**JavaScript Para Web – Crie Páginas Dinâmicas**

1. **Aula 1 – Conhecendo o JavaScript:**
   1. Clicando no Botão:
      1. Podemos escrever código JS dentro do próprio HTML, inclusive dentro de suas tags. Para exemplificar o dinamismo e uma parte da proposta desse curso, fizemos com que uma janela de alerta abrisse quando clicamos no botão “Pom”:





* + 1. O que permite isso é o ‘onclick=”alert(‘Pom’)”. O que está dentro de ‘alert’ precisa necessariamente estar em aspas simples para não entrar em conflito com as aspas duplas do HTML.
  1. Conectar JS com o HTML:
     1. Basta abrir e fechar a tag script com o src levando até a localização do arquivo JS que agora está separado do HTML:

<script src="main.js"></script>

* + 1. Para testar se o código foi devidamente carregado, podemos lançar um alert no nosso arquivo do código JS:

alert('olá, mundo!');



* + 1. No JS o ‘;’ é opcional, mas é sempre bom usar para evitar qualquer tipo de problema futuro. Lembrando que precisamos ter consistência no código, se começamos com ‘;’ precisamos terminar com ele!
  1. Buscar um Elemento:
     1. Para selecionar um elemento específico utilizamos o:

querySelector('.tecla\_pom');

* + - 1. E dentro das aspas funciona como no CSS, ou seja, ‘.’ Para classes, ‘#’ para ID’s, somente o nome da tag se for todas as tags, e assim por diante.
    1. Entretanto, o querySelector não funciona de qualquer forma como o alert. Se quisermos utilizar ele, precisamos seguir o escopo do documento, ou seja:

document.querySelector('.tecla\_pom');

* + 1. Note que ao fazer essa seleção no console do navegador, a nossa tecla será marcada:



* + 1. Ao dar enter no console, ele nos retorna a tag toda do botão selecionado:



* + - 1. Mas se o que estamos selecionando não existir, ele retorna nulo.
  1. Desafio:
     1. Para selecionar um input ou qualquer elemento que possua um type, precisamos colocar assim:

document.querySelector('input[type=tel]');



* 1. O que aprendemos:
     1. Clicando no botão:
        1. Conhecemos a tag audio do HTML e como ela funciona, também como adicionar código JavaScript inline a partir do atributo onclick na tag button, e também como é a função alert() do JavaScript.
     2. Conectar JS com HTML:
        1. Aprendemos porque devemos ter arquivos dedicados para cada linguagem, e a extensão .js para arquivos com JavaScript, porque chamamos o nosso arquivo principal de JavaScript de main.js, e também, como fazemos para inserir um arquivo .js dentro de uma página HTML com a tag script.
     3. Buscar um elemento:
        1. Aprendemos os tipos de seletores que podemos usar no JavaScript (elemento, classe e id), como utilizar a função querySelector para selecionar os elementos da nossa página HTML. Vimos nesta aula a referência document, que representa o documento HTML dentro do JavaScript, e o significa o Reference Error e como solucionar, além de entender que o JavaScript é uma linguagem case sensitive, e uso do operador ponto final para entrar dentro de referências como o document, e que o ponto e vírgula é opcional porém recomendado.

1. **Aula 2 – Funções:**
   1. Play no JS:
      1. Após seguir todo o processo de seleção e encontrar o elemento desejado, podemos colocar mais 1 ‘.’ Depois dele e percorrer por todas as funcionalidades desse elemento, como por exemplo, o play na nossa tag de áudio em HTML que está sem os controles para ficar invisível na página:

document.querySelector('#som\_tecla\_pom').play();

 - Representa que é uma função, quando no console do navegador

* + - 1. Ao dar enter, no navegador, podemos ouvir o som da tag que selecionamos a partir do ID no querySelector.
  1. O que é uma função:
     1. No nosso caso acima tudo só irá correr bem e corretamente caso seja feito no navegador, se fizermos no nosso documento .js dará erro pois colocamos ele no head do nosso HTML e, como o HTML é lido linha por linha até chegar ao seu final, ele está passando pelo js antes de conseguir carregar o body do documento, que é onde estão todos os nossos áudios.
     2. Por conta disso é super importante sempre prestarmos bastante atenção onde colocaremos a tag script no nosso documento HTML, uma vez que irá sempre depender do que queremos que ele faça.
     3. Portanto, se quisermos que ele utilize elementos que estão no body para seu funcionamento, não podemos colocar ele no head, precisa obrigatoriamente ser colocado depois do body para que não ocorra nenhum tipo de erro e nossa aplicação funcione perfeitamente.
     4. Algumas vezes também podem ocorrer bloqueios do navegador, impedindo que sons e outras interatividades sejam iniciadas antes da interação do usuário com a página. Para corrigir isso criamos a nossa própria função que toca o som do Pom e, ao chamar essa função no navegador, o nosso som é executado.
     5. A pesar de ainda não ser o que queremos, já estamos chegando bem perto:

function tocaSomPom(){

  document.querySelector('#som\_tecla\_pom').play();

}



* 1. Clique no botão:
     1. Colocamos o atributo ‘onclick’ no nosso chamado do botão pom que criamos lá atrás e atribuímos a ele a função tocaSomPom, mas sem ‘()’. Não colocamos ‘()’ pois queremos que o JS guarde essa função para apenas quando o botão for clicado, se colocarmos os ‘()’ assim que a pagina for carregada, o som irá tocar (ou dará o erro de interação do usuário comentado acima):

document.querySelector('.tecla\_pom').onclick = tocaSomPom;

* 1. O que aprendemos:
     1. Play no JS:
        1. Como manipular a tag <audio> do HTML através do JavaScript, como selecionar um elemento a partir de um seletor de id e a reproduzir um som a partir da função play(). Além disso, viu também como os erros são apresentados na aba Console da ferramenta DevTools.
     2. O que é uma função?:
        1. O que é uma função, para que servem, como declará-la e sua sintaxe básica. Além disso, viu que a ordem de execução de um código JavaScript importa, desde a inserção da tag script antes do fechamos da tag </body> no HTML até a ordem do código JavaScript dentro do seu próprio arquivo.
     3. Clique no botão:
        1. A utilizar o onclick como atributo no JavaScript, e como atribuir uma função ao onclcik sem que esta função seja invocada imediatamente.

1. **Aula 3 – Listas:**
   1. Lista de Elementos:
      1. Para escrever comentários em JS utilizamos //, diferente de python que é #.
         1. /\* \*/: coloca um bloco todo como comentário.
      2. querySelectorAll(‘’): cria uma lista com todos os elementos encontrados com o mesmo seletor que você procurou:

document.querySelectorAll('.tecla');





* 1. Referências:
     1. São como as variáveis que já estamos MUITO bem acostumados em python:

const listaDeTeclas = document.querySelectorAll('.tecla');

* + - 1. Diferentemente de python, precisamos sempre dizer se é uma referência constante, variável, bool e afins. A tipagem não é dinâmica em JS.
  1. Conhecendo Listas:
     1. Funcionam igual em python. Usamos índices para navegar entre seus itens.
     2. Podemos fazer a atribuição do tocaSomPom no onclick para o elemento índice 0 e ele toca quando clicado:

listaDeTeclas[0].onclick = tocaSomPom;

* 1. O que aprendemos:
     1. Lista de elementos:
        1. O que são os comentários de código e como utilizá-los para auxiliar durante a escrita da nossa aplicação. Vimos também uma alternativa ao querySelector quando necessário buscar muitos elementos, que é o querySelectorAll, que retorna uma lista (NodeList) com todos os elementos do seletor informado evitando a repetição de código.
     2. Referências:
        1. Para que serve uma referência e como declaramos uma referência de valor constante no código.
     3. Conhecendo listas:
        1. A estrutura de uma lista em JavaScript e como podemos acessar os elementos a partir dos índices com a sintaxe do colchetes.

1. **Aula 4 – Iterando Em Listas:**
   1. Percorrendo Uma Lista:
      1. Utilizamos um While para percorrer a lista de teclas:

while (contador < listaDeTeclas.length){

  listaDeTeclas[contador].onclick = tocaSomPom;

  contador = (contador + 1);

  console.log(contador);

}

* + 1. Utilizando o While todas as teclas estão funcionando, mas ainda com o som de Pom porque a nossa função ainda não sabe como identificar qual tecla deve fazer qual som.
  1. Funções com parâmetros:
     1. Utilizamos parâmetros quando precisamos que nossa função receba informações variadas. Igual em python, nada novo:

function tocaSom(idElementoAudio) {

  document.querySelector(idElementoAudio).play();

}

 - Ao executar no console do navegador ele faz o som correspondente.

* + 1. Os parâmetros são os nomes que damos a valores que uma função pode receber em sua chamada, que podem ou não ter um valor padrão. Os parâmetros de uma função são como variáveis ficam disponíveis apenas no corpo da função.
    2. Alternativa correta! Os parâmetros servem para auxiliar em alguma lógica no código dentro de uma função. Por exemplo, uma função calculaMedia(), pode ter como parâmetros notaA e notaB, que serão os valores utilizados para serem caclulados e retornar o valor da média.
  1. Funções anônimas:
     1. São funções que passamos para o While, por exemplo, indicando que aquela constante possui uma função a ser executada, mas não fala qual ainda para que não ocorra problema.
     2. Dentro da função anônima colocamos a função que realmente queremos que seja passada:

while (contador < listaDeTeclas.length){

  listaDeTeclas[contador].onclick = function(){tocaSom('#som\_tecla\_pom')};

  contador = (contador + 1);

}

* 1. Textos Dinâmicos:
     1. Podemos usar os template strings para concatenar textos usando `` e ${}:

while (contador < listaDeTeclas.length){

  const tecla = listaDeTeclas[contador];

  const instrumento = tecla.classList[1];

  //tamplate string

  const idAudio = `#som\_${instrumento}`;

  tecla.onclick = function(){tocaSom(idAudio)};

  contador = (contador + 1);

}

* + 1. No código acima criamos outras 3 constantes para deixar nosso código mais limpo. A primeira é a tecla que recebe a tecla referente ao contador atual, o segundo é o instrumento que recebe a segunda classe da tag button após acessar ela pelo classList de cada tecla da lista de teclas do contador atual e atribui ao instrumento, isso porque a segunda classe de cada tecla se refere especificamente ao nome de cada áudio e ao que está escrito em cada tecla. O idAudio foi criado com o texto dinâmico para que a cada vez que o While passasse o idAudio mudasse para o instrumento referente fazendo a atribuição do áudio respectivamente para sua tecla.
  1. Repetição Otimizada Com For:
     1. Com o for poupamos linhas e deixamos nosso código mais limpo e enxuto. Podemos declarar as variáveis e fazer suas atribuições ainda dentro dos ‘()’. Basicamente de resto funciona como em python:

for (let contador = 0; contador < listaDeTeclas.length; contador++) {

  const tecla = listaDeTeclas[contador];

  const instrumento = tecla.classList[1];

  const idAudio = `#som\_${instrumento}`; //tamplate string

  tecla.onclick = function(){tocaSom(idAudio)};

}

* + 1. Outra vantagem é não ter que declarar outras muitas variáveis caso queiramos acrescentar outros instrumentos na nossa aplicação, por exemplo. Isso deixaria nosso código repetitivo e poluído. Com o for isso é resolvido.
  1. O que aprendemos:
     1. Percorrendo uma lista:
        1. Como percorrer uma lista usando a estrutura de repetição while, a criar referências variáveis com let, e como incrementar o valor de uma variável, e criar uma condição para evitar o loop infinito, também conhecemos o atributo length contido nas listas que nos ajudou a obter dinamicamento o valor do tamanho de uma lista.
     2. Função com parâmetros:
        1. Como declarar e utilizar parâmetros dentro de uma função que criamos, e o que é uma função anônima e o retorno undefined de uma função.
     3. Textos dinâmicos:
        1. Como criar textos dinâmicos utilizando template string e como acessar as classes de um elemento através do atributo classList.
     4. Repetição otimizada com For:
        1. Como é a estrutura de repetição for e como ela pode nos ajudar com um código mais limpo, além da forma de incrementar um valor de variável com o operador ++.

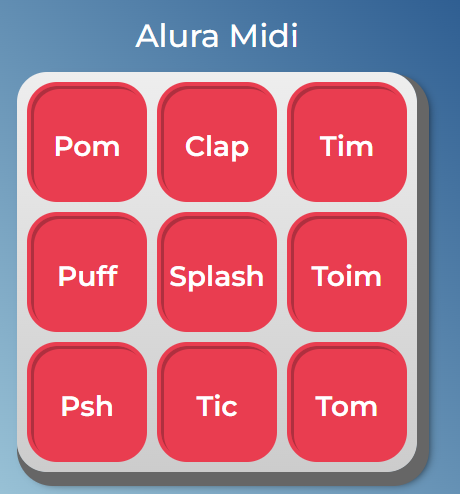
1. **Aula 5 – Eventos e Lógicas:**
   1. Eventos do Teclado:
      1. Podemos adicionar classes usando JS, basta usar a função add no classList:

tecla.classList.add('ativa')

* + 1. Queremos que sempre que estiver navegando pelas teclas da aplicação e apertar espaço ou enter, a tecla coloque o efeito da classe ativa, portanto, fizemos um novo evento de teclado para as teclas da aplicação, colocando uma função anônima que incrementa a classe ativa na tecla ao ser pressionada:

tecla.onkeydown = function(){tecla.classList.add('ativa')};

* + 1. O problema disso é que quando estamos passando pelas teclas da aplicação usando o tab do nosso teclado, todas as teclas acabam ficando vermelhas e com a classe ativa:



* + 1. Isso porque falamos para o JS adicionar a classe, mas não remover ela, portanto, foi o que ele fez.
  1. Adicionando e Removendo Classe:
     1. Para remover é o mesmo processo de adicionar, mas ao invés de add colocamos remove:

tecla.classList.remove('ativa')

* + 1. Se para adicionar a classe colocamos o evento de onkeydown, ou seja, assim que pressionar a tecla do teclado, para remover a classe utilizamos o evento onkeyup:

tecla.onkeyup = function(){tecla.classList.remove('ativa')};

* + 1. Ele corrigiu o problema de manter a classe quando soltamos a tecla enter ou espaço, mas ao pular para o próximo elemento com tab, ele continua vermelho.
  1. Condições no Código:
     1. Para solucionar esse problema do tab, podemos criar uma condição, ou seja, um if no nosso código que adiciona a classe ativa, falando para o nosso código só adicionar a classe se o evento do code acionado for a tecla enter ou espaço:

tecla.onkeydown = function(evento){

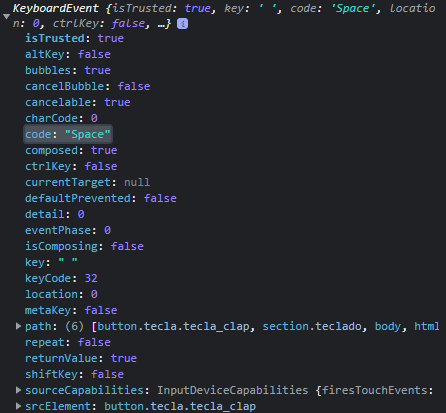
    if (evento.code === 'Enter' || evento.code === 'Space'){

      tecla.classList.add('ativa');

    }

  }

* + 1. Esse code é o nome da tecla que foi pressionado, exibido sempre que uma tecla é pressionada no navegador. Ao colocar um console.log(evento), podemos ver no console do navegador tudo o que está acontecendo quando apertamos uma tecla:



* + 1. Não se esqueça de passar o evento como parâmetro.
    2. Operador de 3 ‘=’ compara, além do valor, o tipo do dado, ou seja, se ambos são numbers, str’s e afins. Operador de 2 ‘=’ compara apenas valores, não importando se um é number e outro str, por exemplo.
  1. Operador Lógico:
     1. || é o operador lógico de ‘ou’, como utilizado na condição acima: ‘Se tal coisa OU outra coisa for verdade, então adiciona a class ativa’.
  2. Mais Condições:
     1. Criamos uma condição comparativa para não permitir o uso da função tocaSom() diretamente com seletores que não existam. Colocamos alertas e mensagens no log caso o seletor escolhido não exista ou não possa ser tocado, mas se puder ser tocado, tocará.
     2. !=: operador lógico de ‘diferente’.

function tocaSom(idElementoAudio) {

  const elemento = document.querySelector(idElementoAudio);

  if (elemento === null) {

    console.log("Elemento não encontrado")

  }

  if (elemento != null && elemento.localName === 'audio') {

    elemento.play();

  }

}

* + 1. Utilizamos outro if para certificar de que o elemento não é nulo e que o localName, ou seja, a tag desse elemento, seja um áudio. Dessa forma, se ambas as condições forem satisfeitas, o áudio será tocado.
  1. Melhorando o Código:
     1. &&: Operador lógico de ‘E’, ou seja, se uma coisa E outra forem verdadeiras, então executa o bloco.
     2. Else: ‘Se não’, ou seja, se a condição proposta acima não for satisfeita, execute esse código:

function tocaSom(idElementoAudio) {

  const elemento = document.querySelector(idElementoAudio);

  if (elemento && elemento.localName === 'audio') {

    elemento.play();

  }

  else {

    //alert('Elemento Não Encontrado ou Seletor Inválido!!')

    console.log('Elemento Não Encontrado ou Seletor Inválido!!')

  }

}

* + 1. Removemos o primeiro if que estava lá, pois se a condição que permaneceu não for satisfeita, queremos que mostre o erro independentemente de o problema ser com o seletor ou o elemento.
    2. Ainda removemos a parte que diz ‘se elemento for diferente de nulo’, pois o JS é inteligente o suficiente para entender que ele precisa fazer a comparação do elemento com nulo, indefinido, vazio ou qualquer outra coisa que seja para determinar se ele existe ou não, ou seja, apenas colocando o elemento lá, ele já verifica se ele existe. Isso evita ficar escrevendo código desnecessariamente.