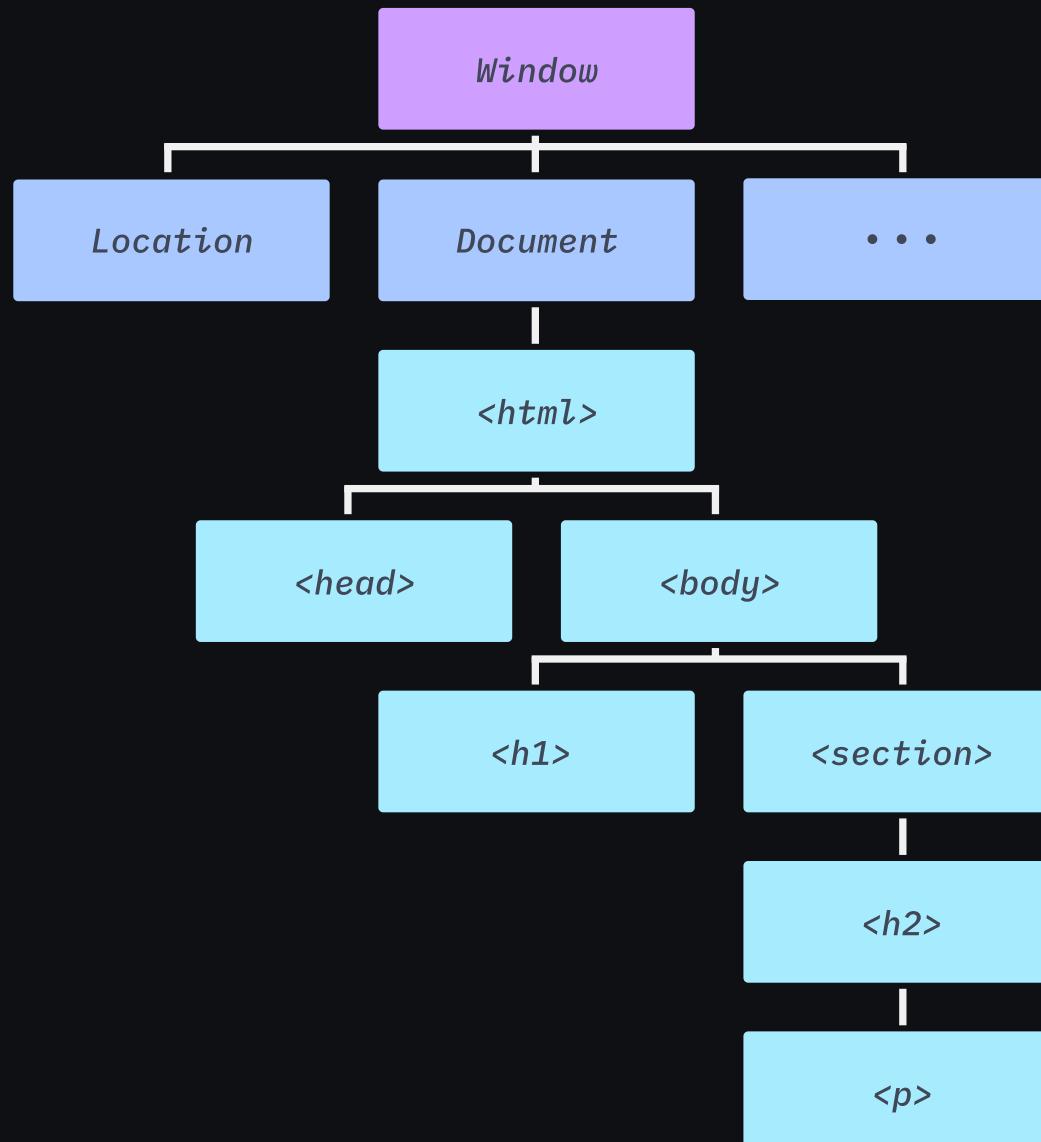
É uma interface que representa documentos HTML e XML através de objetos. Com ela é possível manipular a estrutura, estilo e conteúdo destes documentos.

```
console.log(window);
// window é o objeto global do browser
// possui diferentes métodos e propriedades

window.innerHeight; // retorna a altura do browser
```

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

DOM



Window e Document

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

```
window.alert('Isso é um alerta');
alert('Isso é um alerta'); // Funciona
```

```
document.querySelector('h1'); // Seleciona o primeiro h1
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.

```
const titulo = document.querySelector('h1');

titulo.innerText; // retorna o texto;
titulo.classList; // retorna as classes;
titulo.id; // retorna o id;
titulo.offsetHeight; // retorna a altura do elemento;

titulo.addEventListener('click', callback);
// ativa a função callback ao click no título
```

Exercício

```
// Retorne o url da página atual utilizando o objeto window  
  
// Seleciona o primeiro elemento da página que  
// possua a classe ativo  
  
// Retorne a linguagem do navegador  
  
// Retorne a largura da janela
```

DOM PARA INICIANTES

Seleção de Elementos

ID

`getElementById` seleciona e retorna elementos do DOM

```
// Seleciona pelo ID
const animaisSection = document.getElementById('animais');
const contatoSection = document.getElementById('contato');

// Retorna null caso não exista
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.

```
// Seleciona pela classe, retorna uma HTMLCollection
const gridSection = document.getElementsByClassName('grid-section');
const contato = document.getElementsByClassName('grid-section contact');

// Seleciona todas as UL's, retorna uma HTMLCollection
const ul = document.getElementsByTagName('ul');

// Retorna o primeiro elemento
console.log(gridSection[0]);
```

Seletor Geral Único

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

```
const animais = document.querySelector('.animais');
const contato = document.querySelector('#contato');
const ultimoItem = document.querySelector('.animais-lista li:last');
const linkCSS = document.querySelector('a[href^="https://"]');
const primeiroUl = document.querySelector('ul');

// Busca dentro do Ul apenas
const navItem = primeiroUl.querySelector('li');
```

Seletor Geral Lista

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

```
const gridSection = document.querySelectorAll('.grid-section');
const listas = document.querySelectorAll('ul');
const titulos = document.querySelectorAll('.titulo');
const fotosAnimais = document.querySelectorAll('.animais-lista im')

// Retorna o segundo elemento
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.

```
const titulo = document.querySelector('.titulo');
const gridSectionHTML = document.getElementsByClassName('grid-section')
const gridSectionNode = document.querySelectorAll('.grid-section')

titulo.classList.add('grid-section');

console.log(gridSectionHTML); // 4 itens
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.

```
const gridSection = document.querySelectorAll('.grid-section');

gridSection.forEach(function(gridItem, index, array) {
  gridItem.classList.add('azul');
  console.log(index) // index do item na array
  console.log(array) // a array completa
});
```

É possível transformar array-like em uma Array real, utilizando o método `Array.from(gridSection)`

Exercício

// Retorne no console todas as imagens do site

// Retorne no console apenas as imagens que começaram com a palavra

// Selecione todos os links internos (onde o href começa com #)

// Selecione o primeiro h2 dentro de .animais-descricao

// Selecione o último p do site

DOM PARA INICIANTES

forEach e Arrow Function

forEach

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

```
const imgs = document.querySelectorAll('img');

imgs.forEach(function(item){
  console.log(item);
});
```

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;

```
const imgs = document.querySelectorAll('img');

imgs.forEach(function(valorAtual, index, array){
    console.log(item); // o item atual no loop
    console.log(index); // o número do index
    console.log(array); // a Array completa
});
```

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.

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

titulosArray.forEach(function(item){
  console.log(item);
});
```

Arrow Function

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

```
const imgs = document.querySelectorAll('img');

imgs.forEach((item) => {
  console.log(item);
});
```

Argumentos e Parênteses

```
const imgs = document.querySelectorAll('img');

// argumento único não precisa de parênteses
imgs.forEach(item => {
  console.log(item);
});

// múltiplos argumentos precisam de parênteses
imgs.forEach((item, index) => {
  console.log(item, index);
});

// sem argumentos precisa dos parênteses, mesmo vazio
let i = 0;
imgs.forEach(() => {
  console.log(i++);
});
```

É melhor utilizar os parênteses

Return

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

```
const imgs = document.querySelectorAll('img');

imgs.forEach(item =>
  console.log(item)
);

imgs.forEach(item => console.log(item));
```

*Não é permitido fechar a linha
com ;*

Exercício

```
// Mostre no console cada parágrafo do site  
  
// Mostre o texto dos parágrafos no console  
  
// Como corrigir os erros abaixo:  
const imgs = document.querySelectorAll('img');  
  
imgs.forEach(item, index => {  
    console.log(item, index);  
});  
  
let i = 0;  
imgs.forEach( => {  
    console.log(i++);  
});  
  
imgs.forEach(() => i++);
```

DOM PARA INICIANTES

Classes e Atributos

classList

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

```
const menu = document.querySelector('.menu');

menu.className; // string
menu.classList; // lista de classes
menu.classList.add('ativo');
menu.classList.add('ativo', 'mobile'); // duas classes
menu.classList.remove('ativo');
menu.classList.toggle('ativo'); // adiciona/remove a classe
menu.classList.contains('ativo'); // true ou false
menu.classList.replace('ativo', 'inativo');
```

attributes

Retorna uma array-like com os atributos do elemento.

```
const animais = document.querySelector('.animais');

animais.attributes; // retorna todos os atributos
animais.attributes[0]; // retorna o primeiro atributo
```

getAttribute e setAttribute

Métodos que retornam ou definem de acordo com o atributo selecionado

```
const img = document.querySelector('img');

img.getAttribute('src'); // valor do src
img.setAttribute('alt', 'Texto Alternativo'); // muda o alt
img.hasAttribute('id'); // true / false
img.removeAttribute('alt'); // remove o alt

img.hasAttributes(); // true / false se tem algum atributo
```

É muito comum métodos de *get* e *set*;

Read Only vs Writable

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

```
const animais = document.querySelector('.animais');

animais.className; // string com o nome das classes
animais.className = 'azul'; // substitui completamente a string
animais.className += ' vermelho'; // adiciona vermelho à string

animais.attributes = 'class="ativo"'; // não funciona, read-only
```

Lembre-se que podemos modificar
o valor de uma propriedade
`objeto.propriedade = ''`

Exercício

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

DOM PARA INICIANTES

Dimensões e Distâncias

Height e Width

Estas são propriedades e métodos dos objetos `Element` e `HTMLElement`, a maioria delas são Read Only

```
const section = document.querySelector('.animais');

section.clientHeight; // height + padding
section.offsetHeight; // height + padding + border
section.scrollHeight; // height total, mesmo dentro de scroll
```

Mesma coisa para o `Width`,

`clientWidth` ...

offsetTop e offsetLeft

```
const section = document.querySelector('.animais');

// Distância entre o topo do elemento e o topo da página
section.offsetTop;

// Distância entre o canto esquerdo do elemento
// e o canto esquerdo da página
section.offsetLeft;
```

getBoundingClientRect()

Método que retorna um objeto com valores de width, height, distâncias do elemento e mais.

```
const section = document.querySelector('.animais');

const rect = section.getBoundingClientRect();
rect.height; // height do elemento
rect.width; // width do elemento
rect.top; // distância entre o topo do elemento e o scroll
```

Window

```
window.innerWidth; // width do janela  
window.outerWidth; // soma dev tools também  
window.innerHeight; // height do janela  
window.outerWidth; // soma a barra de endereço  
  
window.pageYOffset; // distância total do scroll horizontal  
window.pageXOffset; // distância total do scroll vertical  
  
if(window.innerWidth < 600) {  
  console.log('Tela menor que 600px');  
}
```

matchMedia();

Utilize um media-querie como no CSS para verificar a largura do browser

```
const small = window.matchMedia(' (max-width: 600px)');

if(small.matches) {
  console.log('Tela menor que 600px')
} else {
  console.log('Tela maior que 600px')
}
```

Dica

- Selecione o elemento no inspetor (dom)
- Abra o console e digite \$0 para selecionar o mesmo
- Os elementos selecionados anteriormente são \$1, \$2 ...

Exercício

```
// Verifique a distância da primeira imagem  
// em relação ao topo da página  
  
// Retorne a soma da largura de todas as imagens  
  
// Verifique se os links da página possuem  
// o mínimo recomendado para telas utilizadas  
// com o dedo. (48px/48px de acordo com o google)  
  
// Se o browser for menor que 720px,  
// adicione a classe menu-mobile ao menu
```

DOM PARA INICIANTES

Eventos

addEventListener

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

```
const img = document.querySelector('img');

// elemento.addEventListener(event, callback, options)
img.addEventListener('click', function() {
  console.log('Clicou');
})
```

O terceiro parâmetro é opcional.

Callback

É boa prática separar a função de callback do addEventListener, ou seja, declarar uma função ao invés de passar diretamente uma função anônima

```
const img = document.querySelector('img');
function callback() {
    console.log('Clicou');
}

img.addEventListener('click', callback); // 🚀
img.addEventListener('click', callback()); // undefined
img.addEventListener('click', function() {
    console.log('Clicou');
})
img.addEventListener('click', () => {
    console.log('Clicou');
})
```

Event

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

```
const img = document.querySelector('img');

function callback(event) {
  console.log(event);

}

img.addEventListener('click', callback);
```

Geralmente utilizam `e` como nome
do parâmetro

Propriedades do Event

```
const animaisLista = document.querySelector('.animais-lista');

function executarCallback(event) {
    const currentTarget = event.currentTarget; // this
    const target = event.target; // onde o clique ocorreu
    const type = event.type; // tipo de evento
    const path = event.path;
    console.log(currentTarget, target, type, path);
}

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.

```
const linkExterno = document.querySelector('a[href^="http"]');

function clickNoLink(event) {
    event.preventDefault();
    console.log(event.currentTarget.href);
}

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.

```
const img = document.querySelector('img');

function callback(event) {
  console.log(this); // retorna a imagem
  console.log(this.getAttribute('src'));
}

img.addEventListener('click', callback);
```

*Geralmente igual ao
`event.currentTarget`*

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.

```
const h1 = document.querySelector('h1');

function callback(event) {
    console.log(event.type, event);
}

h1.addEventListener('click', callback);
h1.addEventListener('mouseenter', callback);
window.addEventListener('scroll', callback);
window.addEventListener('resize', callback);
window.addEventListener('keydown', callback);
```

<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`.

```
function handleKeyboard(event) {
  if(event.key === 'a')
    document.body.classList.toggle('azul');
  else if(event.key === 'v')
    document.body.classList.toggle('vermelho');

}

window.addEventListener('keydown', callback);
```

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.

```
const imgs = document.querySelectorAll('img');

function imgSrc(event) {
  const src = event.currentTarget.getAttribute('src');
  console.log(src);
}

imgs.forEach((img) => {
  img.addEventListener('click', imgSrc);
});
```

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
```

```
// Se o usuário clicar na tecla (t), aumente todo o texto do  
site.
```

DOM PARA INICIANTES

Navegação por Tabs

Adicionar Classes para Manipulação

A ideia de navegação por tabs, é ter uma lista de itens que controla a visualização de uma lista de conteúdo. Cada item da lista possui um conteúdo relacionado ao mesmo.

```
<!-- Primeiro, adicionar classes que irão  
facilitar a manipulação dos elementos -->  
<ul class="animais-lista js-tabmenu">  
    ...  
</ul>  
<div class="animais-descricao js-tabcontent">  
    ...  
</div>
```

Selecionar os itens

```
const tabMenu = document.querySelectorAll('js-tabmenu li');
const tabContent = document.querySelectorAll('js-tabmenu
section');
```

Função Callback

Recebe index como parâmetro para ativar a tab. Sempre que ativar, remove a classe ativo de todos os outros elementos.

```
function activeTab(index) {  
    tabContent.forEach((content) => {  
        content.classList.remove('ativo');  
    });  
    tabContent[index].classList.add('ativo');  
}
```

Adicionar o Evento

Neste caso precisamos passar antes a função anônima no callback, para podermos passar o index como argumento de activeTab

```
tabMenu.forEach((itemMenu, index) => {
  itemMenu.addEventListener('click', () => {
    activeTab(index);
  });
});
```

Assim que Carregar

Adicionar a classe ativo ao primeiro elemento e adicionar a classe js ao html. Assim identificamos se o JavaScript está habilitado ou não.

```
<!-- No head do browser -->
<script>document.documentElement.className += ' js';</script>
```

COPiar

```
// Verificar se existe elemento em tabContent e tabMenu
if(tabContent.length && tabMenu.length) {
    tabContent[0].classList.add('ativo');
    ...
}
```

Animação com CSS

Animação simples com css, sai de display none para display block.

```
.js .js-tabcontent section {  
    display: none;  
}  
  
.js-tabcontent section.ativo {  
    display: block !important;  
    animation: show .5s forwards;  
}  
  
@keyframes show {  
    from {  
        opacity: 0;  
        transform: translate3d(-30px, 0, 0);  
    }  
    to {  
        opacity: 1;  
        transform: translate3d(0px, 0, 0);  
    }  
}
```

*o texto até dos leitores de tela
(acessibilidade)*