DOM

DOCUMENT OBJECT MODEL (DOM)

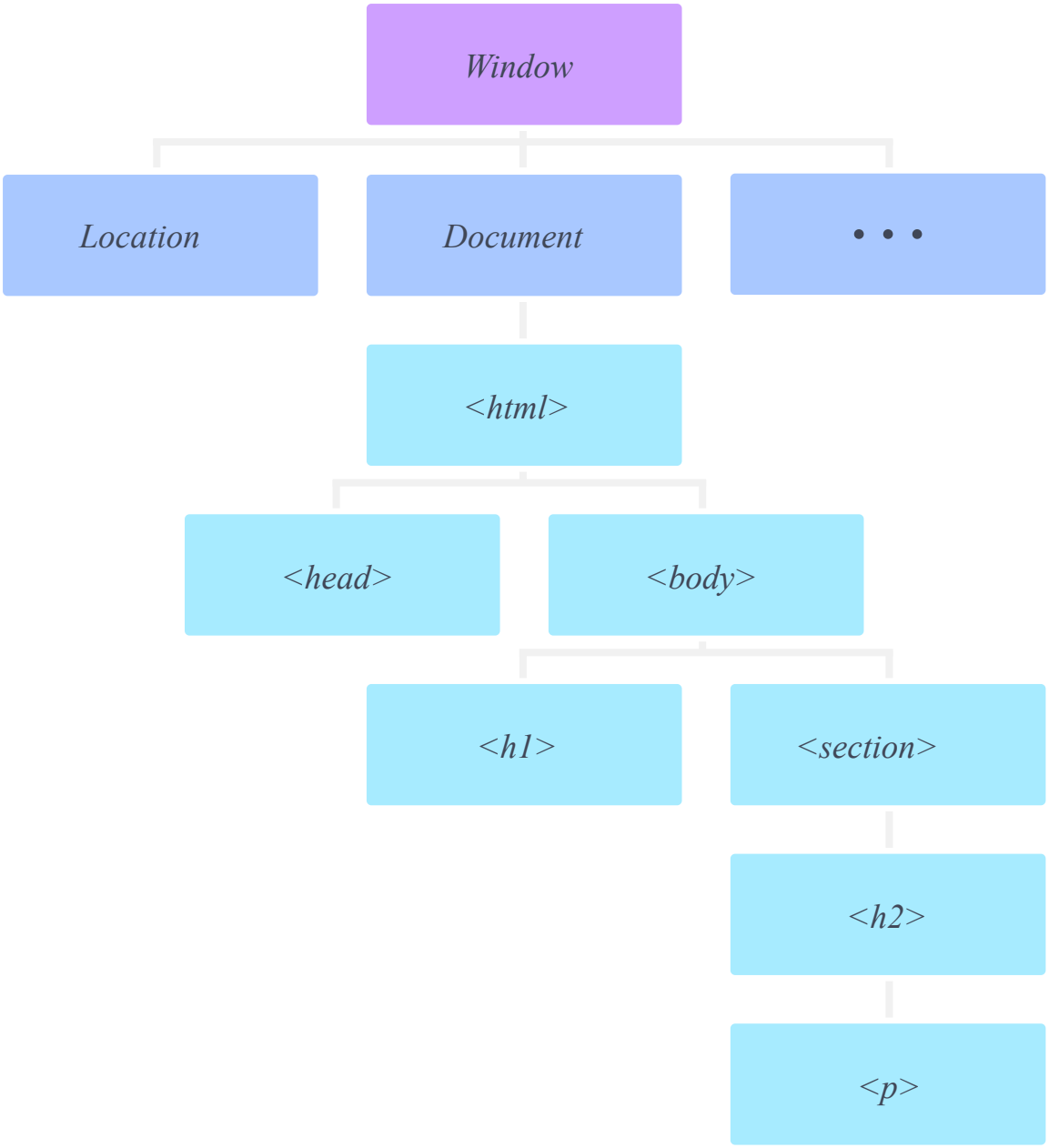
É uma interface que representa documentos HTML e XML através de objetos. Com ela é possível manipular a estrutura, estilo e conteúdo destes documentos.

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

```
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.

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 titulo
```

Seleção de Elementos

ID

`getElementById` seleciona e retorna elementos do DOM

id

```
// 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.

tag

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

// 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.

querySelector

```
const animais = document.querySelector('.animais');
const contato = document.querySelector('#contato');
const ultimoItem = document.querySelector('.animais-lista li:last-child');
const linkCSS = document.querySelector('[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

querySelectorAll

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

// 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.

nodelist

```
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.

array-like

```
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

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

```
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;

foreach

```
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.

array-like

```
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.

arrow

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

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

PARÂMETROS E PARÊNTESES

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

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

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

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

RETURN

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

=>

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

imgs.forEach(item =>
  console.log(item)
  // Não é permitido fechar a linha com ;
);


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

Exercício

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

ex3

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

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

let i = 0;
imgs.forEach( => {
    console.log(i++);
});

imgs.forEach(() => i++);
```

CLASSLIST

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

classlist

```
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.

attributes

```
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

get_set

```
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
```

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

```
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
```

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

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

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

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

```
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.

event

```
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

title

```
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 previnir que o link seja ativado.

prevent default

```
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.

this

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

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

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

```
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);
```

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

```
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', 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

```
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

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

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

menu.outerHTML; // todo o html do elemento
menu.innerHTML; // html interno
menu.innerText; // texto, sem tags

menu.innerText = '<p>Texto</p>'; // a tag vai como texto
menu.innerHTML = '<p>Texto</p>'; // a tag é renderizada
```

TRASVERSING

Como navegar pelo DOM, utilizando suas propriedades e métodos.

transversing

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

lista.parentElement; // pai
lista.parentElement.parentElement; // pai do pai
lista.previousElementSibling; // elemento acima
lista.nextElementSibling; // elemento abaixo

lista.children; // HTMLCollection com os filhos
lista.children[0]; // primeiro filho
lista.children[--lista.children.length]; // último filho

lista.querySelectorAll('li'); // todas as LI's
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

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

lista.previousElementSibling; // elemento acima
lista.previousSibling; // node acima

lista.firstChild; // primeiro node child
lista.childNodes; // todos os node child
```

MANIPULANDO ELEMENTOS

É possível mover elementos no dom com métodos de Node.

elements

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

contato.appendChild(lista); // move lista para o final de contato
contato.insertBefore(lista, titulo); // insere a lista antes de titulo
contato.removeChild(titulo); // remove titulo de contato
contato.replaceChild(lista, titulo); // substitui titulo por lista
```

NOVOS ELEMENTOS

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

createElement

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

const novoH1 = document.createElement('h1');
novoH1.innerText = 'Novo Título';
novoH1.classList.add('titulo');

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

```
const titulo = document.querySelector('h1');
const titulo2 = document.querySelector('h1');
const novoTitulo = titulo;
// titulo, titulo2 e novoTitulo são iguais

const cloneTitulo = titulo.cloneNode(true); //true sinaliza para incluir
os filhos
const contato = document.querySelector('.contato');
contato.appendChild(cloneTitulo);
```

Exercício

- Duplique 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