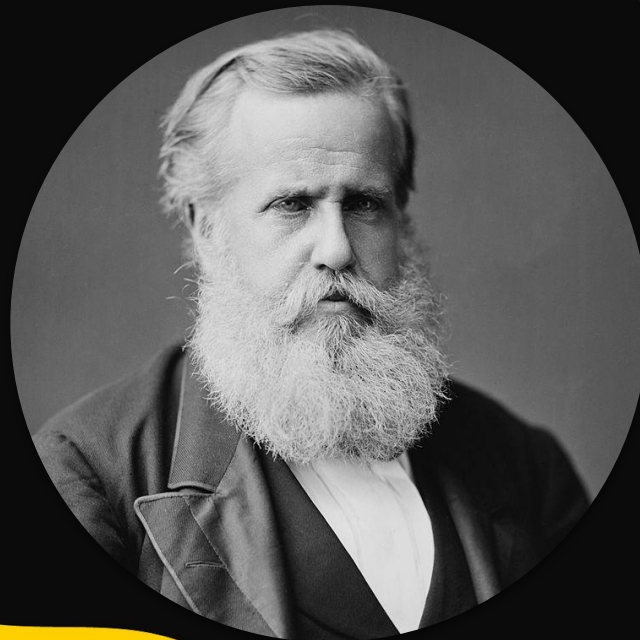




Conhecendo os eventos do DOM

Módulo 2 - Aula 1 - TECH



Retrato por Mathew Brady, 1876



Todos os direitos reservados
©2022 Resilia Educação



INTERATIVIDADE

Tuóffner



Todos os direitos reservados
©2022 Resilia Educação

Integração HTML JS

Para que uma **aplicação web** seja realmente **interativa**, precisamos de uma forma de **manipular** os **elementos** presentes na **página** com o **JavaScript**. Assim, conforme a pessoa usuária utiliza a **aplicação**, ela se **modifica** e se comporta de maneira **específica**.



A formal oil painting of Dom Pedro II, Emperor of Brazil. He is depicted from the chest up, facing slightly to the right but looking towards the viewer. He has a full, white beard and mustache, and his hair is thinning. He is wearing a dark blue or black military-style uniform with elaborate gold embroidery on the lapels and cuffs. A wide sash with red, white, and blue stripes runs diagonally across his chest. He is adorned with several medals and orders, including a large star-shaped medal on his left breast and a cross-shaped medal on his right. The background is a plain, light-colored wall.

DOM



DOM

DOM (Document Object Model) é um padrão utilizado para **acessar** e **manipular** um documento e seus **elementos**. No nosso caso, a **página web**.

Com ele podemos **alterar, deletar, criar e acessar elementos** de uma página **HTML** utilizando **JavaScript**.

DOM: estrutura

O **DOM** é estruturado como uma **árvore**. A raiz (elemento principal) é a **tag HTML**. As tags **head** e **body** são as primeiras **filhas** da tag raiz e podem conter tags mais internas e essas, por sua vez, podem ter tags mais internas ainda.

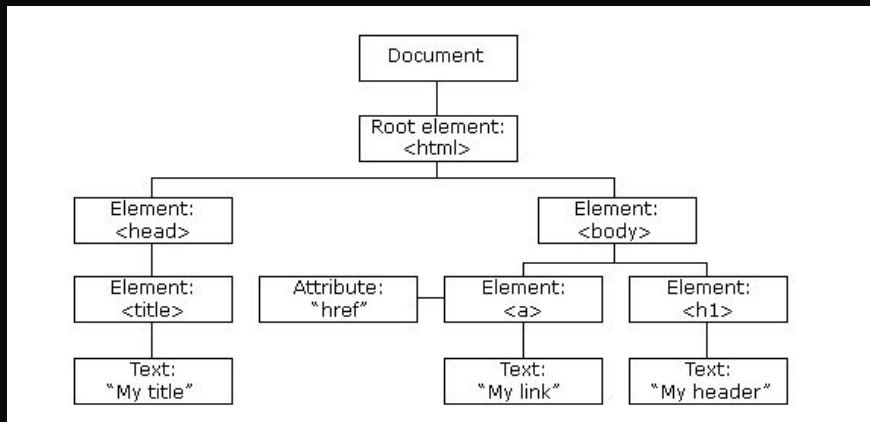


Imagem retirada do w3 schools - Javascript HTML DOM

DOM: interface HTML JS

Para **manipular** a página **HTML** a partir do **JavaScript**, utilizamos como base o *document*.

```
document.body; //body da página HTML
```

```
document.head; //head da página HTML
```

DOM: escrita na página

Para **escrever** numa **página** utilizando o DOM, utilizamos o **método/função** *document.write*.

```
document.write("<h1>Olá mundo!</h1>");
```




Busca por Elementos



DOM: Busca por elementos

Existem **diferentes formas** de se **buscar** por um (ou mais) **elementos** no **DOM**. Essa busca pode ser feita utilizando **como referência**:

- ⇒ **Nome da tag**
- ⇒ **Alguma classe da tag**
- ⇒ **O id da tag**

DOM: Busca por elementos - getElementById

O método/função *getElementById* procura um elemento pelo id fornecido como argumento.

Caso encontrado, este elemento pode ser manipulado via JavaScript.

```
<body>
  <button id='hello' onclick="alert('Olá mundo!')">Diga Olá</button>
  <script>
    var helloBtn = document.getElementById('hello');
    helloBtn.click();
  </script>
</body>
```

DOM: Busca por elementos - `querySelector`

O método/função `querySelector` procura um elemento pelo `id`, `tag` ou `classe` fornecido como argumento. Ela retorna o primeiro elemento da árvore encontrado que satisfaz a condição.

A notação utilizada para cada um é:

- `id: #`
- `class: .`
- `tag: a própria tag`

```
var btn = document.querySelector('button'); //tag
var cabecalho = document.querySelector('#cab-h1'); //id
var foto = document.querySelector('.profile-pic'); //class
```

DOM: Busca por elementos - querySelectorAll

O método/função `querySelectorAll` procura um elemento pelo `id`, `tag` ou `classe` fornecido como `argumento`. Ela retorna `todos` os elementos da `árvore` encontrados, que satisfazem a condição, em um `array`.

```
var fotosArr = document.querySelectorAll('.pictures'); //class
```

A close-up, slightly blurred photograph of a person's hands sorting through a closet. The closet is filled with numerous wooden hangers, each holding a piece of clothing. The clothes feature various patterns, including leopard print, floral, and geometric designs. The person's hands are visible in the foreground, reaching into the closet to touch the fabric of the garments. The overall lighting is warm and soft, creating a cozy atmosphere.

Criando Elementos



DOM: criação de elementos

Para **criar** novos **elementos**, o método/função ***createElement*** é utilizado. Ele recebe como **argumento** a **tag** do elemento a ser criado.

```
var divDesc =  
document.createElement('div');
```


DOM: Adicionando um elemento a um nó

Cada **elemento** da árvore do DOM é um **nó**.
Podemos utilizar o método **append** para **adicionar** um elemento ao **final** de **outro**.

```
var divDesc = document.createElement('div');  
var paragrafoDesc = document.createElement('p');  
divDesc.append(paragrafoDesc);
```

DOM: deleção de elementos

Para **deletar elementos**, o método/função **remove** é utilizado. Ele está associado a um nó/tag da árvore do DOM.

```
var cardDescricao =  
document.querySelector("#card-1321");  
cardDescricao.remove();
```

DOM: atributos de elementos

Atributos são campos/características das tags HTML.

- ⇒ src de uma imagem
- ⇒ texto interno de um cabeçalho ou parágrafo
- ⇒ estilo de uma tag

DOM: atributo `innerText`

O atributo *innerText* de um nó abriga todo conteúdo textual interno da tag. Podemos modificar o conteúdo de uma tag textual utilizando-o.

```
var cabecalho = document.querySelector("h1");  
cabecalho.innerText = "Manipulando a página com DOM!";
```

DOM: atributo innerHTML

O atributo `innerHTML` de um nó abriga todo conteúdo HTML interno da tag. Ao utilizá-lo, o conteúdo atribuído será renderizado em HTML. Podemos adicionar novas tags e conteúdo à página com este atributo.

```
var navHeader = document.querySelector("nav");
navHeader.innerHTML = `
    <h1>Webdev</h1>
    <div>
        <p>Home</p>
        <p>Quem somos</p>
        <p>Contato</p>
    </div>
`;
```

DOM: estilo

Para **modificar** propriedades de **estilo** de um **nó** da **árvore** do **DOM** utilizamos os atributos **style.<propriedade>**, atribuindo novos valores para as propriedades desejadas.

```
var cabecalho = document.querySelector('h1');  
cabecalho.style.color = 'red'; //modifica a cor para vermelho
```



ATIVIDADE

Manipulação do DOM



Atividade: Tá aberto?

Com base nos conteúdos vistos, utilize os métodos `document.querySelector`, `document.append`, `document.createElement`, os atributos de `innerText` e `innerHTML` para implementar o letreiro de funcionamento de um restaurante e o atributo `style.<propriedade>` para alterar o estilo do botão.

Em uma nova pasta, com um novo arquivo HTML e outro JS:

- ⇒ Crie uma página que contenha dois botões (“abrir” e “fechar”)
- ⇒ Ao clicar no botão “abrir”, sua aplicação deve:
 - Criar um novo parágrafo com o texto “Estamos abertos!”
 - Adicionar o elemento criado ao corpo da página
 - Alterar a cor do botão para informar que ele foi clicado (ex: deixá-lo verde)
- ⇒ Ao clicar no botão “fechar”, sua aplicação deve:
 - Encontrar o parágrafo com o texto “Estamos abertos!”
 - Modificar o texto do parágrafo para “Estamos fechados!”
 - Alterar a cor do botão para informar que ele foi clicado (ex: deixá-lo vermelho)

A close-up, low-key photograph of industrial machinery, featuring large, dark, metallic gears and a central shaft. The lighting is dramatic, with highlights on the teeth of the gears and the shaft, creating a sense of depth and texture. The overall color palette is dark, with shades of grey, black, and a hint of blue.

Funções



Funções: cidadãos de primeira ordem

Em JavaScript funções são **cidadãos de primeira ordem** (first-class citizens). Isso significa que as **funções** se **comportam** como **variáveis**!

Elas podem ser **passadas como argumento** e até mesmo **retornadas** por outras funções.

Funções: cidadãos de primeira ordem

Exemplo:

```
function operaArray(arr, func) {  
    return func(arr);  
}  
  
function somaArray(arr) {  
    var soma = 0;  
    for (var i=0; i < arr.length; i++) {  
        soma += arr[i];  
    }  
    return soma;  
}  
  
console.log(operaArray([1,2,3], somaArray))
```

Funções: cidadãos de primeira ordem

No exemplo anterior, a primeira função recebe dois **parâmetros**, um é um array e o outro é uma **função** que será **executada** dentro dela.

A função **operaArray** ainda **não sabe** que tipo de função vai receber, ela só sabe que precisa **executá-la**.

Uma função é **executada** quando colocamos o () depois do nome dela.



Funções Anônimas



Funções anônimas

Como o nome indica, **funções anônimas** são aquelas que **não possuem** um **nome**! Elas podem ser **atribuídas** a uma **variável** ou passada como **argumento** em uma **chamada de função**.

Funções anônimas: atribuição à variável

Exemplo:

```
var saudacao = function (nome) {  
    return `Olá ${nome}`;  
}  
  
console.log(saudacao("Turma"));
```

Funções anônimas: argumento em chamada

Exemplo:

```
function operaArray(arr, func) {  
    return func(arr);  
}  
  
console.log(operaArray([1,2,3], function (arr) {  
    var soma = 0;  
    for (var i=0; i < arr.length; i++) {  
        soma += arr[i];  
    }  
    return soma;  
}));
```

Funções anônimas: argumento em chamada

Diferente do exemplo anterior em que **declaramos** (escrevemos) a **função fora da chamada** e depois passamos o nome dela como **parâmetro**, agora estamos **escrevendo** a função **diretamente como um parâmetro** na chamada da outra.

Usamos esse caso quando **não vamos utilizar** a função em mais **nenhum outro lugar**, dessa forma, não precisamos nos preocupar em dar nome para essa função, nem **declarar ela** de forma **independente**.

A Sony PlayStation 2 DualShock 2 controller is the central focus, lying on a textured, light-colored surface. Behind it is a red carrying case with a black strap and a silver carabiner. To the right of the controller is a small, grey USB adapter. The controller is white with black buttons and a red light bar (not visible). The text "SONY PlayStation" is printed on the top of the controller. The background is slightly blurred, showing more of the textured surface.

Eventos e JavaScript



Eventos e JavaScript

Os **eventos gerados** em uma aplicação web podem ser utilizados como **gatilho** para execução de **funções** em JavaScript.

Dessa forma, o **navegador escuta** os eventos e, caso eles ocorram, as **funções** criadas para **reação** são **acionadas**.

- ⇒ **Click**
- ⇒ **Scroll**
- ⇒ **Drag/Drop**



As estruturas de eventos em JavaScript

As **estruturas** utilizadas para **tratar eventos** em JavaScript são chamados *EventHandlers*. Podemos **adicionar** um *eventHandler* a um **elemento** do DOM com *addEventListener* ou *on<evento>*.

As estruturas de eventos em JavaScript

Exemplo:

```
var cabecalho = document.querySelector('h1');  
cabecalho.onclick = function () {  
    alert("Clique único!");  
}  
  
cabecalho.addEventListener("dbclick", function () {  
    alert("Clique duplo!");  
});
```



Diferença: `on<evento>` e `addEventListener`

Ao adicionar um **tratador de eventos** com `.on<evento>` apenas um tratador pode ser utilizado. Além disso, uma **função** deve ser **atribuída** ao **evento**.

Já com `addEventListener` podemos ter **diversos tratadores** para um **mesmo evento**. Sua **estrutura** tem o **nome do evento** como argumento e a **função** tratadora.

Diferença: on<evento> e addEventListener

Exemplo:

```
var cabecalho = document.querySelector('h1');  
cabecalho.onclick = function () {  
    alert("Clique único!");  
}
```

```
cabecalho.addEventListener("dbclick", function () {  
    alert("Clique duplo!");  
});
```

Quais os tipos de eventos possíveis?

São muitos!!!! E é impossível decorar todos, por isso é importante **sempre consultar a documentação** para nos ajudar com esses casos.

Existem eventos de **clique**, de **movimentação do mouse**, que verificam se houve alguma **mudança no elemento** e muito mais.



MOMENTO APRENDIZAGEM EM PARES

