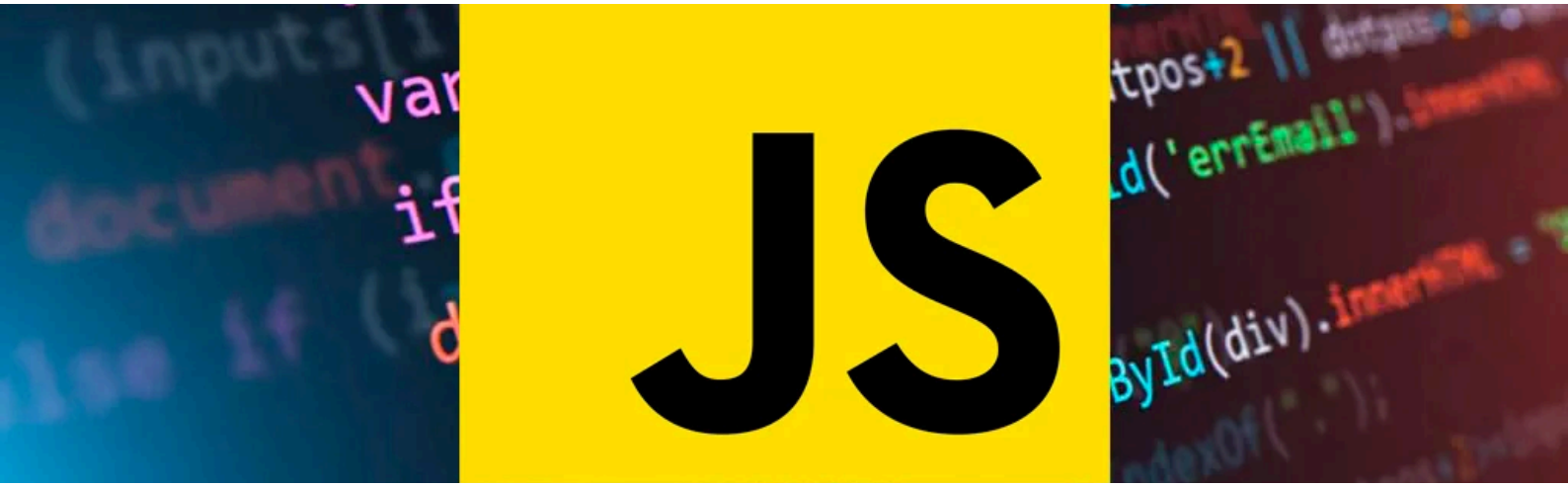
JAVASCRIPT

Curso 2



MANPULANDO ELEMENTOS NO DOM

Consiste na prática de alteração de elementos na tela.

document.getElementByClassName('app__card-button--curto') - Uma forma de encontrar a o item desejado, mas desde que seja uma classe. Neste exemplo nós buscamos a classe que corresponde o botão.

document.getElementId('start-pause') - Este você identifica o id daquele código

DOM - Document Object Model

Trata-se da junção de todos os arquivos criados para aquele projeto, como: **HTML, CSS, JS** e

entre outros. Tudo isso criado se torna o Dom, como por exemplo esta aplicação:

Veja esta linha de código e entenda:

<head> <script src = "/main.js" defer> </script> - Essa linha serve para referenciar a pasta de JavaScript na de html, antes não era possível usar no head por contra de leitura de código, ao final apresentaria erro. Agora, com o uso de **DEFER**, é possível utilizar pois a lógica é a seguinte:

- Rodar toda as linhas de html e por fim usar o de script

MUDANÇA DE CONTEXTO (COR) NA TELA

```
focoBt.addEventListener('click', () => {
    html.setAttribute('data-contexto', 'foco')
})
```

O código apresentado acima corresponde a tal lógica:

- **addEventListener -** é uma função embutida no JavaScript que leva o evento para ouvir e um segundo argumento a ser chamado sempre que o evento descrito for acionado.

- **Click -** Trata-se do comando do ONCLICK de clicar

- **html.setAttribute('data-contexto', 'foco') -** Aqui temos um comando de atribuir a mudança desejada, usando a variável de busca do html

MÉTODO ADDEVENTLISTENER

Este método é utilizado em diversas finalidades, a principal é de executar algo quando for clicar em qualquer botão

getAttribute:

O método **getAttribute** é usado para obter o valor de um atributo específico de um elemento HTML. Ele recebe um argumento, que é o nome do atributo que desejamos recuperar o valor.

---

setAttribute:

O método **setAttribute** é usado para definir ou modificar o valor de um atributo em um elemento HTML. Ele aceita dois argumentos: o primeiro é o nome do atributo que queremos definir ou modificar, e o segundo é o valor que queremos atribuir a esse atributo. Se o atributo já existir, o método setAttribute irá sobrescrevê-lo; caso contrário, ele criará um novo atributo.

---

hasAttribute:

O método **hasAttribute** é usado para verificar se um elemento possui um atributo específico. Ele recebe um argumento, que é o nome do atributo que queremos verificar. O método retornará true se o atributo existir e false se o atributo não estiver presente.

---

removeAttribute:

O método **removeAttribute** é usado para remover um atributo específico de um elemento HTML. Ele recebe um argumento, que é o nome do atributo que desejamos remover. Por exemplo:

EXPLICANDO O CÓDIGO

`html.setAttribute('data-contexto', contexto)` - Aqui é um código dentro da função que tem como objetivo deixar "genérico" as informações para que seja possível aplicar nas outras funções com a finalidade de padronizar.

`banner.setAttribute('src', `imagens/${contexto}.png`)` - Temos a junção de uma sintaxe html com javaScript, a parte "${Contexto}" significa que será usado uma variável de outra função, veja:

```
const html = document.querySelector('html');  // Criando uma variável html
const focoBt = document.querySelector('.app__card-button--foco')
const curtoBt = document.querySelector('.app__card-button--curto')
const longoBt = document.querySelector('.app__card-button--longo')
const banner = document.querySelector('.app__image')


focoBt.addEventListener('click', () => {
    alterarContexto('foco')   // Chamando função para trocar a imagem ao c
})
```

MUDANDO TEXTO

Realizando um switch/case para colocar cada opção de texto a cada tela trocada

obs: Falar sobre p innerHTML

```
switch (contexto) {
    case "foco":

      titulo.innerHTML = `
       Otimize sua produtividade,<br>
           <strong class="app__title-strong">mergulhe no que importa.</strong>

      `
       break;
       case "descanso-curto":
       titulo.innerHTML = `
         Que tal dar uma respirada <br>
       <strong class="app__title-strong"> Faça uma <br> pausa curta!</strong>
       `
       break;
    default:
       break;
}
```

INNERHTML - Conceitos

innerHTML é uma propriedade em JavaScript usada para **acessar ou modificar o conteúdo HTML de um elemento.** Quando você usa innerHTML, pode obter o HTML interno de um elemento ou definir um novo conteúdo HTML para ele. Por exemplo:

```
var conteudo = document.getElementById('meuElemento').innerHTML;

document.getElementById('meuElemento').innerHTML = '<p>Novo conteúdo</p>';
```

Observação: O innerHTML também pode ser utilizado para criação de lista, Veja um exemplo:

```
titulo.innerHTML  += `
        Hora de voltar à superfície.
        <strong class = "app__title-strong">Faça uma pausa longa</strong>`
```

Eu sempre Incremento mais TEXTO.

Ou seja, a Sintaxe exige que vc aplique o “+” antes do “=” para incrementar mais

---

Adicionando e removendo classes

CLASSLIST - O que é ?

classList é uma propriedade em JavaScript que permite **acessar e manipular as classes CSS de um elemento HTML**. Ela oferece métodos convenientes para **adicionar, remover, alternar e verificar classes**. Alguns dos métodos comuns incluem:

- add(className): Adiciona uma classe ao elemento.
- remove(className): Remove uma classe do elemento.
- toggle(className): Alterna a presença de uma classe no elemento.
- contains(className): Verifica se o elemento possui uma determinada classe.

Exemplo:

var elemento = document.getElementById('meuElemento')	Pegar o id do Html para usar no JS
elemento.classList.add('novaClasse')	Adicionar algo neste elemento recuperado usando uma classe já codificada
elemento.classList.remove('classeExistente')	Remover algo no elemento recuperado
elemento.classList.toggle('classeAlternada')	Alterar
var temClasse = elemento.classList.contains('classeChecada')	Verificar se possui a classe desejada

FOREACH/REMOVE

```
botoes.forEach(function (contexto){

contexto.classList.remove('active')

});
```

Esse código remove a classe `active` de cada elemento na lista `botoes`. Ele faz isso iterando sobre cada elemento da lista e aplicando o método `classList.remove` para remover a classe.

Aqui está um resumo passo a passo:

1. `botoes` é uma lista de elementos (provavelmente obtida com `document.querySelectorAll` ou algo similar).

2. `forEach` é um método de array que executa uma função para cada elemento da lista.

3. A função anônima recebe cada elemento (`contexto`) e executa `contexto.classList.remove('active')`, removendo a classe `active` desse elemento.

Criando um objeto para reprodução de um áudio:

```
const musica = new Audio('sons/luna-rise-part-one.mp3')
```

Função responsável por fazer o áudio funcionar:

```
musicaFocoInput.addEventListener('change', () => { Criando uma função utilizando o addEventListener, para inserção de um evento. O evento change é acionado quando algo é mudado no formulário, exemplo: Se o usuário clicar no botão de pausar música, automaticamente o uso do change é importante.
```

Abaixo é possível identificar uma condicional de if, ou seja, caso a musica esteja pausada, ela deve tocar, caso não, ele deverá pausar.

```
  if(musica.paused){ – Paused (método que está na função Audio)
    musica.play()
  }else{
    musica.pause()
  }

})
```

```
const musica = new Audio('sons/luna-rise-part-one.mp3')

musicaFocoInput.addEventListener('change', () => {
    if(musica.paused){
        musica.play()
    }else{
        musica.pause()
    }

})
```

TEMPORIZADOR

```
const contagemRegressiva = () => {

  iniciar()  //Referenciando o novo método

  tempoDecorridoSegundos -= 1  // Tirando valor (Decrementado o valor do temporizador)

 }


tempo.addEventListener('click', contagemRegressiva)


function iniciar(){

  intervaloId = setInterval(contagemRegressiva, 1000)    // Método responsável por executar alguma função em um dterminado período de tempo

}
```

SetInterval - é uma função em JavaScript que permite executar um trecho de código ou uma função repetidamente em intervalos de tempo especificados. Ela é parte da API do temporizador do JavaScript

ClearInterval - Tem como objetivo para com aquela reprodução automática

---

TEXTCONTENT X INNERHTML

innerHTML:

- Define ou retorna o HTML ou XML de um elemento e seus descendentes.
- I**nterpreta e renderiza** qualquer marcação HTML dentro do elemento.
- Quando usado para definir o conteúdo, permite a inclusão de HTML, o que pode criar ou modificar elementos filhos dentro do elemento.
- Deve ser usado com cuidado para **evitar vulnerabilidades de segurança**, como injeção de código (XSS).

textContent:

- Define ou retorna o conteúdo textual de um elemento e seus descendentes.
- **Ignora qualquer marcação HTML** dentro do elemento, ou seja, retorna apenas o texto puro.
- Quando usado para definir o conteúdo, ele substitui todo o texto dentro do elemento, removendo qualquer marcação HTML existente.
- É mais **seguro para evitar injeção de código (XSS)** porque trata todo o conteúdo como texto, sem interpretar HTML.

CRONOMETRO

```
const tempoFormatado = tempo.toLocaleTimeString('pt-br', {minute: '2-digit', second: '2-digit'} )
```

Explicando linha de código - Este código formata um objeto de data e hora (tempo) em uma string que mostra os minutos e segundos no formato de dois dígitos, usando a localidade portuguesa do Brasil. Vamos detalhar:

- tempo.toLocaleTimeString('pt-br', {...}): Este método converte um objeto de data e hora (tempo) em uma String com base nas configurações locais especificadas. Neste caso, 'pt-br' indica que o formato deve seguir as convenções do português do Brasil.

- {minute: '2-digit', second: '2-digit'}: Este objeto especifica que apenas os minutos e segundos devem ser exibidos, e ambos devem ser mostrados com dois dígitos. Por exemplo, se os minutos forem 5, será exibido como "05".