Curso de

JavaScript para Web: Crie páginas dinâmicas

Vanessa Me Tonini

Faça esse curso de JavaScript e:

Identificar os papéis de cada tecnologia da Web para construção de páginas

Associar o JavaScript com HTML através do DOM

Implementar funções com JavaScript

Investigar possibilidades de resolução de problemas de código

Selecionar operadores lógicos adequados para diferentes situações

Produzir soluções simples para páginas dinâmicas

00:00] Olá sejam bem-vindas e bem-vindos. Eu sou a Vanessa Tonini, sou instrutora aqui da Alura e você que veio da nossa formação de HTML CSS ou já sabe codificar páginas com HTML CSS, esse curso é para você.

[00:17] O Alura Midi é um instrumento musical de interface digital, onde a cada botão vamos reproduzir o som de um instrumento. Eu vou mostrar para vocês. Aqui o Pom, o Tim, o Splash.

[00:35] E o código do projeto já vai vir com o HTML e o CSS prontos. Isto para focarmos no desenvolvimento dos JavaScript do zero, utilizando recursos atuais da linguagem, como constantes, variáveis, querySelector, template string, eventos de teclado e de mouse e muito mais.

[00:58] Vale lembrar que neste curso não vamos utilizar recursos avançados do JavaScript, como integração com backend, APIs do navegador, classes e objetos a fundo. Se você ficou com interesse em desenvolver o Alura Midi, eu espero você no nosso próximo vídeo.

CLICANDO NO BOTAO

Olá! Agora que você já conhece o Midi, precisamos descobrir como vamos começar a desenvolver as funcionalidades dele. Por ser um instrumento musical, ao clicar nos botões, queremos que reproduza um som. Porém, como fazemos para o HTML reproduzir um som para nós? Vamos abrir o nosso código lá no Visual Studio Code, nosso editor de código.

[00:29] E abrimos toda a estrutura de pastas dos arquivos do curso e vemos que temos uma pasta com o nosso CSS, com as imagens e com os arquivos de som, que são os arquivos que vamos reproduzir, os sons que vamos reproduzir ao clicar nos botões. E o nosso em “index.html”, que é o HTML principal da nossa aplicação.

[00:50] Podemos ir direto para tag body e olhamos que tem uma section com uma série de buttons, que são os botões que vimos no navegador, que já estão estilizados com CSS para ter essa cara do Midi.

[01:09] Embaixo dessa sessão com os botões temos uma série de tags áudio. Essas tags áudios são as responsáveis por carregar e fornecer ao navegador os sons dos instrumentos que queremos reproduzir. A tag áudio não está aparecendo no HTML porque omitimos um atributo importante para a visualização dela. É um atributo dentro da tag áudio, chamado controls, esse atributo, colocando ele vai começar a exibir a tag.

[01:45] Então vou colocar em todas as minhas tags, vou selecionar e apertar o atalho “Ctrl + D”, que seleciona todos daquela mesma tag, e vou escrever aqui o atributo controls nas 9 tags áudio, vou salvar o meu arquivo HTML com atalho “Ctrl + S” e vou até o navegador.

[02:05] No navegador eu vou recarregar a minha página e vou notar que as tags áudios começaram a ser exibidas, justamente porque adicionamos o atributo controls. E percebemos que essa tag áudios já vem com controles nativos, como botão de play, uma barra de progresso, o número de tempo do som, um botão para aumentar ou diminuir volume e se eu clicar no botão de Play, já vai reproduzir os sons dos nossos instrumentos, que no caso são sons de percussão, de uma bateria. Eu vou reproduzi-los para vocês escutarem.

[02:41] Então a tecla Pom, o som do Clap, o som do Tim e o restante dos sons vocês podem ir conhecendo no projeto de vocês. No Firefox a tag áudio tem uma aparência muito específica e muito parecida com um mini player mesmo, ela é quadrada, com fundo preto, os botões são brancos e se a gente abrir no Chrome, então vou abrir o meu Chrome e vou atualizar.

[03:11] Eu começo ver esse mini player de uma forma completamente diferente. Na verdade, sim, é perceptível que é um player, porque tem os botões de play, tem a barra de progressão, o tempo o som, porém ele tem um fundo branco, bordas arredondadas, botões pretos, o que começa a divergir essa aparência entre um navegador e o outro e isso não é legal. Queremos uma consistência de design na nossa aplicação para os nossos usuários. E esse tipo de tag é muito difícil customizar ou não é possível customizar com CSS diretamente.

[03:49] Portanto, precisamos passar o controle da reprodução do som que está na tag áudio para os nossos botões, e é aí que começamos a precisar utilizar uma linguagem de programação que trabalhe com HTML e que trabalhe com CSS, que é o JavaScript.

[04:11] Agora vamos voltar para o VS Code e vamos retirar esses atributos controls da tag áudio, não vamos precisar. E vamos até ao botão, o primeiro botão do nosso Alura Midi, a tecla Pom, o botão da tecla Pom.

[04:25] E para fazermos alguma coisa com JavaScript nesse botão, podemos utilizar o atributo HTML aqui na tag button chamado onclick. O atributo onclick, como alguns vários atributos HTML, também tem o sinal de igual e aspas duplas, onde dentro das aspas duplas eu já posso escrever qualquer código JavaScript. E vamos começar com um código bem simples, só para testarmos esse dinamismo, coisa que o JavaScript é capaz de adicionar a uma página.

[05:03] Então eu vou usar uma função muito famosa do JavaScript chamada alert, um alert cria uma janela com uma mensagem, onde podemos dar o OK para essa janelinha, e ela é muito utilizada em várias aplicações para testar, para mandar mensagens para os usuários.

[05:22] Então eu escrevo dentro das aspas alert(‘’), precisa ser aspas simples para não entrar em conflito com aspas duplas do HTML. E dentro das aspas simples eu vou escrever uma mensagem. Como eu estou clicando em uma tecla de um som, eu vou escrever dentro Pom, que é o som do instrumento que está nesse botão.

[05:49] Então agora eu vou salvar o meu HTML com “Ctrl + S” e vou retornar o meu navegador. No meu navegador do Chrome eu vou recarregar e vou clicar na tecla Pom.

[06:03] Ao clicar na tela Pom eu já consigo visualizar o alert, aquela janelinha com o ok. E vamos ver como isso aparece no Firefox? Então vou abrir o Firefox, vou recarregar e vou clicar na tecla Pom.

[06:18] E a janelinha do alert já apareceu. E essa janelinha é o nosso primeiro código JavaScript que fazemos nesse curso, adicionando dinamismo a nossa aplicação. Claro, ainda está bem longe da nossa proposta, porém já é o nosso primeiro passo para saber o que o JavaScript faz. Então agora no nosso próximo vídeo você vai saber como eu faço um código ainda melhor para conseguir adicionar algum comportamento com o JavaScript.

**CONECTAR JS COM HTML**

00:00] Agora já sabemos como colocar um Alert dentro de um onClick e reproduzir um primeiro código JavaScript bem simples. Porém, se olharmos no nosso HTML, fizemos um código dentro da tag e escrevemos JavaScript dentro da tag do HTML.

[00:22] E se lembrarmos do CSS, isso é muito próximo com o CSS inline, que já sabemos que não é muito bom fazer o CSS inline, justamente porque fica difícil reproduzir os mesmos estilos para outras tags, se eu quiser modificar alguma coisa, eu vou ter que mexer em todas as outras tags, correndo o risco de cometer algum erro. E é a mesma coisa com JavaScript. Acabamos de fazer JavaScript inline através do atributo onClick.

[00:52] Porém, imagina se tivéssemos 88 teclas, como um piano completo ou outros instrumentos. Então ficaria muito difícil reproduzir essa lógica se tivéssemos que colocar elemento por elemento, tecla por tecla do nosso instrumento e se tivéssemos que fazer uma modificação.

[01:14] Então sabemos que o JavaScript inline não é legal, apesar de podermos e conseguirmos fazer para fins de teste e conhecer as potencialidades do JavaScript no HTML. Sem contar que cada linguagem tem a sua responsabilidade, o HTML tem responsabilidade de fazer a estrutura semântica da página, o CSS de fazer os estilos da página e o JavaScript vai ser o responsável por adicionar o dinamismo e as atualizações de programação, a lógica na página, utilizando essa lógica de programação que não é fornecida dentro do HTML.

[01:57] Portanto, como que podemos então fazer o JavaScript de uma maneira consistente e correta? Com o arquivo externo, assim como o CSS tem um arquivo CSS, o HTML tem um arquivo HTML, também vamos ter um arquivo JavaScript. Então eu vou criar dentro da minha estrutura de arquivos, no nosso Visual Studio Code, eu tenho um atalho, “New File”, é um ícone com um mais, uma folhinha e o +, “New File” ou novo arquivo, o meu está em inglês.

[02:34] E o nome desse arquivo novo que eu vou criar vai se chamar “main.js”. Vou dar o “enter” para confirmar a criação deste arquivo e ele já entra nesse arquivo novo criado ponto JS.

[02:52] Esse nome main, do inglês significa principal e temos uma convenção dos desenvolvedores de JavaScript em utilizar esse nome para o arquivo principal de JavaScript sempre para facilitar em todo o site ou aplicação escrita com JavaScript, facilitar qual é o arquivo que da entrada a toda a lógica e todos os arquivos JavaScript. Portanto vamos seguir essa convenção utilizada pelo mercado.

[03:22] E como eu faço agora para ligar esse arquivo JavaScript dentro do HTML? Então no meu HTML eu tenho a tag head que já tem tags que ajudam a configurar a minha página, por exemplo metadados, temos o título, links com ícones, com CSS, com fontes e também podemos ter a tag script, que é a tag responsável de carregar o JavaScript dentro do nosso HTML. É legal você sempre utilizar essa ajuda do VS Code para escrever script, para garantir que você não cometeu nenhum erro de digitação. Então vou dar o “Enter”.

[04:06] E assim como a tag IMG, a tag áudio, a tag script tem o atributo SRC, que é um atributo que significa source, ou seja, recurso que vai ser carregado por essa tag. Então eu vou usar o sinal de igual, aspas duplas e dentro das aspas duplas eu vou indicar o caminho desse arquivo, que vai ser o nosso “main.js”, e também já está sendo indicado pelo VSCode qual é o arquivo.

[04:34] Então vou dar um “Enter”, vou salvar o meu arquivo HTML e agora como eu garanto que esse arquivo foi corretamente carregado e está funcionando no meu HTML? Bom, da mesma forma que usamos o alert para conhecer o onClick no botão, eu também posso utilizar ele dentro do meu arquivo “main.js” para testar o funcionamento desse arquivo.

[05:02] Então alert (‘Olá mundo’);, vou escrever essa mensagem e no final, depois do parêntesis, vou colocar ponto e vírgula. O ponto e vírgula no JavaScript é opcional, ele funciona na maioria dos casos, porém é altamente recomendado utilizar o ponto e vírgula para evitarmos alguns erros no futuro. Então eu vou salvar o arquivo js com o atalho “Ctrl + S” e vou agora voltar para o meu navegador.

[05:51] Voltando no navegador, eu vou recarregar a página e uma mensagem com a caixinha do alert de primeira já é exibida para nós, “Olá mundo”.

[06:02] No Firefox fica tudo cinza e a caixinha cinza com o botão Ok, e depois aparece o nosso Alura Midi. E no Chrome também eu vou recarregar e a página em branco com a caixinha do Chrome com botão azul “Olá mundo”, confirmando que o nosso arquivo “main.js” foi devidamente carregado na nossa aplicação no Midi.

[06:28] Agora olhando o nosso arquivo “main.js”, o nosso HTML, temos um desafio, um desafio inclusive para o nosso próximo vídeo, que é como eu faço para pegar esse botão que está dentro do HTML e chamar dentro do “mains.js” e fazer alguma coisa com esse botão. Então no nosso próximo vídeo vamos descobrir isso.

BUSCAR UM ELEMENTO

[00:00] Agora precisamos descobrir como vamos colocar um botão do HTML dentro do JavaScript. É muito parecido com o que já fizemos no CSS.

[00:13] Vamos utilizar no navegador aquela ferramenta que auxilia os desenvolvedores para nos ajudar a fazer essa tarefa, que é o DevTools. Então clica com o botão direito na minha página e posso usar a opção de Inspect, no meu está em inglês, no de vocês pode ser “inspecionar”, “inspecionador de elementos”, “inspect element”, depende bastante de navegador para navegador, então eu vou clicar nessa aba e vai abrir para mim o DevTools.

[00:48] Pode ser que abra para vocês embaixo, eu gosto muito de trabalhar com ele no lado direito, então clico nesses três pontinhos e coloco ele pra direita. E também eu vou aumentar a letra para vocês poderem enxergar um pouquinho melhor.

[01:09] Sempre começamos a trabalhar com esse DevTools, essa ferramenta de desenvolvedores, na aba “Elements”. Essa aba contém todos os nossos elementos HTML do nosso “index.html” e embaixo temos essa aba “Styles”, que tem o CSS de cada elemento, como já vimos também nos cursos de CSS, no dia a dia desenvolvimento front-end, que conseguimos ver o CSS, os estilos das nossas tags.

[01:38] E para conseguimos selecionar e adicionar o estilo em um botão, por exemplo, precisamos criar uma classe como boa prática mesmo do CSS, que é criar uma classe para adicionar os estilos, poderíamos ter estilizado direto a tag pelo nome Button, e também é possível fazer com o ID dela. O correto é sempre usar a classe, o nome da classe ou o nome do elemento.

[02:11] E para o JavaScript não é diferente, precisamos ter um nome para encontrar esse elemento que queremos fazer alguma coisa dentro do JavaScript. E no DevTools vamos utilizar a aba “Console”, que fica do lado da aba “Elements”, que nele vamos poder executar qualquer código JavaScript dentro desse arquivo “index.js”, que está carregado nesse momento. E isso vai nos ajudar a desenvolver, a entender o nosso código.

[02:47] Seguindo essa lógica parecida com CCS, a primeira coisa que precisamos é ter nome do seletor, que pode ser o nome da tag, pode ser o nome da classe, pode ser um ID, enfim, qualquer seletor CSS. Eu vou usar por enquanto o seletor do nome da tag chamado “tecla Pom”. Então se eu escrever aqui ‘tecla\_pom’ e dar o “Enter”, vai me retornar ‘tecla\_pom’.

[03:19] Eu estou usando o Chrome. Então não teve nenhum retorno, nenhum problema, se tivesse algum com problema nessa escrita que eu teria feito, já saberíamos, porque o console também nos retorna todos os problemas do nosso código. Então ‘tecla\_pom’ é um seletor válido para nós.

[03:42] Só que só esse ‘tecla\_pom’ é basicamente um texto, é a mesma coisa que eu escrever entre aspas o meu nome, por exemplo, Vanessa ou Alura. É um texto. Eu preciso passar esse nome, esse texto, para alguém, algo que consiga fazer essa busca por esse elemento através desse seletor informado. Então eu vou usar essa funcionalidade de limpar o console, vou clicar nesse botão “Clear console” e vou digitar e apresentar para vocês uma função que é capaz de buscar e trazer para nós o elemento que informamos o seletor desse elemento, que é a função querySelector.

[04:36] Então a função querySelector, com q minúsculo e S maiúsculo, faz uma busca pelo seletor e é exatamente isso que significa o nome dela. Query em inglês, uma das traduções é busca, e selector, seletor, o mesmo seletor de CSS. Como é uma função, lembra como já vimos o alert? Ela tem essa sintaxe padrão. Tem que vir com o parêntese depois e entre parênteses, aspas simples, porque vamos passar um texto para ela e vamos passar o nome do seletor.

[05:14] Como o seletor é um nome, é um texto e eu preciso decidir se é uma classe, se vai ser o nome da tag, se vai ser o ID é igual lá no CSS. Se for o nome da tag, é direto o nome da tag button, se for uma classe ponto e o nome da classe, .tecla\_pom, se for um ID, eu vou começar com hashtag #som\_tecla\_pom, que é o ID de um outro elemento, se abrimos a aba elements vamos ver que por exemplo, é o ID de uma das tags áudio.

[06:02] E então só informei IDs que existem. Então eu vou aqui selecionar o botão, porque é o botão que eu quero acessar, é esse botão da tecla pom, então querySelector(‘.tecla\_pom’). E se eu der um “Enter”, eu vou ter o meu primeiro erro no JavaScript.

[06:26] Esse erro significa que querySelector não está definido, é um erro referência. Porque diferente do alert, que é uma funcionalidade que está em um escopo maior, que é a nossa janela, podemos chamar o alert em qualquer lugar do nosso código e ele é uma funcionalidade que está ali disponível para qualquer parte, qualquer momento do nosso código e para qualquer situação.

[06:55] Agora, o querySelector é uma funcionalidade que precisa ter um escopo, eu preciso dizer para o JavaScript aonde que eu quero que seja buscado este elemento, com este seletor. Então eu sempre tenho que executar o querySelector dentro de um outro lugar que precisamos informar, que pode ser uma outra tag ou geralmente usamos dentro do nosso documento HTML. Eu peço para o JavaScript, dentro do documento HTML, buscar o elemento que tem este seletor que é a classe ‘tecla\_pom’.

[07:37] Então para fazer isso, o JavaScript tem uma palavra reservada, chamada document. Essa palavra ela representa todo o documento HTML, inclusive acabei de escrever e toda minha página ficou destacada com azul, como se eu tivesse passado o inspecionador de elemento, aquela setinha e tivesse selecionado todo documento HTML.

[08:05] Representa todo esse documento. E agora para eu executar alguma coisa dentro desse documento, eu vou usar um caractere que acessa esse documento, que entra dentro desse documento, dessa palavra, que é o ponto. O ponto é capaz de entrar dentro dessa representação do document e acessar atributos, acessar funcionalidades que são possíveis serem executadas dentro do documento, e uma delas é o querySelector.

[08:44] Então querySelector(‘’) sempre com Q minúsculo S maiúsculo, parênteses, abre e fecha parênteses, dentro dos parênteses abre e fecha aspas simples, então document.querySelector(‘.tecla’) tecla já funcionou. Se eu escrever só tecla, porque todos os meus botões tem duas classes, tem a classe tecla e tem a classe tecla\_som.

[09:11] Então a própria classe tecla já funciona, já está até destacando no nosso HTML o elemento correto, embaixo já está dando uma prévia do que esse código que eu escrevi que o HTML vai retornar, então já disse ali que vai ser a tecla pom, ‘tecla\_pom’ vai retornar a primeira.

[09:38] Se eu colocar \_clap vai retornar a segunda. Então vamos pegar a Pom primeiro, vou dar um “Enter” e com o “Enter” eu confirmo a execução desse código JavaScript que vai trazer para mim a representação, a referência daquele botão. Então esse código acessa o documento HTML, ele busca aquele elemento daquele seletor informado e retorna para mim essa referência desse elemento, esse botão que eu quero trabalhar.

[10:11] E só mais um detalhe, que se eu colocar um que não existe, por exemplo tum, tum não existe, se eu der “Enter” ele vai retornar null, ou seja, nulo, não encontrou.

[10:23] E outro detalhe, se eu escrever queryselector com S minúsculo, vamos ter um erro também, porque ele vai dizer que query selector todo minúsculo não é uma função, não existe, porque o JavaScript é sensível a letras maiúsculas e minúsculas, então temos que ter muita atenção na hora de escrever. Às vezes escrevemos tudo minúsculo ou até um erro de digitação, esquecemos de escrever uma letra da palavra, escrever query seletor e não selector, também vai dar um erro. Então estou tentando fazer o erro aqui para vocês.

[11:06] Vai dar o mesmo problema. Então esse erro é porque aconteceu alguma coisa na hora de digitar, tanto o query selector ou document, por isso sempre utilize o auto complete, ou seja, essas sugestões que tanto o navegador ajuda para digitarmos o código corretamente quanto no Visual Studio Code.

[11:29] Sabendo que esse código está correto, eu vou copiá-lo mesmo se quiser, e colar no nosso “main.js”, posso copiar com o mouse, ou posso, se quiser praticar mais um pouco, digitá-lo aqui no “main.js”. Então voltando no VS Code agora, eu vou apagar aquele alert, eu não preciso mais dele, e vou digitar document.

[11:55] Olha o Visual Studio Code me ajudando a digitar, document.querySelector(‘.tecla\_pom’); ele até adivinhou que eu ia usar o querySelector, dou um “Enter” para ele auto completar.

[12:20] Então sabemos que esse código quando for lido pelo JavaScript vai trazer para mim um elemento HTML, no caso ali com aquele seletor, é o nosso botão. Eu vou salvar “main.js”, vou recarregar e nada acontece, nenhum problema, nenhuma letra vermelha no console.

[12:40] O nosso botão Pom ainda não reproduz nada e no Firefox, então eu vou abrir o Firefox. E no Firefox também vou recarregar, vou clicar com o botão direito, vou abrir o DevTools, vou abrir a aba console, vou ver que não tem nenhum erro, vou clicar na tecla Pom, recarregar aqui e vejo que não tem nenhum problema, então estamos corretos.

[13:09] Já temos o acesso agora a esse botão através desse código, porém ainda não fazemos nada com ele, então para nossa próxima aula vamos descobrir como adicionar uma funcionalidade dentro desse botão e reproduzir o som do nosso instrumento musical. Até lá.

A ferramenta DevTools presente nos navegadores web é de grande ajuda para comunidade de desenvolvedores web, a ponto de serem apelidados carinhosamente de “melhor amiga/o” da pessoa desenvolvedora.

Marque as alternativas com afirmações verdadeiras a respeito da ferramenta DevTools.

Alternativa correta

Através dessa ferramenta, conseguimos editar a aplicação em tempo de execução, adicionar ou remover tags HTML, mexer no CSS da página, escrever código JavaScript no console e ver as modificações sem persistência.

Alternativa correta! É isso mesmo! Com essa ferramenta, conseguimos realizar modificações em tempo real tanto de HTML, como de CSS e JavaScript, porém sem alterar o código fonte real.

Alternativa correta

Diferente do método tradicional para resolver problemas do código fonte que consistia em abrir o código e olhar linha a linha para descobrir o erro, o DevTools apresenta uma maneira mais simples e direta do problema no código, bastando acessar a aba Console que lá estará indicando o tipo e o arquivo onde se encontra o erro.

Alternativa correta! Como conseguimos mexer em tempo real no código da aplicação através do navegador, uma prática muito comum, principalmente se tratando do CSS, é ir escrevendo as propriedades e vendo como o site se comporta, para aí depois, colocar o código final no código fonte da aplicação.

FAÇA COMO EU FIZ

Faça como eu fiz: Brinque com seletores no JS

PRÓXIMA ATIVIDADE

Praticar ajuda bastante no aprendizado de um novo conceito, assim, é muito importante que você esteja sempre praticando. Pensando nisso, gostaria de propor este pequeno desafio que envolve o que foi visto em aula.

A partir deste código a seguir:

HTML

<h1> Alura Fone </h1>

<section class="teclado">

<input type="button" value="1">

<input type="button" value="2">

<input type="button" value="3">

<input type="button" value="4">

<input type="button" value="5">

<input type="button" value="6">

<input type="button" value="7">

<input type="button" value="8">

<input type="button" value="9">

<input type="button" value="\*">

<input type="button" value="0">

<input type="button" value="#">

</section>

<input type="tel" placeholder="Digite seu telefone">COPIAR CÓDIGO

CSS

\* {

box-sizing: border-box;

}

body {

display: flex;

justify-content: center;

align-items: center;

flex-direction: column;

background-color: #191919;

min-height: 100vh;

}

input {

border: none;

color: inherit;

font-size: inherit;

font-weight: inherit;

font-family: inherit;

}

h1 {

font-family: 'Montserrat', sans-serif;

font-size: 24px;

color: #fff;

}

.teclado {

display: grid;

grid-template-columns: repeat(3, 1fr);

gap: 10px;

background-color: #cccccc;

border-radius: 30px;

padding: 10px;

}

input[type=button] {

border-radius: 20px;

cursor: pointer;

font-family: 'Montserrat', sans-serif;

height: 80px;

width: 80px;

}

input[type=tel] {

background-color: #cccccc;

border-radius: 10px;

font-family: monospace;

font-size: 22px;

height: 40px;

margin-bottom: 24px;

padding: 10px;

text-align: center;

width: 280px;

}COPIAR CÓDIGO

Escreva o código em JavaScript capaz de selecionar o elemento HTML input do tipo telefone:

Como fazer:

Abra o editor de código de sua preferência;

Crie dois arquivos: desafio.html e desafio-estilo.css;

Copie o código acima e cole dentro da tag < body> do arquivo chamado desafio.html e o código CSS no arquivo desafio-estilo.css;

Para resolver este desafio, você pode usar apenas a aba Console da ferramenta DevTools;

VER OPINIÃO DO INSTRUTOR

Opinião do instrutor

Gabarito:

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

Para selecionar o elemento HTML input do tipo tel podemos utilizar o código document.querySelector(‘input[type=tel]’), porquê com essa linha, o JavaScript vai acessar o documento HTML (representado por document) e procurar pelo elemento input que tenha como atributo [type=tel], os colchetes fazem parte da sintaxe para seleção de atributos de um elemento HTML, funciona tanto para o CSS quanto para o HTML.

Para que fique mais claro o porquê de se utilizar arquivos separados para HTML e JavaScript, leia o artigo da instrutora Rafaella Ballerini, que explica a diferença entre essas linguagens.

Artigo: HTML X CSS X JavaScript. Quais as diferenças?

O QUE APRENDEMOS

Nessa aula, você aprendeu:

* Clicando no botão:

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.

* Conectar JS com HTML:

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.

* Buscar um elemento:

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 virgula é opcional porém recomendado.

FUNÇÕES

[00:00] Na aula anterior aprendemos como trazer um botão que está no HTML para dentro do JavaScript, e portanto assim vamos conseguir adicionar um clique e fazer as demais funcionalidades que precisamos para funcionar o nosso Alura Midi. Porém, agora precisamos descobrir como fazemos para reproduzir um som que está carregado dentro de uma tag áudio.

[00:28] Acessando o nosso navegador, eu estou no console, vou abrir a aba “Elements” e a aba de elementos tem a nossa tag áudios. Lembrando que a nossa tag áudios só não está sendo exibida porque omitimos o atributo controls, porém este atributo, essa tag contém todos os controles necessários para reprodução de uma mídia, como um som. Portanto vamos ter disponível também, via JavaScript esses controles para reprodução do som.

[01:02] Então seguindo a mesma lógica do botão, precisamos acessar essa tag áudio para então conseguir acessar os controles dela. Então vamos usar o document, porque queremos procurar o elemento dentro do documento, utilizando a função, então document.querySelector(‘’), que é a função que busca um elemento através do seu seletor, então abre e fecha parênteses e abre aspas simples dentro dos parênteses.

[01:33] Podemos olhar na aba “Elements” novamente qual é o seletor que queremos usar, podemos usar tanto o nome da tag áudio quanto o ID dela. E para sermos mais certeiros, o ideal é utilizarmos então o ID dessa tag. Então vou colocar para o seletor começar com ID, então #som\_tecla\_pom.

[02:00] Então ao dar “Enter” eu vou ver que realmente eu localizei e consegui trazer com o JavaScript a referência desse elemento áudio. Agora que eu sei acessá-lo é muito fácil entrar dentro dele e encontrar os controles de reprodução da mídia que ela está carregando.

[02:20] Portanto eu posso repetir esse mesmo código, document.querySelector(‘#som\_tecla\_pom’) e esse código representa uma tag, um elemento áudio, portanto, se logo após os parênteses do querySelector eu colocar um ponto, eu estarei acessando este elemento, estarei entrando dentro dele e conseguindo acessar todas as funcionalidades que estão disponíveis dentro desse elemento áudio, que está sendo retornado pelo querySelector.

[03:04] Então como é uma tag, um elemento de multimídia, eu vou ver que quando eu cliquei o ponto ele já abriu uma janelinha embaixo no console com uma lista de coisas, de atributos, de funcionalidades disponíveis para esse tipo de elemento. E com a setinha do teclado para baixo eu consigo ver e navegar todas as coisas que tem dentro dessas funcionalidades ou atributos e eu percebo que existe uma chamada Play.

[03:34] E é justamente essa que dá o play no nosso som. E eu sei que é uma funcionalidade, porque quando selecionei Play, eu vi que logo embaixo tem um F desenhado, isso significa que é uma função. Então se é uma função, tenho que usar sintaxe de função para reproduzir. Abre e fecha parênteses.

[04:00] Eu não preciso passar nada entre parênteses para que ela funcione, eu só vou apertar o “Enter” e esse código vai ser executado e vamos escutar o som do pom. Então já sabemos como reproduzir o som de uma tag áudio com JavaScript, porém esse retorno que veio, que é uma promise pending, não precisamos nos preocupar e não vamos usar isso. Então vamos seguir agora com esse código dentro do nosso “main.js”.

[04:36] Então do nosso “main.js” eu vou dar “Enter” e vou escrever o código que fizemos e executou lá no nosso console. Então document.querySelector(‘#som\_tecla\_pom’).play();. Vou salvar e vou abrir o meu navegador e vou recarregar. O que esperar dessa execução?

[05:20] Então recarreguei e recebi um erro. Gente, está tudo bem receber erros, temos que estar preparados e de coração aberto para conseguir tratar esses erros.

O QUE E UMA FUNÇÃO

[00:00] Eu recarreguei o Chrome e no console eu já obtive um erro. Não precisamos ter medo de erros, precisamos aprender a lidar com os erros e vamos ver que eles nos ajudam a entender o nosso código.

[00:13] Então ele está falando que não consegue ler as propriedades de nulo e ele está tentando ler uma propriedade Play. Outro lugar que podemos conseguir entender melhor nosso erro é abrir outro navegador. Eu estou no Chrome, então vou abrir o Firefox, vou recarregar o meu navegador e ele vai apresentar o mesmo erro, só que um pouco diferente.

[00:33] Então document querySelector is null e está falando que está na linha 3 do nosso arquivo “main.js”. No Chrome também está falando que é na linha 3. Se abrirmos a linha 3, está o nosso código recém-escrito, que document querySelector é nulo.

[00:53] E o que acontece? Ele está dizendo que essa execução não existe e retorna nulo, esse código retorna nulo, portanto, ele não consegue executar o play de alguma coisa que é nula, que não existe. E esse erro está propositalmente aqui para entendermos uma coisa muito importante, que é o local correto do arquivo “main.js”.

[01:20] O navegador lê o nosso arquivo HTML linha por linha e já vai chamando e executando as coisas que são escritas, declaradas no arquivo. Então no momento que o navegador passa na linha 18 e entra no arquivo “main.js” para executar esses códigos e esses códigos dependem do nosso body, esse momento da linha 18, o navegador ainda não chegou no body e não construiu o body, portanto temos um código que depende dessa estrutura do HTML abaixo e que ela não está existindo ainda nesse momento.

[02:06] Então aqui tem uma lição muito importante sobre o local da tag script. Conseguimos colocar a tag script tanto no head quanto no body e em qualquer lugar, mas para decidirmos se vamos colocar no head ou no body, é que se o nosso JavaScript depende da estrutura HTML carregada e pronta, devemos colocá-la dentro do body e ao final do body, antes logo antes do fechamento da tag body e não dentro da tag head. Vamos fazer isso.

[02:43] Então vamos tirar, vou recortar da tag head e vou colocar logo antes da tag body e vou salvar. Se meu arquivo JavaScript tiver um monte de funcionalidades que não dependem do que tem declarado dentro da tag body, então eu posso colocar sim dentro da tag head.

[03:04] Então eu vou salvar, vou voltar no meu navegador e conseguimos resolver o problema do nulo, agora temos um outro problema. E às vezes quando estamos fazendo JavaScript é assim mesmo, temos um problema atrás do outro e isso é muito legal, que vamos aprendendo como a linguagem funciona. Então vou recarregar aqui no Chrome, também no Firefox e eu vi que o erro também aconteceu sobre o play que estamos fazendo.

[03:36] Tem uma coisa muito peculiar da natureza da nossa aplicação, porque precisamos executar um som e o JavaScript, como eu falei para vocês, ele vai sendo lido e executado imediatamente pelo navegador, portanto, o que acontece? Imagina que você abriu um site e quando você abriu o site, você nem fez nada, o site está carregando e já começa a tocar um som, um monte de sons indesejados. Então os navegadores têm uma política de bloquear esse tipo de código, que já executa mídias e sons antes do usuário interagir com a página.

[04:25] Então é esse erro que está acontecendo, há um problema com a permissão e controle dos usuários. Então mesma coisa no Firefox, ele não permite que demos o Play pelo JavaScript antes do usuário fazer algum tipo de atividade na nossa página, interagir com a nossa página. E realmente não faz sentido, não queremos que os sons do nosso Midi sejam reproduzidos ao abrir a página, só queremos que seja reproduzido quando clicamos no botão.

[05:02] Então vamos ter que fazer alguma coisa para guardar esse código do play e ele ser chamado somente quando eu quiser. E quando eu preciso de um código que é chamado somente quando eu preciso, precisamos criar uma função. Então agora vamos abrir nosso arquivo “main.js” e vamos ver como fazemos para criar nossa própria função.

[05:31] Porque até agora usamos funções prontas que já vem no JavaScript ou nos elementos. Então eu vou voltar para minha linha 1 e vou empurrar o meu código mais para baixo dando “Enter” e escrever na linha 1 uma declaração de função, escrevendo a palavra reservada do JavaScript function.

[05:54] Quando eu digito function ele já fica até com uma outra cor e em itálico, justamente identificando que é uma palavra reservada. E quando eu crio uma função, eu preciso dar um nome para essa função. Lembrando que a nossa função vai armazenar esse código que toca o som de algum instrumento, de alguma tecla nossa, portanto vai ser uma função que toca um som. Eu vou colocar o nome dessa função de tocaSom e como ela vai tocar o som da tecla pom, todo instrumento pom, eu vou colocar tocaSomPom para ser mais específica.

[06:38] E continuando na sintaxe de criação de uma função, eu preciso declarar parênteses, abre e fecha parênteses e depois do fechamento dos parênteses, abre e fecha chaves. Dou um “Enter” dentro das chaves e tudo que vai dentro das chaves vai ser o código que vai ser guardado dentro dessa função para ser executado apenas quando a função for chamada por alguém.

[07:07] Então agora eu vou transferir esse código que está como play, posso arrastar com o mouse ou dar um “Ctrl + X”, “Ctrl + V” e vamos salvar o nosso código, posso tirar esses espaços que colocamos, vou salvar esse código e vou voltar para o navegador e vamos ver se aquele erro continua. Vou recarregar. O erro saiu no Firefox e lá no Chrome também saiu.

[07:40] Mas e agora, o que acontece? Eu clico na nossa tecla pom e nada acontece. O que acontece é que a nossa função foi criada, ela existe e ela já está disponível para ser utilizada, inclusive de dentro do console. Se eu pegar e digitar no nosso console do navegador “tocaSom...”, ele até completa para nós, porque ele percebe que dentro do “main.js” já tem funções disponíveis para nós. Então tocaSomPom e lembra, o f significa que é uma função, abre e fecha parênteses para executar tocaSomPom(), quando der “Enter” vai dar play no nosso som.

[08:22] Tocou. Esse undefined que apareceu, vocês não precisam se preocupar também, aparece porque o retorno não está definido e não vamos trabalhar com isso, então está tudo certo. Nossa função já está funcionando, falta só um pequeno detalhe, que vamos ver no nosso próximo vídeo, que é ligar o clique com o botão que já trouxemos para o JavaScript com a função que reproduz o som daquela tecla. Então falta muito pouco para concluirmos a nossa atividade principal do Alura Midi. Até logo.

Como vimos nesta aula, é possível inserir um arquivo JavaScript em dois locais do documento HTML, dentro da tag head ou da tag body.

O script depende de elementos do body já carregados pelo navegador.

Alternativa correta! Quando o código JavaScript que vamos adicionar depende de elementos que estão no body, por exemplo um querySelector que espera receber determinado seletor, o local mais apropriado para sua inserção é dentro da tag body e na linha anterior ao fechamento da tag </body>. No head podemos inserir scripts, porém eles não devem ter dependencias de elementos do body, por exemplo uma biblioteca de funcionalidades.

Faça como eu fiz: Para que serve uma função?

PRÓXIMA ATIVIDADE

Como vimos nesta aula, declaramos uma função no JavaScript quando precisamos guardar um código que será executado num segundo momento, ou seja apenas quando ele for chamado/invocado.

Agora declare a função tocaSomClap, e desenvolva dentro dela o código necessário para dar play no áudio do som clap.

Gabarito:

***function tocaSomClap() {***

***document.querySelector('#som\_tecla\_clap').play()***

***}COPIAR CÓDIGO***

Para declarar uma função no JavaScript, usamos a pavra function e depois o nome que escolhemos, no caso tocaSomClap seguido da sua sintaxe padrão, que são os parenteses e depois chaves.

O espaço dentro das chaves é conhecido como bloco ou corpo da função, e é nele que colocamos o script que desejamos executar quando a função for chamada.

Como queremos executar o som do clap, devemos localizar o elemento audio pelo seu seletor id: #som\_tecla\_clap.

***document.querySelector('#som\_tecla\_clap')COPIAR CÓDIGO***

Este código irá retornar o elemento, portanto logo em seguida já podemos usar o operador ponto para entrar dentro deste elemento e já acessar a função play() que é nativa dos elementos audio do HTML.

***document.querySelector('#som\_tecla\_clap').play()***

CLIQUE NO BOTÃO

[00:00] Como vimos no vídeo anterior, agora já sabemos como pegar um botão com JavaScript e também já sabemos como reproduzir o som de uma tag áudio quando quisermos, através da nossa própria função. Agora falta só um único detalhe, que é como vamos unir o botão que vai chamar a função que vai dar play e executar o som do nosso instrumento. Precisamos no meio de alguma coisa que vai chamar essa função.

[00:34] E já sabemos, na verdade, o que é. Sabemos que é esse momento do usuário clicar em cima do botão do Midi. E na verdade até já vimos como fazer isso, de como fazer alguma coisa quando o usuário clica no botão. Então abrindo o VS Code, se formos no nosso elemento button e usar aquele e aquele atributo onclick que vimos que não é legal fazer, mas os usarmos só para testar a nossa função tocaSomPom.

[01:14] Então digitei onclick="tocaSomPom()" salvei o HTML e voltamos no navegador e recarregarmos, eu vou ver que ao clicar no botão pom já vai funcionar.

[01:31] Então já fizemos tudo isso, porém sabemos que JavaScript inline não é legal por vários motivos que não precisamos repetir, manutenção, difícil escalabilidade, mesma coisa que o CSS, então precisamos fazermos esse onclick lá de dentro do arquivo “main.js”, que é onde estamos mantendo as nossas boas práticas de JavaScript.

[02:00] Lembrem que criamos aqui um código document.querySelector, passando o seletor da tecla pom, que é a classe tecla\_pom. E sabemos que esse código vai trazer o botão da tecla Pom. Se estamos com dúvida, eu posso até copiar esse código, vou copiar aqui, voltar lá no navegador, abrir a aba console colar e dar “Enter”. Só para garantir que é desse button com som pom. Vou recarregar para tirar o onclick.

[02:43] E da mesma forma que conseguimos acessar com querySelector, o elemento da tag áudio através do ponto. No “main.js” eu posso tirar o ponto vírgula e continuar o código do nosso querySelector(‘. tecla\_pom’). usando o ponto.

[03:08] O ponto vai entrar para dentro daquele elemento que está sendo buscado no querySelector e vai trazer todas propriedades e funcionalidades que são possíveis ser consultadas ou executadas dentro daquele elemento. Então com um ponto o próprio Visual Studio Code abriu uma janelinha de coisas que são possíveis fazer.

[03:30] E o que queremos aqui agora é acessar o atributo onclick desse elemento, que sabemos que existe porque está lá no HTML, se podemos usar ele no HTML, também podemos usar ele aqui no JavaScript. Então eu vou escrever onclick. Escrevendo onclick eu vou ter acesso a esse atributo e como ele é um atributo, para eu passar e definir um valor para ele, eu uso o sinal de atribuição, o mesmo do HTML, que é o sinal de igual =.

[04:06] E esse sinal vai poder atribuir tudo que vem para o lado direito aquilo, nesta propriedade/atributo o que está do lado esquerdo do sinal. Portanto queremos que quando o onclick deste elemento da tecla pom seja executado, queremos que chame a função tocaSomPom();.

[04:33] Vamos salvar e ver se dá certo. Vou recarregar na minha página e eu tenho um problema. O problema do Play, que o usuário não interagiu com a página antes, e já vimos esse problema.

[04:48] Esse problema já havíamos corrigido, porém voltamos a tê-lo por culpa nossa mesmo. O que acontece no JavaScript? Quando atribuímos uma função para um elemento e, na verdade, quando escrevemos o nome da função e abrimos e fechamos parênteses imediatamente, não estamos guardando essa função dentro do atributo onclick, isso é uma peculiaridade do arquivo JavaScript. Já estamos dizendo para o navegador passar nessa parte e executar essa função imediatamente.

[05:31] Portanto, se dentro do arquivo “.js” queremos guardar uma função em um atributo onclick, precisamos retirar esse parênteses, porque só assim se ele vai guardar dentro do onclick a referência, esse nome da função e não vai necessariamente executá-la no mesmo momento em que ela é guardada. Tem um pouco a ver com a ordem de execução do código JavaScript.

[06:02] Então eu vou meu navegador, vou recarregar e vou ver que esse problema deixa de acontecer, porque agora não estamos chamando a função tocaSomPom, que toca imediatamente o play da tag áudio. Estamos realmente guardamos a função tocaSomPom dentro onclick do botão pom. Então se eu apertar o pom vai executar o som.

[06:31] Vamos ver no Firefox se tem algum problema. Recarreguei o Firefox, vou apertar o botão e o som tocou novamente. Agora para nossa próxima aula vamos aprender como reproduzir esse som de forma automática para todos os botões do nosso teclado, porque já conseguimos fazer funcionar para uma, mas como fazemos automaticamente para todos os nossos botões? Lembrem-se, imagine se tivéssemos um piano com 88 teclas? Até lá.

O que aprendemos?

PRÓXIMA ATIVIDADE

Nesta aula, você aprendeu:

Play no JS:

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.

O que é uma função?:

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.

Clique no botão:

A utilizar o onclick como atributo no JavaScript, e como atribuir uma função ao onclcik sem que esta função seja invocada imediatamente.

LISTA DE ELEMENTOS

[00:00] Olá. Bom, já vimos nas aulas passadas como conseguir reproduzir o som que está dentro da tag áudio e passar o controle para nossa tag button do Alura Midi. Agora temos um desafio de como fazer a mesma tarefa para as próximas teclas do nosso Alura Midi. Vamos voltar lá no nosso Visual Studio Code e a maneira mais simples ou com o que sabemos até agora, seria copiar esse código e renomear para cada tipo de som.

[00:38] Vamos fazer isso só para ver como vai ficar e o tamanho que vai ficar o nosso arquivo “main.js”. Eu vou fazer e vou mostrar para vocês como isso vai ficar no fim.

[00:55] Então basicamente vamos ter que copiar a função e o querySelector, tocaSomPom e vai ficar tocaSom..., vamos ver o outro instrumento, Clap, então tocaSomClap, document.querySelector(‘#som\_tecla\_clap’).play() e tocaSomClap. E vamos reproduzir isso mais sete vezes e vamos ver como vai ficar no fim.

(vídeo acelerado do código sendo digitado mais sete vezes)

[01:49] Agora conseguimos reproduzir o código que fizemos para todas as nossas nove teclas, olha como que está a situação desse código. Muito igual, muito parecido, já está salvo eu só vou recarregar meu navegador e vou mostrar para vocês.

[02:08] Já aconteceu problemas que eu digitei alguma coisa errada, “splah is not defined”. Vou até olhar o meu código. Olha como fica difícil encontrar onde está o erro dessa forma, nem consigo encontrar direito onde está o splash.

[02:29] Achei, faltou um S. Agora vou reproduzir os sons e vocês vão ver que está tudo funcionando, porém o nosso código está horrível, convenhamos.

[02:50] Bom, se não quisermos avançar o código e só quisermos organizar e nos ajudar a entender, um recurso que o JavaScript tem e todas as linguagens de programação tem, é um recurso de comentários, que é basicamente um texto, uma frase que você pode escrever e que não vai ser executada como um código JavaScript, ela vai ser ignorada e serve só para nós desenvolvedores podermos ter uma ajuda, nós mesmos do futuro, quando revisitamos esse código, daqui um ano, daqui meses, conseguir entender o que nós mesmo escrevemos.

[03:28] Então os comentários podem começar com barra barra //, que é o comentário em linha, podemos escrever por exemplo POM e podemos colocar umas linhas para separar e tem uma função do pom e tem o querySelector do botão pom.

[03:53] Agora do Clap eu posso usar um comentário em bloco, que é o barra asterisco /\* e então eu consigo dar um “Enter” e todas as linhas abaixo já ficaram cinzas, significa que a partir dali tudo está comentado, mas eu posso encerrar o meu comentário de bloco com asterisco barra \*/e entre esses dois símbolos, essas linhas são ignoradas, são consideradas como comentário.

[04:26] Também posso fazer o clap para me ajudar nesse código, entender esse código, posso ir dividindo o meu código com comentários para conseguir ajudar a entender e dar manutenção. Mas, de novo, entramos naquela mesma questão: temos comentários, temos o código funcionando, mas imagina se tivéssemos 88 teclas.

[05:02] Com 9 teclas já estamos com um código de 60 linhas e se começarmos a colocar os comentários, potencialmente já vai virar umas 90 linhas. Agora imagina se tivermos um monte de teclas, seria praticamente um horror conseguir trabalhar nesse código.

[05:19] Agora então que vimos que não compensa fazer repetição de código, até porque se precisarmos mudar alguma coisa na funcionalidade das nossas funções que tocam som, teríamos que mexer em pelo menos nove funções, precisamos aprender uma nova coisa do JavaScript, que é automatizar as nossas funcionalidades e principalmente trabalhar com muitos elementos de uma vez só, precisamos aprender a trabalhar com listas de elementos.

[05:52] E para isso, a primeira coisa que temos que fazer é ao invés de selecionar um único botão do nosso teclado, precisamos saber como selecionar todos os botões de uma vez e pegar todos eles de uma vez e colocar a função que toca o som. Então para isso eu vou usar o console do navegador, estou no Firefox, e no console vou digitar o nosso documento, document, porque dentro do meu documento eu quero fazer uma busca, então document.querySelector.

[06:31] Só que ao invés de eu usar agora o querySelector, que o querySelector busca o primeiro elemento que ele encontrar com aqueles seletor informado, eu vou usar o querySelectorAll. O querySelectorAll é uma funcionalidade do JavaScript que busca todos os elementos com aquele seletor que vocês passaram aqui entre os parênteses, então eu vou abrir aspas simples e vou passar o seletor que todos esses botões tem.

[07:06] Eu posso inspecioná-los na aba inspecionador e ver qual é. Sabemos que o nome da tag button não é um seletor muito bom, porque pode ser que eu tenha outros botões que não sejam do meu teclado no futuro da minha aplicação e pode dar algum problema. Eu quero garantir que sejam só os botões do meu teclado do Alura Midi.

[07:30] Então eu vejo que tem aqui uma classe tecla, que tem todos os meus botões. Eu vou usá-la como seletor de todos os botões. Então seletor de classe começa com .tecla e vou dar “Enter” para ver o que acontece.

[07:46] Dando “Enter” eu vou ver que a execução que acontece no console é trazer uma linha escrito node list e abre um número 9, tem um número 9 me indicando qual a quantidade de elementos que ele encontrou com esse seletor, abre um colchete e dentro desse colchete começa cada elemento button encontrado. Eu consigo passar o mouse em cima e ele destacar na nossa página cada um dos elementos que foram encontrados.

[08:19] Se eu clicar nesse triângulo que tem do lado esquerdo do node list, ele vai abrir e mostrar essa lista ainda com o índice do lado, 012345678, então são nove elementos começando do zero ao 8º, somando 9 e cada botão desse elemento.

[08:41] Eu vou executar agora o mesmo código no Chrome para vocês verem a diferença. Então vou recarregar, document.querySelectorAll(‘.tecla’), vou dar o “Enter”. Então também aparece o node list com 9 elementos e o triângulo para abrir e ver os índices dessa lista e também todos os elementos que estão dentro dela.

[09:14] Notem que a lista começa com colchete, então antes de abrir o triângulo vimos todos os elementos, em volta deles tem um colchete, mesma coisa no Firefox, se eu recolher esse triângulo eu consigo ver que elas estão em volta de um colchete. Isso é a característica que estamos trabalhando com listas.

[09:35] E agora precisamos saber como colocar o nosso código para funcionar junto com essa lista. Então no lugar de um querySelector de um botão, no “main.js” eu vou apagar todo esse código, então vocês não precisavam fazer essa repetição que eu fiz, era só para termos noção de quão caótico pode ser.

[09:57] E lá onde eu tenho o querySelector da tecla pom e o onclick eu vou apagar esse código sem dó, quem tiver dó pode comentar, que ele não vai ser comentário, mas não tenham dó de código, pode apagar, e vou usar o querySelector no lugar. Então document.querySelectorAll, agora vamos usar aqui a nossa nova função, querySelectorAll, abre e fecha parênteses, aspas, ponto tecla.

[10:31] E partir da próxima aula vamos ver como fazemos a mesma coisa que estávamos fazendo aqui com onClick agora trabalhando com uma lista.

Porque o document.querySelectorAll()?

PRÓXIMA ATIVIDADE

Como vimos durante a aula, apesar de utilizarmos o script document.querySelector(seletor) para capturar elementos do HTML individualmente, no Alura MIDI, trocamos ele pelo document.querySelectorAll(seletor).

Desta forma, o uso do document.querySelectorAll se justifica por:

Utilizamos o document.querySelectorAll() para selecionarmos todos os elementos do seletor desejado, reduzindo a quantidade de repetição de código, que consequentemente, promoverá uma otimização no código em alternativa a repetição.

Alternativa correta! Ao utilizarmos esse script, ao invés de capturarmos um elemento por vez, conseguimos pegar todos de uma só vez. Portanto isto facilitará a manipulação, reutilização e manutenção do nosso código para todos os elementos que recebem o mesmo tipo de função.

REFERENCIAS

00:00] Como vimos no vídeo anterior, conseguimos capturar todos os elementos do nosso teclado, todos os botões valor Alura Midi de uma vez só e obtivemos uma lista elementos. E agora precisamos começar a entender como essas listas funcionam no JavaScript e como podemos automatizar alguns códigos utilizando a lista.

[00:25] Mas primeiro de tudo precisamos deixar nosso código muito legível e muito fácil de entender. Vamos voltar lá no “main.js”, no VS Code e já colocamos o document.querySelectorAll, selecionando todos os elementos com a classe tecla.

[00:46] Porém se batermos o olho de primeira, às vezes fica difícil entender que por trás desse código existe uma lista e uma lista de elementos que compartilham essa mesma classe, portanto, uma coisa que poderíamos fazer para melhorar a legibilidade, o entendimento desse código, é criar alguns nomes que possam nos ajudar entender o código. O ideal seria se tivéssemos no lugar desse document.querySelectorAll, pudéssemos utilizar o português mesmo e algo como "lista de teclas", por exemplo.

[01:34] Que seria uma maneira de conseguir entender que eu sei que tem uma lista e que o que tem dentro dessa lista são teclas. E é possível fazer isso com o JavaScript, usando um recurso da linguagem chamado referências. As referências no JavaScript devem ser criadas ou declaradas com base no valor que elas vão receber e ter que guardar. Por exemplo, a nossa lista de teclas. Será que em algum momento, no nosso JavaScript, na execução do nosso JavaScript a lista de teclas vai se tornar outra coisa? Vai receber um outro valor?

[02:17] Muito provavelmente não e o ideal seria que realmente a lista de teclas sempre armazenasse a mesma lista de teclas, portanto, terá seu valor constante. Porém, se fosse um outro tipo de valor que fosse armazenado nessa referência, que pudesse ter seu valor modificado ao longo do script, essa outra referência poderia ser uma variável, que não é o nosso caso.

[02:47] E o ideal mesmo é usarmos sempre referências constantes, e caso necessitarmos de uma variável a utilizamos como variável. Então como eu crio uma referência constante no JavaScript? Já criamos o nome dela como listaDeTeclas, que seria o ideal e é isso mesmo, podemos criar com o nome que quisermos, porém antes colocamos a declaração dessa referência, no nosso caso vai ter um valor constante, então escrevemos const listaDeTeclas.

[03:31] Então por isso que é importante saber se ela vai ser uma constante, uma variável, porque isso define essa palavra que vamos escrever antes do nome da referência do lado esquerdo. E depois do lado direito temos o sinal de atribuição, que já conhecemos, que vai armazenar o que está do lado direito dentro dessa referência. Então queremos colocar a nossa lista de teclas que está dentro do document.querySelectorAll, armazenado dentro de lista de teclas.

[04:03] Então eu vou recortar o código ou podemos arrastar movendo e vamos colocar para o lado direito do sinal de atribuição. E desta forma conseguimos ter uma lista de teclas guardadas em um nome que faça sentido, que tenha significado, que seja fácil de entender. E a partir da lista de teclas que vamos começar a manipular nossa lista de botões nas nossas próximas aulas.

[04:30] Então inclusive, da forma que eu fiz e declarei, se eu for lá no navegador, então vou no Firefox, vou recarregar Se eu abrir o console e digitar “lista de teclas” ele já vai aparecer para mim até para auto completar, eu vou dar o “Enter” e ele vai aparecer aquela node list.

[04:52] Muito mais fácil trabalhar com lista de teclas do que document.querySelector todo tempo. Ótimo. Então agora que o nosso código está mais fácil de entender, no nosso próximo vídeo vamos conhecer um pouquinho mais de detalhes sobre as listas para definitivamente conseguirmos aplicar a funcionalidade de execução do som de cada instrumento para cada botão. Até lá.

CONHECENDO LISTAS

[00:00] Agora que já sabemos como deixar o nosso código mais legível, criando uma referência constante, que armazena todas as nossas teclas do nosso Alura Midi, nessa referência lista de teclas precisamos entender como eu acesso individualmente um elemento dessa lista.

[00:24] No nosso console do navegador temos disponível a referência a listaDeTeclas que contém todos os nossos elementos. Percebemos também que temo um colchete que representa essa lista. Só que a listaDeTeclas, quando eu coloco o ponto para entrar dentro dela, que nem já fizemos com o document ou com um elemento retornado no querySelector, percebemos que eu não consigo acessar, por exemplo, uma propriedade onclick para colocar nos elementos de dentro.

[01:10] Porque a lógica não é feita dessa maneira, eu não coloco na lista o evento de clicar, mas sim em cada elemento da lista. Portanto, apesar de eu ter conseguido concentrar todos os meus elementos em uma linha de código, eu ainda vou precisar acessá-los individualmente para conseguir em cada um deles atribuir um valor para o onclick, ou seja, uma função que vai reproduzir o determinado som daquela tecla. Portanto, eu não tenho mais a propriedade onclick logo em seguida como teria se fosse para acessar um elemento.

[01:58] Então precisamos saber como eu acesso um elemento que está dentro dessa lista. E para isso usamos essa sintaxe de colchetes que já vimos que é usada para lista. E dentro do colchete vamos passar o número com o índice que o elemento está dentro dessa lista. Então eu vou colocar um número, que por exemplo, eu quero o elemento do índice 0, que é o da tecla pom listaDeTeclas[0].

[02:30] Então perceba que já está destacado na nossa página o elemento que está armazenado no índice 0 dessa lista de teclas. Índice 1 um temos o Clap, 2 temos o Tim, 3 temos o Puff e assim até o número 8, representando todos os elementos da lista. Se colocarmos um número que não existe, vai retornar que está indefinido (undefined).

[03:03] Então se eu quiser voltar a ter o meu onclick no pom, eu preciso acessar este elemento pela lista, mas informando o número do seu índice para conseguir acessá-lo e então acessar o onclick. Lembrando que no node list, quando abrimos, conseguimos verificar o número do índice de cada botão, lembrando que esse índice é organizado pela ordem de declaração do HTML, então foi o primeiro elemento a ser declarado com essa classe foi o nosso botão pom, por isso ele ficou como primeiro elemento da lista com índice zero.

[03:47] É muito comum que as listas dentro da programação comecem com índice zero e vão até um índice de um número total -1. No Firefox eu vou clicar nessa lixeirinha que tem na aba console, que ela limpa o console para mim. E vou acessar novamente a minha lista de teclas e vou pegar aqui os colchetes, vou adicionar o colchete e vou colocar o índice 0 para acessar a tecla pom listaDeTeclas[0].

[04:24] Então se eu fizer isso e der “Enter”, já vai me retornar o elemento com a tecla pom. E então se eu fizer a mesma coisa e colocar o ponto logo em seguida dos colchetes, ou seja, lista de teclas, colchete, 0 dentro do colchete e após os colchetes eu colocar o ponto, eu vou voltar a acessar todas as propriedades funcionalidades de um elemento, porque agora a minha referência para este elemento é através da lista de teclas e eu não preciso mais usar o document. querySelector de um único seletor.

[05:00] E aqui eu consigo fazer o onclick e atribuir ao onclick a nossa função listaDeTeclas[0].onclick = tocaSomPom. Lembrando que eu não posso passar os parênteses, senão temos aquele problema do play, então vamos guardar aqui no onclick a referência àquela função que já declaramos. Então vou dar um “Enter” e vou clicar no botão pom para ver se vai reproduzir o som.

[05:29] Reproduziu. E esse código está só em tempo de execução do nosso navegador, agora que ele já está funcionando eu posso copiá-lo e colocar dentro do meu “main.js”, porque se eu recarregar a minha página eu vou perder essa execução e vai deixar de funcionar o nosso botão pom.

[05:57] Então eu vou fazer novamente no arquivo “main.js” no VS Code, que é listaDeTeclas[0].onclick = tocaSomPom;. Voltando navegador e recarregando nossa página, se eu clicar no botão pom vai reproduzir. Voltando ao código que tínhamos algumas aulas atrás, apenas um botão funciona e as outras não tem mais funcionalidade.

[06:37] E se eu fosse usar essa mesma lógica para colocar o som nos outros botões, eu teria que fazer algo parecido com isso, ou seja, listaDeTeclas[1].onclick e criar uma função tocasSomClap, listaDeTeclas[2].onclick = tocaSomTim e criar essa função. Porém sabemos que se eu criar todas essas funções, o meu código vai ficar enorme, não vai ficar funcional, vai ficar repetitivo, eu vou ter os mesmos problemas que estávamos tendo antes.

[07:12] Então fica de desafio para nossa próxima aula como podemos usar as listas para conseguir realmente automatizar essa funcionalidade, essa tarefa de atribuir ao onclick de cada elemento da minha lista uma função que toca o som do seu respectivo instrumento.

Faça como eu fiz: Declarando uma referência

PRÓXIMA ATIVIDADE

No JavaScript podemos criar nossas próprias referências armazenar de maneira mais simples valores para utilizarmos ao longo do código.

Desta forma, a partir deste código no CodePen, faça:

Uma referência que armazene uma lista com todas as teclas e imprima o valor dela na aba console.

Como fazer:

Abra o link do CodePen;

Abra a aba JS e escreva seu código dentro dela;

VER OPINIÃO DO INSTRUTOR

Opinião do instrutor

Gabarito:

const listaDeTeclas = document.querySelectorAll('input[type=button]');

console.log(listaDeTeclas);COPIAR CÓDIGO

Para selecionarmos todas as teclas do Alura Fone, é necessário usar o script const listaDeTeclas = document.querySelectorAll(‘input[type=button]), porquê o JavaScript vai criar uma referência chamada listaDeTeclas e armazenar dentro dela a NodeList com todas as teclas do Alura Fone, pois o script document.querySelectorAll(‘input[type=button]’) acessa o documento HTML (representado por document) e captura todos os elementos input que tenha como atributo type=button. É necessário passar o type, ou seja, o seletor de atributo, pois os inputs do HTML não tem classes nem IDs, e também temos um input que não é do tipo type, impossibilitando o uso do seletor de nome de tag input.

O que aprendemos?

PRÓXIMA ATIVIDADE

Nessa aula, você aprendeu:

Lista de elementos:

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.

Referências:

Para que serve uma referência e como declaramos uma referência de valor constante no código.

Conhecendo listas:

A estrutura de uma lista em JavaScript e como podemos acessar os elementos a partir dos índices com a sintaxe do colchetes.

PERCORRENDO UMA LISTA

[00:00] Olá. Na aula passada aprendemos como obter todos os botões do teclado do Alura Midi utilizando uma função chamada querySelectorAll, que recebe a classe que todas têm em comum. E o retorno disso armazenamos em uma constante, que é uma referência que vai facilitar o entendimento do nosso código, que criamos e demos o nome de listaDeTeclas. E depois aprendemos como acessar cada elemento individualmente dentro dessa lista utilizando os índices.

[00:38] Agora precisamos saber como acessamos todos de uma vez, na verdade cada item dessa lista de forma automatizada, pois não vai ser viável continuarmos repetindo esse trecho de atribuição a um clique de cada elemento, uma função de uma lista que não sabemos o tamanho. Aqui estamos no escopo pequeno, trabalhando em uma lista de 9 botões, mas imagina se fosse um teclado com 80 botões, com 1000 botões e que precisássemos adicionar alguma função ao onclick delas?

[01:11] Portanto, toda linguagem de programação contém uma estrutura de repetição para trabalhar nesses casos de códigos que precisam ser repetidos muitas vezes até um determinado ponto. E no JavaScript vamos usar, por enquanto, uma estrutura chamada while. Essa estrutura, essa condição While significa em inglês - até vou deixar aqui em cima do código comentado - significa enquanto.

[01:42] Enquanto determinada condição que vai ser informada aqui nos parênteses que escrevemos logo após a palavra while, a declaração while, vamos criar uma condição que vai determinar onde essa repetição deve parar, porque senão ela vai rodar para sempre, vai causar um loop infinito e vai travar o nosso navegador e possivelmente até o computador. Portanto, precisamos ter uma condição que tenha fim.

[02:15] E depois dos parênteses temos as chaves, onde vai ter a rotina dessa repetição, ou seja, o que vai ser executado, guardado e feito a cada loop e que chamamos de laço também, a repetição dessa função. Então dentro das nossas chaves vamos colocar a rotina de acessar um elemento da lista, acessar o onclick deles e colocar uma função dentro desse onclick. Então eu vou passar esse código que já tínhamos, o listaDeTeclas[0].onclick = tocaSomPom para dentro das chaves da while.

[02:57] E agora vamos precisar entender como vamos criar a condição do while. Ele espera receber uma condição que seja verdadeira ou falsa, ou seja, algo que o JavaScript vai conseguir verificar. No nosso console do navegador, vamos voltar lá, vamos abrir o DevTools e a aba console.

[03:22] Conseguimos testar um pouco esses valores que são verdadeiros e falsos, representados pela palavra true no inglês significa verdadeiro, este tipo de valor existe em várias linguagens de programação, e também tem o false de falso. Precisamos ter um true ou false e precisamos comparar as condições para passarmos no while.

[03:50] Então o que temos? Temos o tamanho da nossa lista, que é de nove elementos e temos repetições. Então podemos contar essas repetições e enquanto essas repetições não atingirem o tamanho da nossa lista, elas vão tendo que ser incrementadas, ter o seu valor contado.

[04:12] Então por exemplo, podemos utilizar aquele operador matemático que aprendemos, que é o de maior e de menor para comparar esses dois números, que é o número da repetição atual do while com o tamanho da lista. Então por exemplo, na primeira repetição, que seria a repetição 0, temos o número zero e podemos comparar com o tamanho da lista, que é o tamanho de nove elementos. Então enquanto zero for menor, o sinal de menor, que 9, true.

[04:45] Então é fácil, ele vai comparar: "0 é menor que 9?" True. Depois na próxima repetição: "um é menor que nove?" True. E vai repetindo. E sempre vamos ter do lado direito um número que vai condicionar a situação de falso. Então: "2 menor que 9?", true, até chegar em: "9 é menor que 9?", vai dar falso. Então precisamos ter do lado esquerdo do nosso operador algo que sempre conte as repetições e compare com o número total da nossa lista.

[05:20] Então vamos precisar criar um contador. E para isso precisamos criar uma referência variável, que vai se chamar contador. No JavaScript, para criarmos uma referência variável, escrevemos a palavra let, que define uma determinada variável, algo que vai receber sempre um valor novo. No nosso caso vai ser o contador, que vai começar com 0, let contador = 0;

[05:52] Então aqui dentro da condição do while eu vou ter o contador e vou compará-lo com 9, while (contador < 9). Só muito cuidado nesse ponto do código porque se você salvar e abrir em um navegador, vai dar loop infinito, porque o nosso contador ainda vale 0. Então lembre-se: 0 comparado (é menor que) 9, true, então aqui vai bugar o seu navegador e até seu computador. Então não salvem e não recarreguem.

[06:24] Agora precisamos saber como mudamos o valor do contador a cada repetição do nosso while. Então primeiro ele vai valer zero, vai comparar aqui com nove e vai dar true e vai entrar no contador. Então como eu vou mudar o valor desse contador? Como dentro do while eu posso fazer qualquer coisa, eu posso inclusive mudar o valor de contador, porque ele é uma variável, é uma referência variável, ele permite o chamarmos de novo embaixo, contador e vai receber, por exemplo, o número um.

[07:02] Só que se eu colocar um, quando ele passar de novo aqui, fazer a segunda repetição vai valer 1, vai ser menor que 9, verdade e depois vai valer 1 de novo, ou seja, vai ser de novo um loop infinito, sempre verdade. Então, na verdade temos que ter contador para receber algum valor algum mais 1, só que esse algum valor vai ter que ser o valor antigo do contador, então pode ser ele mesmo, contador = contador+1;.

[07:40] Como assim? Vamos pensar. Então aqui na primeira repetição o contador vai valer 0, 0 é menor que 9? Verdade. Então 0 + 1 vai ser 1, então ele primeiro faz essa conta, ele sempre vai processar essa conta primeiro, como se tivesse entre parênteses como eu coloquei, e depois ele vai fazer 0 + 1 = 1, então vou guardar 1. Então volta a retornar. Contador vai ser 1, 1 é menor que 9? É. Então agora o contador vai valer um e vai somar com mais 1, ou seja, vai ser 2 e vai guardar o novo valor 2. E assim por diante.

[08:30] Mas como eu sei que isso vai funcionar? Como eu vejo esse valor do contador sendo incrementado e modificado ao decorrer ou durante a execução desse while. Podemos usar agora uma funcionalidade que nos ajuda a entendermos o nosso código em alguns momentos, principalmente dentro de funções que não conseguimos acessar no console, mas de dentro do JavaScript conseguimos acessar o console atravessar da referência dele, que se chama console.

[09:04] Então dentro do JavaScript existe uma palavra reservada chamada console, que é o console do DevTools. E entrando no console temos aqui uma função chamada log, que consegue mandar mensagens lá para o nosso console do navegador, de quem for abrir essa página. Então eu posso colocar aqui o nome do nosso contador e ele vai imprimir o contador várias vezes, quer dizer, até essa condição ser falsa console.log(contador). Agora sim eu posso salvar, porque temos uma comparação que pode sim se tornar falsa.

[09:46] Então eu vou salvar e vou entrar no meu navegador e vou recarregar. Olha só 123456789. Já apareceu aqui o nosso contador, foi impresso pelo “main.js” linha 16, se entramos na nossa linha 16, vai ver que está o console.log. Então nós que criamos esse contador, esses números. E se tocarmos aqui nas teclas, a pom tocou, clap, tim, splash, toim, nenhuma delas ainda tocou, porque falta um pequeno detalhe no nosso código.

[10:26] Onde eu tenho o índice de cada elemento, também não posso ser sempre 0 nas nove vezes, eu preciso que seja uma vez o número 0, outra vez o número 1, depois o 2, depois o 3, depois o 4, depois o 5, ou seja, quem é que tem esse número 1234567... até a condição ser falsa? O próprio contador.

[10:49] Então podemos, no lugar de 0 escrever a referência contador, que vai guardar um número variável, um número que uma hora vai ter um valor e outra hora vai armazenar um outro valor, um outro número. Então vamos salvar, vou recarregar a minha página do navegador e vou agora apertar a tecla clap.

[11:17] O Clap deu pom, o tim deu pom, todos estão tocando pom. Claro, porque estamos com a função tocaSomPom para todos os nossos botões, as nossas teclas. Então precisamos, para as próximas aulas, descobrir como fazemos uma função mais genérica, que consiga descobrir qual é o som da tecla que estamos clicando.

[11:46] E só mais um detalhe aqui no nosso código antes de encerrarmos esse vídeo. É sobre a nossa condição que compara o contador com o número 9. A nossa lista, sim, sabemos que tem 9 elementos, mas será que sempre vamos saber quantos elementos tem em uma lista? Nem sempre. Como falamos, de repente estamos trabalhando com um teclado de 100 teclas, de 1000 teclas, enfim. Pode ser que seja um teclado dinâmico, pode ser outro tipo de funcionalidade, não necessariamente musical.

[12:20] Então podemos acessar o console do navegador e acessar a listaDeTeclas, e podemos ver que a listaDeTeclas, quando a abrimos, além de conferir os índices, podemos ver que existe uma propriedade/atributo chamado length. Length em inglês significa tamanho, o comprimento dessa lista que guarda o número 9.

[12:43] Então ao invés de usarmos o número 9 dentro do while, vamos usar a listaDeTeclas e o atributo length, que então esse atributo vai guardar o número 9. while (contador < listaDeTeclas.length).

[12:56] Então veja que estamos fazendo uma comparação entre duas referências que armazenam números nelas. Dessa forma conseguimos ter mais ideia e o nosso código consegue ter um pouco mais de legibilidade, apesar de ele ficar um pouco maior em termos de palavras, ao invés de comparar números, que nem sempre sabemos o significado dos números dentro do nosso código. Dessa forma fica mais dinâmico o nosso código.

[13:30] Enfim, vejo você no próximo vídeo para melhorarmos essa função que toca o som dos instrumentos musicais.

Foi necessário usar o while como estrutura de repetição para auxiliar o acesso a cada elemento dentro da lista de teclas, podendo aplicar a rotina de associar uma função no atributo onclick de cada um destes elementos.

Alternativa correta! O while é uma estrutura de repetição que foi necessária para trabalhar com a lista de teclas. Esta estrutura possibilita que uma variável de contador tenha seu valor incrementado a cada repetição, e que pode ser utilizado para acessar os indices de uma lista além de ser comparado com o tamanho de lista, formando a condição para a repetição cessar.

FUNÇÃO COM PARAMETROS

[00:00] No vídeo anterior aprendemos como acessar cada elemento da nossa lista de forma otimizada, utilizando a estrutura de repetição while, para aplicar uma rotina em cada elemento. No nosso caso a rotina é acessar cada elemento, e acessar o atributo onclick e atribuir uma função que vai tocar um som.

[00:26] Porém percebemos que dessa forma todas as nossas teclas do Midi ficaram com o mesmo som, pois estamos atribuindo a função tocaSomPom. Portanto, nosso desafio nesse vídeo é fazer uma função mais genérica, que possa tocar o som de qualquer tecla. A primeira coisa que vamos fazer é criar uma função que tenha um nome mais genérico. Então ao invés de ser função tocaSomPom, então no VS Code, no arquivo “main.js”, eu até vou tirar esse comentário que eu coloquei, podemos começar a eliminar coisas no arquivo que não vamos usar.

[01:14] E na declaração da função, aqui na linha 1 no meu caso, function tocaSomPom, eu vou fazer um nome mais genérico. E um nome mais genérico seria algo que não esteja atribuído ao nome do som, então vou colocar apenas function tocaSom e dentro do da nossa função, do bloco da função onde tem o código dela que ela está armazenando para ser executado quando chamarmos essa função, temos o document.querySelector, buscando o seletor da tecla pom, da tag áudio, que tem esse ID ’#som\_tecla\_pom’).play().

[01:57] Isso aqui também não pode estar fixo, já que o áudio vai ser informado, vamos precisar de certa forma informar qual vai ser a tag áudio específica e precisamos obter o ID dessa tag áudio em específico, que vai ter um som que vai ser reproduzido por essa função que toca um som de um dado ID, de uma determinada tag áudio.

[02:27] Portanto esse seletor também não pode ser fixo, ele precisa ser variável e precisa ser informado à nossa função. Quem chamar a função tocaSom vai precisar passar o ID para que a nossa função consiga tocar o som desse elemento áudio. Portanto vamos necessitar de um parâmetro.

[02:53] O parâmetro é alguma coisa que conseguimos passar para nossa função e que a execução, que o código da nossa função depende para funcionar corretamente. E muitas vezes esse parâmetro pode ser opcional, no nosso caso não é, realmente precisamos desse parâmetro, ou seja, desta informação, para nossa função tocar o som corretamente. Então para eu declarar um parâmetro de uma função basta que na assinatura dessa função, que é a declaração function, o nome desta função e o parâmetro dentro dos parênteses vamos criar o parâmetro.

[03:33] No nosso caso, o que vamos precisar? Do ID da tag áudio. Então eu posso escrever idElementoAudio, quanto mais descritivo, mais informativo for o nome das nossas referências, sejam elas constantes, variáveis, funções, parâmetros, melhor e mais fácil de entender o nosso código fica, chamamos isso de nomes significativos. Então é importante criarmos nomes que tenham sentido.

[04:06] Então deixa eu só escrever um português idElementoAudio, vamos precisar desse parâmetro, ou seja, que quem utilizar a função tocaSom informe a essa função o ID do elemento áudio. E portanto, como isso é um valor variável, eu posso já imediatamente utilizar dentro da minha função que vai precisar dentro do querySelector. Como é uma variável, ou seja, uma referência, eu não posso colocar isso entre aspas dentro do querySelector, eu apenas digito document.querySelector(idElementoAudio).play();

[04:48] Então vou usar o auto complete para me ajudar a escrever corretamente e já tenho nossa função um pouco mais genérica funcionando de certa forma no nosso código. Para testarmos, primeiro eu tenho que arrumar dentro do while o onclick = tocaSomPom, agora vai ser onclick = tocaSom então vou corrigir o nome de acordo com a nossa mudança, vou salvar o arquivo e agora vou voltar no navegador e vou testar a nossa função tocaSom utilizando os parâmetros.

[05:29] No nosso onclick dentro do while ainda não estamos informando os parâmetros, logo vamos fazer isso, primeiro precisamos testar se o código que fizemos está correto. Então eu vou recarregar a minha página no Chrome, vou limpar o console, porque estamos imprimindo o contador, então eu só limpo no botão de limpar o console e vou chamar a função tocaSom.

[05:54] Aqui temos a referência, abre e fecha parênteses e ao colocar o cursor dentro do parênteses ele já me informa qual é o parâmetro que eu preciso passar, que é o ID do elemento áudio. Se eu não passar, ele já deve estar me antecipando no console o erro que eu vou obter se eu não passar o parâmetro, vou dar um “Enter”, ele vai falar que não consegue ler a propriedade play de algo que está nulo.

[06:19] Ou seja, no meu “main.js” essa parte do código document.querySelector e idElementoAudio que não foi passado vai retornar um código nulo, portanto ele não consegue executar o play de algo que está nulo. Então vamos informar um ID de um elemento áudio que existe.

[06:43] Então vou na aba “Elements” e vou olhar minhas tags áudios de vou ver aqui os IDs delas. Então só uma tecla pom já sabemos, vamos pegar aqui um ID diferenciado, vamos pegar o som tecla Tic. Então eu vou copiar, vou escrever tocaSom(‘’), abre parênteses, vou colar.

[07:04] Lembrando que tem que estar entre aspas o valor, porque o valor que vamos informar é um texto, então as aspas vão ter que ser passadas junto no parâmetro. E lembrando que esse seletor é um ID, um atributo ID, então precisa começar com hashtag. Fazendo isso o parâmetro fica entre aspas tocaSom(‘#som\_tecla\_tic’), fecha aspas. Estou usando aspas simples e fecha o parênteses. Vou dar o “Enter” e assim que dermos o “Enter” já vai sair o som do Tic.

[07:39] Ficou muito baixinho o som do Tic, eu vou escolher outro. Saiu, mas bem baixinho. Vou colocar o Tom. Vemos que é uma caixa. Vamos tentar o psh. Saiu. Então a nossa função já está correta, ela já está recebendo o som adequado, de acordo com o ID, já está reproduzindo o som adequado de acordo com a ID da tag áudio.

[08:11] Agora que já temos uma função mais genérica de tocar som, precisamos saber como dentro do nosso “main.js”, dentro do while, utilizamos essa função de forma mais genérica. Pois se tentarmos executá-la agora, vou recarregar, vamos ter alguns problemas no nosso código e vamos entender o que que está acontecendo.

O que é parâmetro de uma função?

PRÓXIMA ATIVIDADE

A função é o conjunto de instruções de código, que executa uma determinada tarefa, que é utilizada somente quando chamada/invocado, podendo receber parâmetros ou não.

Deste modo, o que são os parâmetros em uma função?

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.

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.

FUNÇÕES ANONIMAS

00:00] No vídeo anterior vimos como fazer uma função mais genérica que possa executar qualquer tipo de som armazenado numa uma tag áudio, dado o seletor da tag áudio. Fizemos a função tocaSom. E conseguimos testá-la no console passando o ID daquelas tags áudios e escutar o som de cada instrumento daquela tag áudio. Porém, quando fomos clicar dentro do Alura Midi nas nossas teclas no navegador, obtivemos um erro no console dizendo que o querySelector falhou, pois [Object PointerEvent] não é um seletor válido.

[00:46] Isso é porque não informamos o seletor correto para nossa função, o parâmetro correto para nossa função, então o JavaScript tentou resolver com as informações que ele tinha ali. Se olharmos aqui dentro do while, no nosso “main.js”, onde temos a instrução do onclick atribuindo a função tocaSom, vemos que não passamos o parâmetro dela.

[01:10] Então como vamos conseguir passar o parâmetro? Então se eu escrever aqui os parênteses e passar um ID fixo, por exemplo, só para testarmos ‘#som\_tecla\_pom’ e salvar o arquivo e recarregar, vamos ter aquele velho problema que a página não pode executar o play, porque o usuário ainda não interagiu com a página.

[01:45] Ou seja, toda vez que colocamos os parênteses ao lado do nome de uma função, já a invocamos imediatamente, antes mesmo do usuário interagir com a página e isso é uma restrição, um bloqueio que o browser (navegador) faz e não é isso que queremos, não queremos que ela seja executada nesse momento que o while ainda está atribuindo e guardando essa função em cada click de cada tecla do Midi.

[02:14] Então precisamos de certa forma poder passar um parâmetro para essa função, para esse atributo onclick, sem executá-la imediatamente. E o JavaScript tem uma funcionalidade muito interessante, chamada funções anônimas, que eu vou utilizá-la no lugar da função tocaSom.

[02:41] Então no onclick eu vou remover o tocaSom por enquanto e vou escrever uma função anônima, ou seja, uma função sem nome. E essas funções sem nome só podem ser utilizadas nesse contexto de quando elas são o valor de algum atributo ou estão sendo armazenadas dentro de alguma referência constante ou variável. Então eu estou aguardando uma função, vou escrever function dentro desse atributo onclick.

[03:09] E não preciso colocar um nome nela, abre e fecha parênteses e depois abre fecha as chaves e dentro das chaves eu vou conseguir chamar a minha função tocaSom() com parênteses, porque na nossa atribuição, a função anônima no onclick, vamos declarar ou criar uma função nova, portanto, quando criamos uma função não a invocamos imediatamente, só criamos e declaramos que ela existe naquele contexto, naquele momento.

[03:47] Para chamá-la eu vou ter que fazer o clique no botão, naquele determinado botão. Portanto agora dentro da minha função anônima, eu vou poder chamar a função tocaSom e aí colocar os parênteses e passar um dado ID. Então vou colocar os parênteses e aspas simples tocaSom(‘#som\_tecla\_pom’), por exemplo.

[04:16] Então vou salvar o meu arquivo “main.js”, vou recarregar o Alura Midi e vou clicar na tecla pom. Vamos ver na Splash, pom também e na tom? Também pom. Todas estão com o mesmo som ainda. Apesar da nossa função tocaSom já estar preparada para receber um ID dinamicamente, ainda estamos chamando-a fixo, o ID da nossa tecla ainda está fixo dentro do nosso código. Então agora o nosso desafio é como eu crio um ID dinamicamente ou como eu tenho acesso ao ID das teclas áudio nesse momento do meu código. Vamos ver no próximo vídeo.

TEXTOS DINAMICOS

[00:00] Até o momento descobrimos como fazer uma função receber um parâmetro dinamicamente para que executasse a sua função, a sua funcionalidade de forma dinâmica.

[00:12] No nosso caso a função tocaSom está recebendo o ID de um elemento que vai ser buscado pelo querySelector e então reproduzido o seu som através da função play. Essa função tocaSom está sendo utilizada dentro do nosso while, que acessa cada tecla do nosso Alura Midi, utilizando a variável que criamos para contar as repetições e então acessar o índice, elas vão conter um índice que vai acessar um determinado elemento da lista de teclas.

[00:48] E portanto, a cada elemento vamos acessar seu onclick e atribuir uma função anônima e aí sim chamará função tocaSom. Porém, os parâmetros que estamos passando para a função tocaSom ainda está fixo, porque escrevemos o texto fixamente, porém sabemos que essa última parte do nosso parâmetro, a parte do nome do instrumento, do nome do som, deve ser dinâmica.

[01:18] Se entrarmos no nosso “index.html” e olharmos as tags áudio, vamos ver que existe um padrão para os IDs, todos começam com som\_tecla\_pom, fazendo só a última parte ser diferente a cada elemento áudio, cada tag áudio.

[01:36] E nos nossos botões podemos perceber que temos uma parte dessa informação, temos além da classe tecla, também cada botão tem uma classe chamada tecla e o nome do som do instrumento, pom, clap, tim. Então dentro do while não temos acesso a tag áudio, porém temos acesso a um elemento button e dentro desse elemento button, através da sua classe, podemos acessar qual é o instrumento em questão que está sendo clicado.

[02:10] Então é a partir dessas classes que estão dentro dos elementos button que vamos conseguir montar dinamicamente esse texto, pois texto é um tipo de dado que é dinâmico, podemos escrever e inventar do jeito que precisarmos. Então vamos usar o console do navegador, estou no Chrome, vou limpar o console e vou acessar a lista de teclas e um índice qualquer, vou acessar o índice zero, por exemplo, que é o da tecla pom.

[02:42] Eu vejo que ele retorna o botão se eu der “Enter”, então eu vou acessar um atributo, que quando colocamos o ponto, lista de teclas colchete zero ponto, depois dos colchetes, conseguimos entrar dentro desse elemento, ver todos os atributos e funcionalidades que estão disponíveis para esse elemento como já temos feito.

[03:02] E um desses atributos, assim como o onclick é um atributo, o class também é um atributo e ele é disponível da forma mais organizada pelo listaDeTeclas[0].classList. E pelo próprio nome já sabemos que isso vai nos retornar uma lista que é um tipo de dado que também já estamos aprendendo a utilizar. Se eu dar um “Enter” ele vai me retornar naquela anotação também que já estamos começando a conhecer melhor, que é a de colchetes representando uma lista.

[03:36] E também nas listas temos o índice. No nosso caso, em uma lista de classes, temos o índice, o zero, o valor armazenado é a classe, a classe simplesmente é um texto, portanto está aqui entre aspas, representando o que é um texto, no índice 1 também temos a segunda classe, nos totalizando uma lista de 2 itens. Lembrando que os índices sempre começam com zero.

[04:05] Portanto, se eu quiser acessar a segunda classe dessa lista, eu tenho que usar a anotação de colchetes com índice 1. Dessa forma listaDeTeclas[0].classList[1] e assim vou conseguir acessar a segunda classe da minha lista de classes.

[04:33] Com esse dado, agora já sabemos como obtê-lo, vamos trazê-lo dentro do nosso while. Então vou voltar no “main.js” e vou ter a nossa lista de teclas. Agora que usamos o contador que está armazenando o valor do nosso índice, então listaDeTeclas[contador].classList[1], que sabemos que vai retornar a lista que tem aquela instrução, aquele pedaço de texto que se chama tecla\_pom, enfim, a tecla que estiver aqui no momento.

[05:07] E podemos inclusive utilizar o console.log para ver se realmente essa informação que colocamos está correta. E para facilitar o entendimento do nosso código eu vou armazena-la dentro de uma referência constante para ficar mais fácil de entender o que esse trecho de código faz.

[05:28] Então eu vou declarar uma constante que vai ter, por exemplo, o instrumento. Instrumento é o nome da minha constante e vai armazenar essa instrução que vai obter o nome da segunda classe deste elemento. Então eu vou dar um console.log(instrumento) nessa referência instrumento, que é uma constante.

[05:54] Então vou recarregar no navegador a minha página e já vou ver que no console está dando o console tanto do nosso instrumento como também do valor do contador que mantivemos. Então na repetição 1 tem a tecla\_pom, na repetição 2 eu tenho a tecla\_clap, na repetição 3 Tim, na repetição 4 Puff e assim por diante, vai até a repetição 9, que encerra o nosso while.

[06:24] Voltando no “main.js”, podemos perceber que tem uma repetição de código no nosso while, que é a lista de teclas contador. Estamos usando tanto para acessar o classList, tanto para fazer o onclick. Essa repetição de código não é legal e podemos também atribuir isso para uma constante que tenha o nome mais significativo, afinal eu vou ter listaDeTeclas[contador]?

[06:55] O que isso tem por baixo dos panos? Então quando eu acesso um elemento de uma lista de teclas eu tenho uma tecla, então eu posso criar uma constante, uma referência tecla, que vai conter o elemento daquele momento da lista de teclas. Então passa o nosso índice que está dentro do contador e eu já posso substituir esse código listaDeTelas[contador] do classList por tecla e do onclick também por tecla.

[07:32] Já deixando nosso código um pouco mais limpo. E agora que eu já tenho a tecla, tenho o instrumento, falta só termos o ID correto, essa parte de #som pois dentro da referência instrumento temos tecla\_pom.

[07:56] Como que seria isso? Se eu tivesse uma string seria algo como um pseudocódigo dentro de um comentário. Seria algo como ‘#som e o restante do código eu já tenho, que está dentro desse instrumento. Então vou representar como chaves essa parte que é variável, que é dinâmica, dentro da nossa string.

[08:29] É possível fazer isso com JavaScript e é muito simples. Eu vou criar uma nova referência constante que vai guardar o ID da tag áudio. Então vou chamá-la de idAudio. E ela vai receber essa nossa string que vai ter um pedaço dinâmico dentro dela. Só que para eu fazer isso eu não posso utilizar a string comum, que é com aspas simples ou aspas duplas, eu vou ter que substituir pela crase. E a crase é uma outra forma de criar strings (textos) dentro dos JavaScript.

[09:09] Então vou começar a minha string com crase e vou finalizar com crase. Veja que funciona, fica com o mesmo destaque de cor, amarelinho, sinalizando que é uma string.

[09:23] E agora tudo que é variável dentro da minha string eu preciso envolver entre chaves, porém iniciando $ antes da abertura das chaves. Veja que até a cor do highlight do VS Code já muda, sinalizando que é um pedaço de código.

[09:43] Esse ${} faz uma abertura para o código de JavaScript de dentro da string, eu posso acessar variáveis, métodos fazer contas aqui dentro, é incrível. E o nome desse recurso que utilizamos da linguagem JavaScript é template string. Eu até vou escrever para vocês conhecerem.

[10:05] E agora eu vou fazer o console.log em vez do instrumento, eu vou fazer o console.log da nossa referência idAudio para vermos como ficou. Então vou salvar no “main.js”, vou recarregar o navegador e já temos os IDs dinamicamente, som\_tecla\_pom, som\_tecla\_clap.

[10:31] Vamos olhar no Firefox, isso é muito importante, sempre verificar nos navegadores se está com algum problema. E vemos que está tudo certinho no Firefox também.

[10:41] Então agora dentro da nossa função, onde tem tocaSom, eu vou tirar esse parâmetro fixo que está dentro de função, da nossa função anônima, da tocaSom, e vou começar a passar a nossa idAudio, que é o parâmetro, a nossa string que está sendo montada a cada iteração do while.

[11:10] Vou salvar, vou recarregar, estou no Firefox e vou executar os sons de cada um dos nossos instrumentos. Vou clicar no Pom, saiu. E no clap? Saiu. No tim? Saiu. No splash, no tic, no pom, no toim, no puff, no psh, todos deram certo! Então finalmente no Alura Midi está funcional, com código super inteligente, super reaproveitado, podemos tirar esses console.log para não ficar poluindo nosso console da aplicação.

[11:53] Na verdade eu posso tanto apagar, mas se vocês quiserem manter para se lembrar, podemos colocar com barra barra, comentar esse código, salvar.

[12:03] Vou recarregar, já não vai ter mais esses consoles. Ótimo. E basicamente as principais funcionalidades do Alura Midi está implementada e a partir dos próximos vídeos vamos trabalhar ainda mais nas melhorias desse código, em como deixá-lo ainda mais profissional, conhecer mais recursos da linguagem e principalmente, em saber como começamos também a trabalhar na navegação via teclado.

REPETIÇÃO OTIMIZADA COM FOR

[00:00] No nosso último vídeo conseguimos finalmente terminar de implementar as funcionalidades completas do Alura Midi. Então se eu clicar em cada tecla do teclado do nosso Midi, o som daquele instrumento é reproduzido.

[00:17] E no código conseguimos fazer de certa forma um código inteligente, automatizado, que tem poucas repetições ou basicamente nenhuma repetição, e conseguiu espelhar essa funcionalidade para todas as teclas. Agora pensando em melhoria de código, ainda há bastante coisas que podemos fazer. E aqui nesse vídeo vamos melhorar um pouco mais em relação a essa forma de percorrer cada elemento por uma lista, que estão dentro de uma lista.

[00:52] Utilizamos a estrutura de repetição while, que vimos que depende de uma condição ser verdadeira ou falsa para determinar se aquela repetição, essa rotina que está dentro das chaves do while vai ser aplicada. Porém, imagina que podemos aumentar o Alura Midi e em vez de ser apenas um teclado digital, podemos querer oferecer para os usuários outros instrumentos, um bongo, um piano, uma harpa, enfim, podemos aumentar a quantidade de instrumentos que queremos oferecer para os nossos usuários do Alura Midi.

[01:40] Portanto, o while dependeu de um fator externo para que ele pudesse funcionar, que é no caso o contador. Tivemos que criar uma variável, uma referência variável chamada contador, que é verificada a cada repetição do while e tem seu valor modificado também a cada repetição da nossa rotina do while.

[02:07] Porém, imagina se tivéssemos outros instrumentos que tem botões, então se fôssemos reproduzir outros whiles, se tivéssemos outras listas para reproduzir, teríamos que declarar outros contadores e então começaria a voltar a ter repetições em nosso código, teríamos que criar, por exemplo, um contador let contador2. Ou contadorBongo, contadorPiano, enfim, essa variável contador iria ficar repetitiva.

[02:51] E o JavaScript oferece outras estruturas na linguagem que podem resolver esse problema de forma mais otimizada, ou seja, já oferece contido nele toda a estrutura, tanto do contador como da parte da condição para nossa rotina de repetição em encerrar e também da parte da incrementação desse valor contador.

[03:16] E essa estrutura que vamos utilizar no lugar do while se chama for, e for significa “para”. O for uma estrutura de laço de repetição ou de loop, que também vai fazer a mesma coisa, enquanto uma instrução, uma condição for verdade, ele vai repetir essa rotina, porém a vantagem em relação ao while é que o contador pode ser declarado ainda dentro dos parênteses do for.

[03:51] Então agora eu substitui a palavra while por for e dentro do parênteses eu vou fazer a declaração do meu contador, let contador = 0; e depois do ponto vírgula eu vou ter minha condição. Então eu não vou mais precisar ter a variável let declarada fora do nosso laço de repetição, da nossa estrutura, declaração de repetição.

[04:26] E também a terceira vantagem do nosso for, a primeira é ter a declaração do contador dentro, junto com o segundo parâmetro do for ter a condição para o nosso laço encerrar, e também a terceira é que essa incrementação que estamos fazendo aqui no contador, que é contador recebe contador mais 1, pode ser declarada nesse terceiro parâmetro do for, com uma sintaxe muito enxuta e muito bonita que é contador++.

[05:06] Esse contador++ vai fazer exatamente o que essa linha de código na linha 21 está fazendo, que é pegar o contador, o valor atual da variável contador, incrementar mais 1 e guardar novamente essa execução desse código na variável contador, reatribui esse novo valor. Então não vamos mais precisar colocar esse código da linha 21 dentro do for. Vamos usar console.log só para verificar novamente, eu vou tirar o comentário, vou tirar o console.log do nosso idAudio para o nosso código ficar mais bonito.

[05:53] Vou salvar. Também vou mover o template string para depois da linha da constante idAudio para ficar tudo bonitinho, as declarações das nossas constantes, o onclick e o console.log no final. Vou salvar o meu “main.js”, vou no navegador, vou recarregar o Chrome.

[06:13] 12345678 como já vimos, 0 a 8. Lá no Firefox agora eu vou recarregar. Também confirmou o nosso contador ainda funcionando do jeito que esperamos. E vamos ver também agora se os nossos botões estão ainda reproduzindo o som. Então vou tocar a tecla Puff.

[06:37] Deu certo. Vou tocar a tecla Tim. Deu certo aqui no Firefox, vamos ver no Chrome, vou recarregar novamente, colocar a tecla psh e a tecla tom, deu certo. Então vou só remover esse console.log para o nosso código ficar mais limpo.

[06:57] É sempre bom deixar também no fim do código uma nova linha e não encerrar o final do arquivo com um final de código é sempre bom deixar esse respiro, porque depois se o nosso código vier a sofrer algum tipo de transformação por ferramentas que juntam o código, que ofuscam o código, isso vocês podem conhecer em cursos mais avançados. Então é uma boa prática sempre deixar uma linha vazia no fim do arquivo.

[07:27] Então agora nosso código está um pouco mais bonito, um pouco mais funcional e pronto para receber novos instrumentos caso quisermos.

[07:37] Bom, até agora vimos uma melhoria em nosso código usando o for. Agora vamos seguir no nosso desenvolvimento do nosso Alura Midi fazendo algumas melhorias de acessibilidade e usabilidade do Alura Midi descobrindo como vamos ffazer para adicionar funcionalidades e interações usando as teclas do teclado.

Faça como eu fiz: Utilizando o For

PRÓXIMA ATIVIDADE

Como vimos nesta aula, usamos a estrutura de repetição for, para percorrer uma lista de forma mais otimizada, onde em sua declaração criamos a variável contadora da repetição, a condição de interrupção das repetições e a condição de incrementação da variável contadora.

A partir deste código CodePen, faça:

Um script que percorra a lista de teclas do AluraFone, e ao clicar em qualquer tecla do teclado, imprima o valor no campo Digite seu telefone.

Como fazer:

Abra o link do CodePen;

Abra a aba JS e escreva seu código dentro dela;

VER OPINIÃO DO INSTRUTOR

Opinião do instrutor

Gabarito:

const listaDeTeclas = document.querySelectorAll('input[type=button]');

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

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

const tecla = listaDeTeclas[indice];

tecla.onclick = function () {

inputTel.value = inputTel.value + tecla.value;

}

}COPIAR CÓDIGO

Primeiro, você precisa criar uma referência para receber a lista com todas as teclas do AluraFone e uma outra para capturar o input Digite seu telefone, que no caso do código foi const listaDeTeclas e const inputTel, respectivamente.

O segundo passo é criar o laço de repetição for, que vai percorrer a lista de teclas, por esta razão, a condição do for é indice < listaDeTeclas.length, ou seja, enquanto o indice for menor que o tamanho de listaDeTeclas, executa um comando.

Dentro do for, é necessário que se crie uma referência constante const tecla para que a cada iteração, receba o valor correspondente da tecla listaDeTeclas[indice].

Após isto, adicione no onclick uma função anônima que receberá o valor do campo Digite seu telefone inputTel.value, e atribua a ela a soma entre o valor do campo Digite seu telefone e o valor da tecla clicada tecla.value, ficando inputTel.value = inputTel.value + tecla.value.

O que aprendemos?

PRÓXIMA ATIVIDADE

Nessa aula, você aprendeu:

Percorrendo uma lista:

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.

Função com parâmetros:

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.

Textos dinâmicos:

Como criar textos dinâmicos utilizando template string e como acessar as classes de um elemento através do atributo classList.

Repetição otimizada com For:

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 ++.

EVENTOS DO TECLADO

[00:00] Bom, agora o nosso Midi está quase pronto, quer dizer ele já está pronto, ele já está funcionando, se clicarmos com o mouse nas teclas já vamos escutar o som de todos os instrumentos.

[00:15] Vou clicar no psh. Então o que mais falta para nossa aplicação Midi, o Alura Midi ficar sensacional? Bom, como usamos a tag < button> para cada botão do Alura Midi, o < button> já tem uma navegação via teclado por padrão, é um funcionamento padrão da tag < button>, então se eu pressionar a tecla “Tab” do meu teclado do computador, veja que uma luzinha amarela, que já foi adicionada pelo CSS, vai aparecer mostrando qual é o foco da navegação de tab da nossa aplicação.

[01:03] Então se eu der “Tab” ele vai selecionando qual botão está em foco. E se eu pressionar a tecla “Enter” ou a barra de espaço essas duas teclas vão reproduzir o som ou reproduzir aquilo que foi programado para o nosso evento de clique do mouse. Então vou apertar agora com a tecla “Enter” e com a barra de espaço.

[01:37] Vocês já devem ter percebido uma coisa que aconteceu de estranho. Quando eu apertei o “Enter”, vou apertar de novo agora em uma tecla pouco menos barulhenta, na psh, “Enter”. Reproduziu o som e o fundo do botão não ficou vermelho, não deu aquela característica, aquela impressão que o usuário pressionou o botão visualmente.

[01:59] Agora vou apertar com a barra de espaço. Veja que com a barra de espaço temos essa impressão de estar sendo pressionado o botão. Eu estou no Chrome, vamos ver no Firefox. Eu vou recarregar, com o “Tab” eu já tenho navegação via teclado funcionando, agora eu vou apertar no “puff” a tecla “Enter”, reproduziu o som e não ficou vermelho o fundo. E a barra de espaço também reproduziu o som e não ficou vermelho o fundo.

[02:30] Se eu clicar com o mouse, fica vermelho o fundo. No Chrome, se eu clicar com o mouse fica vermelho o fundo. Então perceba que a navegação via teclado do Midi está problemática, está inconsistente e precisamos arrumar isso nessa aula.

[02:48] Para começar precisamos dar uma olhada no nosso CSS. Então vou abrir o Visual Studio Code e vou abrir o arquivo “estilo.css”, que tem todos os estilos da nossa aplicação.

[03:02] E na linha 49 eu vou ver que já foi programado os estilos para esse estado active do button, que é quando o usuário está pressionando aquele botão, ele é representado por esse estado, essa pseudoclasse active. Porém também foi destinado uma classe ativa para representar esse mesmo conjunto de estilos e vamos ter que fazer a inserção dessa classe ativa pelo JavaScript nos botões que forem "clicados" nesse teclado do nosso Alura Midi com o “Enter” e com o espaço.

[03:50] Podemos, inclusive, voltando para o navegador, estou com o Chrome aberto, abrir um botão, por exemplo, vou pegar a tecla “Tim” e testar na aba “Elements” adicionando uma classe manualmente dentro do atributo class, vou colocar ativa. Se eu escrever ativa já vai aparecer o botão vermelho e esses são os estilos de quando o botão está sendo pressionado.

[04:17] Agora eu preciso saber como fazer isso com o JavaScript. Vou recarregar só para remover essa classe ativa que eu coloquei e vamos para o console para testar e tentar descobrir como eu faço isso via código.

[04:33] Já sabemos como acessar um elemento que está dentro do nosso teclado do Midi, criamos uma referência listaDeTeclas com todas as teclas. Para acessar um elemento eu uso a anotação de colchetes e o número do índice. Eu vou usar o 2 que vai selecionar o “Tim”, o botão “Tim”. Então se eu dou “Enter”, se eu digito isso já vejo que é realmente referência àquele botão.

[04:58] E também com o ponto conseguimos acessar todos os atributos e funcionalidades para um elemento HTML, e também já conhecemos o atributo classList, que armazena uma lista de todas as classes que estão naquele elemento e vimos que existe a classe tecla e a classe tecla\_tim.

[05:18] Porém, é uma lista, precisamos algo nessa lista, adicionar uma nova classe, portanto se eu colocar o ponto depois de classList eu consigo ver também uma série de funcionalidades disponíveis para um classList, então podemos ver que tem muitas coisas aqui, inclusive algumas coisas que já vimos em outros tipos de lista, como por exemplo length, que é o tamanho.

[05:45] E aqui queremos adicionar, portanto existe essa função add, que no português significa “adicionar”, e eu já sei que é uma função porque justamente o f embaixo indicando que aquilo é uma função. Então eu vou usar essa função add, toda função precisa ter parêntesis para ser chamada. E eu vou passar uma string, ou seja, um texto, então eu tenho que abrir aspas, pode ser simples ou duplas, e preciso passar o nome da classe que eu quero adicionar, no nosso caso é ativa. Então ao executar esse código listaDeTeclas[2].classList.add(‘ativa’) e quando der “Enter” a tecla “tim” vai ter que ficar vermelha.

[06:26] Viram? Aconteceu. Agora temos que reproduzir esse código dentro do nosso “main.js” e esse código de adicionar a tecla ativa tem que ser no momento em que eu dou “Enter” com o teclado, que eu pressiono uma tecla do teclado. E diferente do que temos trabalhando aqui, que foi cliques do mouse.

[06:50] Agora eu vou voltar no “main.js” e dentro do nosso for, onde já temos uns eventos, o evento de clique destinado, embaixo eu vou começar adicionar mais um código.

[07:05] Eu vou dar “Enter”, vou chamar a tecla em questão do nosso loop que está sendo acessado pelo for, a nossa lista de elementos e vou agora entrar nessa tecla e ver quais são as outras opções que existem para ela de eventos. Então, por exemplo, existe o evento on, e acontece todos, o on conseguimos acessar vários tipos de evento no Visual Studio Code, ele mostra também, mas nesse caso ele não está me mostrando, mas às vezes ele mostra.

[07:37] O evento que eu quero é um evento de teclado, e o nome desse evento de quando eu aperto e a minha tecla está embaixo. Então o nome dele é onkeydown, key significa “tecla”, então quando a tecla estiver abaixada, tecla.onkeydown.

[07:57] Então agora já sabemos qual é o evento que representa esse momento que uma tecla é apertada no JavaScript e vamos atribuir uma função anônima igual fizemos com o tocaSom no onclick, então depois de onkeydown = function () { e vamos agora dentro colocar aquele código que já testamos no console, que é acessar a tecla em questão, acessar o classList, acessar a função adicionar e colocar a classe ativa.

[08:32] Então eu tenho a tecla em questão, então tecla.classList.add(‘ativa’); vou salvar e vou testar no meu navegador. Vou dar um recarregar e vou começar a navegar via teclado no meu Alura Midi no Chrome. Então vou dar o “Enter”.

[09:03] Ficou vermelho. Vou dar barra de espaço, ficou vermelho. Vou dar “Tab”, também. Puff também. Eu nem estou apertando e nem estão reproduzindo som e está ficando vermelho todas as teclas agora. O que está acontecendo?

[09:23] Então o que está acontecendo é um problema que quando começamos a trabalhar com inserção de classes no JavaScript precisamos prever. Se abrimos a aba “Elements” do nosso console vamos ver que todos os nossos botões estão com a classe ativa que foi inserida via JavaScript.

[09:41] Eu inclusive vou dar mais um “Tab” para vocês verem que o JavaScript vai fazer piscar os elementos e adicionar mais uma classe. Vou dar um “Tab”, vocês vão ver.

[09:57] Piscou de roxo e adicionou a classe ativa no “Tic”. Mais um, clicou, piscou de roxo e isso identifica que o JavaScript está manipulando essa árvore de elementos e os atributos desses elementos, então eles são destacados.

[10:17] Tudo que colocamos no JavaScript não vai "automagicamente" sumir. Então o JavaScript não sabe, ele é uma linguagem de programação que ele só faz aquilo que você diz, e só dissemos para ele adicionar classe, não dissemos qual é o momento de removermos essa classe. Portanto agora vamos ter que descobrir como fazemos para remover uma classe de um elemento com JavaScript.

ADICIONANDO E REMOVENDO CLASSE

[00:00] De novo vamos voltar no console e vamos fazer o mesmo exercício, vamos acessar a listaDeTeclas, vou pegar um índice qualquer, vou pegar o índice 4, que vai pegar bem a tecla “splash” no meio, vou acessar o classList dela e vou usar o ponto depois do classList com todas as opções que eu tenho e vou observar que eu tenho uma função chamada remove.

[00:24] Então como estou acessando classList, esse remove só poderia significar remover uma classe dessa lista de classes, então vou abrir e fechar parênteses, vou passar entre aspas o nome da classe que eu quero tirar, lembrando que como já estávamos trabalhando com classes, não há necessidade de voltar o ponto no início, pois já são classes, não estamos trabalhando com seletores e sim com classes.

[00:52] Então eu vou escrever o nome da classe que eu quero remover ativa listaDeTeclas[4].classList.remove(‘ativa’) e quando eu der “Enter” o “Splash” tem que voltar a ser branco.

[01:03] Pronto, funcionou. Agora a mesma lógica que fizemos para adicionar, vamos ter que fazer para remover. Então qual é o momento que fazemos aqui que identificamos na ação do teclado que eu preciso remover essa classe? E podemos usar um outro evento do teclado, que é quando o usuário solta a tecla.

[01:27] Então eu vou lá no meu “main.js” e vou criar um novo evento, com uma nova função. Então tecla.onkeyup, que é quando o dedo do usuário está no alto, solta aquela tecla do teclado. Vou passar uma função function, abre e fecha parênteses, uma função anônima, abre e fecha parênteses, abre e fecha chaves e vamos usar o tecla.classList.remove(‘ativa’);. Vou salvar, vou recarregar o meu navegador e vamos ver se vai funcionar.

[02:17] Então vou apertar com a barra de espaço o “pom” vocês vão ver que reproduziu, se eu manter pressionada ele vai manter vermelho e se eu soltar ele vai reproduzir o pom. Mas agora temos um problema. Se eu continuar usando o “Tab”, ele vai continuar pintando todos de vermelho.

[02:41] Então temos mais um problema para resolver. O desafio que temos aqui agora na nossa aula é que a nossa lógica de a cada tecla que é pressionada e removida está sendo válida para qualquer tecla do nosso teclado. Então se eu apertar a tecla “H”, se eu apertar a tecla “G”, se eu apertar a tela “J”, não reproduz o som, assim como a tecla “tab” não reproduz o som porque não é o comportamento padrão dos buttons fazer a mesma coisa que o clique, como o “Enter” e o espaço faz.

[03:13] Porém, como eu abri as opções, qualquer tecla que for pressionada coloca a tecla ativa, estamos com a nossa lógico muito ampla, precisamos limitar, precisamos dizer quando aperta uma tecla, mas essa tecla tem que ser “Enter” ou barra de espaço, aí sim você coloca a classe ativa. Então precisamos saber como vamos fazer isso e espero você no próximo vídeo.

Faça como eu fiz: Modificando estilos com JavaScript

PRÓXIMA ATIVIDADE

Como vimos nesta aula, é possível modificar a classe de um elemento com o JavaScript de uma forma simples.

A partir deste código CodePen, faça:

Um JavaScript que percorra a lista de teclas do AluraFone, e ao clicar na tecla Enter ou Espaço, adicione a classe ativa no elemento e também resolva o momento de retirar a classe ativa do elemento.

Como fazer:

Abra o link do CodePen;

Abra a aba JS e escreva seu código dentro dela;

VER OPINIÃO DO INSTRUTOR

Opinião do instrutor

Gabarito:

const listaDeTeclas = document.querySelectorAll('input[type=button]');

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

const tecla = listaDeTeclas[indice];

tecla.onkeydown = function (evento) {

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

tecla.classList.add('ativa');

}

}

tecla.onkeyup = function () {

tecla.classList.remove('ativa');

}

}COPIAR CÓDIGO

O primeiro passo é criar uma referência (const listaDeTeclas) para poder receber a lista com todas as teclas do Alura Fone, ficando const listaDeTeclas = document.querySelectorAll('input[type=button]').

O segundo passo é criar o laço de repetição for, que vai percorrer a lista de teclas, por esta razão, a condição do for é indice < listaDeTeclas.length, ou seja, enquanto o indice for menor que o tamanho de listaDeTeclas, executa um comando.

Dentro do for, para facilitar o entendimento, crie uma referência constante const tecla para que a cada iteração , receba o valor correspondente da tecla listaDeTeclas[indice], ficando const tecla = listaDeTeclas.

Após isto, adicione no evento de ao apertar uma tecla do teclado onkeydown uma função anônima que recebe como parâmetro um evento. Dentro do escopo da função, vamos criar um if com a condição: se evento.code (o código do evento onkeydown) for igual a "Enter" ou for igual a "Space", adicione a classe ativa no elemento tecla, ficando então assim tecla.classList.add(‘ativa’);

Como foi resolvido a adição da classe ativa a partir do clique da tecla Enter ou Space, é necessário remover essa classe logo após deixamos de pressionar a tecla. Para isso, você precisa adicionar na tecla o evento de onkeyup (quando o usuário solta a tecla) uma função anônima, e dentro do escopo desta função, vamos remover a classe ativa da classe, ficando assim: tecla.classList.remove(‘ativa’);

CONDIÇÕES NO CODIGO

[00:00] Até o momento conseguimos adicionar a classe ativa utilizando teclas do nosso teclado. Então ao navegar com o teclado no Alura Midi, por exemplo, com o “Tab” e eu apertar o “Enter” do meu teclado conseguimos acionar o som e também acionar o estado de tecla ativa, que é representado por essa cor vermelha e que já estava sendo representado com o clique do mouse pelo comportamento padrão do CSS, que aciona aquele estado “active”.

[00:40] Porém, com o teclado isso não ocorria, portanto foi necessário adicionar via JavaScript a classe ativa para que houvesse a consistência de estilos quando usuário estivesse navegando via teclado e mouse.

[00:58] Então agora, neste momento qualquer tecla do teclado é capaz de adicionar essa classe ativa e vamos precisar agora resolver qual é o momento certo de adicionar a classe ativa, que não é a cada tecla do meu teclado e sim, apenas se for pressionado a tecla “Enter” ou a tecla barra de espaço, que são as que já têm programado para acionar o som nativamente pelo navegador e reproduzir o som dos instrumentos.

[01:39] Para começarmos a solucionar esse problema, vamos voltar no “main.js”, no Visual Studio Code. E dentro do for, onde temos os eventos do teclado, o onkeydown, o onkeyup, no onkeydown, que é quando adicionamos a classe ativa, precisamos acionar esse código tecla.classList.add de acordo com uma dada condição. E qual é a condição? É se a barra de espaço ou a tecla “Enter” do teclado do computador forem pressionadas, e não a qualquer tecla, que é o que está acontecendo aqui.

[02:28] Eu não estou delimitando qual tecla do teclado, estou dizendo para o JavaScript que qualquer tecla que for apertada execute este código que está na função atrelada para o onkeydown. Então no nosso caso temos uma condição para esse trecho de código a ser executado.

[02:53] E no JavaScript, assim como em qualquer linguagem de programação, temos uma estrutura condicional que pode nos ajudar a fazer esse tipo de situação em código, é super trivial, é comum que isso aconteça.

[03:07] Então podemos até escrever em português por enquanto, por exemplo, "se dada a condição que eu vou me informar para o JavaScript entre os parênteses for verdadeira, então se (dada condição for verdadeira) { faça o trecho de código que estará entre as chaves dessa condição".

[03:42] Mas sabemos que o JavaScript é uma linguagem de programação, então o código das estruturas precisa ser em inglês, então esse “se” em inglês é o famoso if, que como eu já comentei, tem em várias linguagens de programação. Então esse código só vai ser executado se a condição que eu informar nos parâmetros for verdadeira. E agora precisamos descobrir como montar essa condição. E um pedaço dela já sabemos, que é se a tecla do teclado for espaço ou for “Enter”, eu quero que isso aconteça.

[04:29] Então vamos ter que de certa forma ter a informação dessa tecla e compará-la com o nome dessa tecla, por exemplo espaço. Vou colocar em português por enquanto, porém logo vamos descobrir qual vai ser o nome correto. Então eu coloco if (‘Espaço’) {. Agora vamos precisar descobrir qual tecla o usuário apertou do teclado dele para conseguir terminar de montar essa condição.

[04:59] No JavaScript temos um recurso que a própria linguagem e o próprio executor ou o navegador que executa o código JavaSript fornece para trabalharmos com os eventos. Então esse recurso está disponível para todas as funções que são diretamente atreladas a um evento. E dentro dessa função, temos o parâmetro dela, é fornecido para ela como primeiro parâmetro os detalhes do evento que foi acionado e podemos dar um nome qualquer a este parâmetro que vai representar o evento que foi acionado.

[05:41] Normalmente você vai encontrar em códigos que você for trabalhar como event, ou apenas como "e", e aqui vamos dar o nome significativo, vamos escolher aqui evento em português, para você saber que esse nome eu escolhi e que dentro desse nome o JavaScript vai fornecer todos os detalhes do evento que foi acionado. tecla.onkeydown = function (evento) { ...

[06:07] E vamos usar aqui aquela função console.log para entendermos o que tem dentro desse evento que foi fornecido pelo JavaScript para conhecermos os eventos que estão acontecendo, os detalhes do evento que está acontecendo ao clicar em uma tecla do Midi, então console.log(evento).

[06:26] Então vou salvar e vou voltar no navegador. Recarreguei o navegador, eu estou no Firefox, vou dar a tecla “Tab” e vou dar de novo a tecla “Tab”, vou pressionar de novo no meu teclado. Perceba que foi imprimido no console os detalhes desse evento do keydown, vou abrir aqui nesse triângulo cinza que expande todos os detalhes desse evento.

[06:57] Podemos ver que tem aqui o objeto desse evento, ou seja, uma série de propriedades com valores que guardam informações sobre esse evento e aqui conseguimos dar uma olhada, eu sei que tem muitos, mas conseguimos ver que tem algumas propriedades que tem informação desse evento, por exemplo, eu pressionei no teclado o “Tab”, então ele me falou que foi através da propriedade code o “Tab”.

[07:26] Também consigo ver que tem outras propriedades que tem essa informação como a key, também tenho o a keyCode, que dá um número para essa tecla que foi pressionada. Então também tem o tipo do evento que foi keydown, tem esse which que também mostra o número, por baixo dos panos também tem números para o nome das teclas, mas é melhor usarmos nomes com significado.

[07:51] Então agora, por exemplo, vou clicar no teclado uma das teclas que queremos usar no Midi. Então eu vou voltar aqui no foco da tela e vou apertar o “Enter”. Vai reproduzir o som. E conseguimos ver que no code tem o nome “Enter”, no key também e vamos testar agora com a tecla de espaço.

[08:25] Também. Então vimos que vai ter o code: "Space", porém, no key a tecla de espaço é representada com aspas e um espaço vazio, o que pode ser um pouco complicado. E isso nos faz tomar a decisão correta de qual dessas propriedades podemos usar para trazer no JavaScript e montar a nossa lógica. Eu gosto muito mais desse code, porque tem um nome escrito da tecla que foi pressionada, ao contrário de ser o que a tecla faz mesmo, que é o que é essa propriedade key mostra, ela adiciona um espaço.

[09:03] Podemos até testar com outras teclas do teclado, por exemplo uma letra do alfabeto, por exemplo G. Vai mostrar code: "KeyG", key ele vai mostrar o G. Então isso para mim é o critério que vou usar de qual dessas propriedades do evento eu vou utilizar. Então agora voltando lá no “main.js”, no Visual Studio Code, esse evento eu vou entrar nele com o ponto e vou acessar a propriedade code e vou imprimir no console, então console.log(evento.code).

[09:39] Então vou voltar no console, vou recarregar a minha página, vou voltar no navegador, vou recarregar a página e vou pressionar a tecla só para vocês verem o que acontece, isso tem que estar com foco no botão. Então vou clicar “Tab” e vou apertar, por exemplo, a b c d, “Enter”, espaço, “tab”.

[10:03] Então vemos que em tempo real, no mesmo momento que pressionamos uma tecla do teclado, já é impressa qual tecla, a informação de qual tecla o usuário apertou. E agora então temos a informação necessária para criar a condição dentro do if. Então agora dentro do if eu vou fazer if (evento.code, precisa ser igual ao texto, eu escrevi “Espaço”, mas sabemos que é em inglês, Space, então if (evento.code ‘Space’) { para conseguir executar o código tecla.classList.add(‘ativa’);.

[10:44] Só que faltou o operador de igualdade para fazer essa condição verdadeira. Então eu preciso colocar esse operador de igualdade em if (evento.code ‘Space’) {. Normalmente, quando viemos da Matemática, usamos o operador de igual =, evento.code é igual a Space? Só que sabemos que no JavaScript o operador, o igual não tem essa função de igualar, mas sim de atribuir, de adicionar o que tem do lado direito para dentro do lado esquerdo.

[11:23] Então nesse caso é a mesma coisa, eu preciso usar o operador de igualdade, não de atribuição, que é representado pelo igual igual ==. Só que existem dois operadores de comparação, esse operador de igualdade já vai me retornar, vai fazer essa conta e vai retornar verdadeiro ou falso, uma condição verdadeira. Inclusive eu posso colocar essa operação no console.log para eu ter certeza. Então event.code == ‘Space’, vou salvar, vou recarregar no navegador e vou dar um foco em uma tecla do Midi e vou pressionar o espaço.

[12:06] true, verdade, eu apertei “espaço”. Agora vou apertar “Enter”. false. Ou seja, a condição já deu falso. Eu vou apertar qualquer tecla, por exemplo o H. false de novo. Ele agora incrementou para dois (false). Vou apertar “Space” de novo. Então vou clicar aqui no “Space”. true.

[12:26] Então verificamos no console que essa "conta" evento.code == ‘Space’, vai retornar true. Vai verificar se a condição for verdadeira. Então agora vamos ver se vocês notaram que quando estou apertando outras teclas do teclado, então vou voltar no foco, vou apertar J e não está ficando vermelho, K, não está ficando vermelho, “Enter”, vai reproduzir som, mas não vai ficar vermelho. Isso no Firefox, vou apertar “espaço”, ficou vermelho. Porque a condição foi verdadeira.

[13:02] E também já temos que colocar para a teclar “Enter”. Mas antes vamos conhecer um pouco mais sobre esse tipo de operador. Existem dois tipos de operadores de igualdade no JavaScript o “==” e o “===”, três vezes. E este é o mais seguro para utilizarmos para nossas lógicas. Por quê? A diferença entre eles, vou explicar no console do navegador. Vou limpar meu console e vou explicar para vocês.

[13:37] Por exemplo, eu tenho o número 1, eu posso escrevê-lo com ou sem aspas. Porém, qual é a diferença? Sem aspas ele é um número do tipo Number e com aspas ele vai ser um texto com o número 1 dentro. E se eu usar o operador de igual igual, 1 == “1” ele vai comparar se o valor 1 é igual ao valor 1 que está dentro do texto e ele vai retornar true, porque realmente ele está comparando apenas valores, então verdade.

[14:11] Agora se eu comparar 1 (número) com 1 (dentro de um texto), 1 === “1”, falso, porque o número 1 é diferente de um texto que tem escrito dentro o número 1. E tão com o operador de três iguais ele vai comprar não só o valor, mas também o tipo do dado, se ele é um texto ou se é um número, então se eu comparar 1 (número) com outro número 1, 1 === 1, aí sim vai ser verdadeiro.

[14:44] Então esse é operador mais seguro para usarmos em comparações de igualdade. Agora que sabemos como criar uma condição no nosso código e ainda com base em uma informação que vem no evento do teclado, precisamos continuar nosso código para a tecla “Enter” e perceber como reduzir repetições do código com o operador lógico ou. Vejo você lá.

OPERADOR LOGICO

[00:00] Anteriormente criamos uma condição em nosso código com if para adicionar a classe ativa em um botão em um botão do nosso Alura Midi que fora acionado por uma tecla do nosso teclado do computador. Essa tecla que escolhemos por enquanto foi a tecla Space. Agora precisamos repetir ou replicar esta mesma lógica para a tecla Enter.

[00:31] A primeira coisa que vamos fazer é usar essa mesma estrutura que já conhecemos, a estrutura condicional if, if () {}. sempre recomendo você seguir essa ordem de escrita do if para não esquecer de nenhum detalhe dele. E dentro da condição, dos parênteses, que tem a condição do if vamos fazer da mesma forma que fizemos para o Space, if(evento.code ===’’) e vimos na aula anterior, que para o “Enter” era a palavra, “Enter” sempre começando com a letra maiúscula.

[01:15] Lembrando que o JavaScript é sensitivo a letras maiúsculas e minúsculas. Entao if (evento.code === ‘Enter’){ vamos colocar a classe ativa. Então tecla.classList.add(‘ativa’);. Vou salvar, vou voltar no meu navegador, estou no Firefox, vou recarregar o código e vamos ver agora se vai acionar a nossa classe ativa quando apertar o “Enter” do meu teclado do meu computador. Então tem que ficar vermelho o fundo.

[01:58] Inclusive eu posso depois olhar e verificar a inserção dessa classe usando o inspecionador. Vamos ver então se vai acionar o “pom”, não só o som, mas também como o fundo vermelho. Deu certo e vou colocar a barra de espaço.

[02:17] Deu certo também. Vocês devem ter notado que console.log true e \*false no console. Vamos ver de onde está saindo esse console.log. É da linha 20, se olharmos no navegador está falando a linha 20 do “main.js” do console do navegador, voltamos no “main.js”, no Visual Studio Code tem esse console só comparando se é igual “Space”.

[02:41] Então vamos retirar esse console.log para ele não nos confundir, salvar e depois vou carregar o navegador, e vamos olhar também essa modificação dentro da nossa aba inspector do DevTools, do navegador. Então se eu olhar todas as minhas teclas button, quando eu pressionar com o teclado as teclas corretas, que são o “Enter” ou o espaço, vai ter que inserir a classe ativa.

[03:11] E assim que eu soltar vai ter que remover também. Então “Tab” não está acontecendo nada, se eu aperto teclas do teclado que não são “Enter”, por exemplo, do alfabeto, números, não está acontecendo nada. Agora vou apertar o “Enter”. Virão? Eu apertei no “tim”. Então vou volta no foco, espaço, deu certo.

[03:40] Mas voltando no Visual Studio Code, percebemos que ao fazer e resolver essa situação de programar essas duas teclas temos código repetitivo. Então duas vezes o mesmo código e isso não é legal. Uma forma de resolvermos isso é conseguir usar um operador lógico do JavaScript que vai conseguir fazer uma condição de verdadeiro ou falso um pouco mais abrangente para o nosso if.

[04:15] Porque a ideia seria a seguinte: Se o código do teclado, se a tecla que foi clicada for igual a "Space" ou igual a “Enter”, então adicione essa classe ativa. Então basicamente o que precisamos é manter essa condição de “Enter” e não repetimos o if e não repetimos ativa.

[04:45] E essa condição do Ou podemos colocar dentro do primeiro if utilizando algo que representa essa condição lógica o ou, esse que é conhecido como um operador lógico no JavaScript. Por exemplo, se fosse or, que em português significa “ou”. Então seria se a tecla do teclado for "Space" ou se a tecla do teclado for “Enter”, então adicione a classe ativa. if (evento.code === 'Space' or evento.code === 'Enter')

[05:20] Só que no JavaScript or é uma palavra que não existe, não está prevista, então temos um símbolo que representa esse operador de alternância, que é o pipe pipe , duas vezes esse carácter |, que é a barra reta. E o que isso vai acontecer? Qual é a lógica que vai rolar aqui? if (evento.code === 'Space' || evento.code === 'Enter')

[05:42] Esse operador vai indicar se for a tecla de espaço verdadeira ou se a tecla de “Enter” for verdadeira, então vai ser a condição suficiente para dar verdade, vai ser o suficiente para entrar dentro desse if.

[05:58] Então nesse caso precisamos ter apenas uma verdade, uma condição verdadeira para poder entrar. Então quando eu apertar a tecla de "Space", eu não estarei apertando a tecla de “Enter” junto, então sempre um desses lados vai estar falso.

[06:14] Mas o operador ou vai poder permitir que essa condição seja verdade e ativar para as duas teclas. Vamos ver, vamos recarregar, agora vou dar o foco, vou apertar aqui no puff, “Enter”.

[06:32] Deu certo. E no espaço também deu certo. Agora que já aprendemos a criar estruturas condicionais podemos ainda explorar mais as funcionalidades do nosso código e deixá-lo mais seguro, conhecendo ainda mais possibilidades do JavaScript. Vejo você lá.

MAIS CONDIÇÕES

[00:00] O nosso Alura Midi está lindo! Com todas as funcionalidades prontas, ou seja, todos os botões estão reproduzindo o som dos seus respectivos instrumentos, e também a navegação está completa, podendo ser feita via mouse ou via teclado e os estilos consistentes.

[00:20] Agora nos resta melhorar um pequeno probleminha da nossa aplicação que está na função tocaSom. E eu vou mostrar no console do navegador para vocês. Podemos chamar essa função diretamente pelo seu nome, tocaSom. Idealmente passamos o ID de um elemento audio, mas o que vai acontecer se eu passar como parâmetro um ID ou uma classe de um elemento que não está declarado no nosso documento HTML?

[00:56] Por exemplo, classe piano, tocaSom('.piano'). Não existe ninguém com essa classe no documento, então vou dar um “Enter” e vai me retornar um problema dizendo que o document.querySelector de piano é nulo.

[01:12] Lembrando que esse seletor é passado, vou mostrar para vocês a nossa função tocaSom que está na linha 1 do código “main.js”. Então recebemos um seletor e logo passa para o querySelector buscar o elemento e depois dar play.

[01:32] O outro problema é justamente essa parte do play, então vamos retornar no console do navegador e eu vou chamar a função tocaSom e passar um seletor de um elemento que existe no nosso documento HTML, porém não é uma tag de áudio, um elemento áudio. Então por exemplo, se eu passar button o que vai acontecer?

[01:58] O play daquele elemento não é uma função, ou seja, ele vai dizer que o resultado do querySelector, que é o botão, não tem uma função play, é basicamente isso que está me dizendo. Portanto, essa função precisa ser aprimorada e precisamos tratar estes problemas com algumas estruturas com o que já vimos anteriormente, mas agora aplicada aqui para tratar problemas de código.

[02:28] Vamos voltar no Visual Studio Code. E uma pequena coisa que podemos já arrumar, sabendo que não estamos recebendo apenas IDs de elementos áudio, estamos recebendo seletores, podemos simplesmente nomear esse parâmetro para algo mais consistente. Claro que podemos dar um nome que ajude a pessoa que use a função tocaSom diretamente entenda o que ela tem que passar, por exemplo, seletor de uma tag áudio.

[03:06] Porém a pessoa pode tentar trapacear e passar um outro tipo de tag para tentar encontrar algum problema, alguma vulnerabilidade em nosso código, em nossa aplicação.

[03:19] E eu vou tirar esse play por enquanto do querySelector, vou deixar para usarmos depois. E sabemos que então esse código do document.querySelector vai procurar um seletor e já vimos no console do navegador que dependendo do que foram passado pode retornar um nulo ou vai retornar que não consegue dar o play, ou seja, vai retornar um elemento.

[03:44] Então eu vou guardar o retorno desta busca dentro de uma referência constante, vou criar uma constante, vou chamá-la de elemento e vou armazenar este retorno para poder ser mais fácil trabalharmos no nosso código mais para frente. const elemento = document.querySelector(seletorAudio).

[04:04] E agora temos uma condição para executar o nosso código. Por exemplo, o que queremos fazer é chamar o elemento e dar um play nele, porém eu só posso fazer esse código elemento.play(); se o elemento realmente existir. Porém, se ele não existir sabemos que o valor dele vai ser nulo, portanto, antes de executar esse código podemos verificar se o elemento está nulo.

[04:33] Então se temos uma condição para fazer alguma coisa caso o elemento seja nulo ou não, podemos usar a estrutura do if que já conhecemos, if (elemento === null) então se o elemento for estritamente igual a nulo, e o null escrevemos assim diretamente, sem aspas nem nada. E agora temos que pensar o que vamos fazer se o elemento for nulo. Para não dar um erro para o usuário podemos mandar uma mensagem avisando do problema, por exemplo, usando o alert ou podemos o console.log.

[05:23] Vamos testar com alert, por exemplo, alert(‘Elemento não encontrado’), vou salvar isso e vamos testar no navegador. Então vou recarregar e vou passar a função tocaSom com um elemento que não existe, nossa classe piano.

[05:46] Ao executarmos a classe piano, a função com a classe piano, vemos que realmente o nosso alert funcionou. Claro que eu fui bem dramática, coloquei um alert, mas poderia ser um console.log, se queremos ser mais discretos, por exemplo, console.log(‘Elemento não encontrado’), então fica a critério do que você acha que vai ser a melhor experiência para o usuário de vocês do Midi. Então vou colocar um barra barra e vamos testar de novo tocaSom passando o piano.

[06:23] Elemento não encontrado, então tem essas duas formas de apresentar um problema para o usuário. Outra coisa que temos que fazer agora é quando o elemento é diferente do nulo, ou seja, se o querySelector retorna um elemento. Então podemos usar o if também, if (elemento ) diferente de nulo. Temos um operador novo que não conhecemos que é o diferente. Diferente no JavaScript, ou not equals, “não é igual”, é denotado por exclamação e igual, junto, e do outro lado o que vamos comparar, então if (elemento != null).

[07:07] Então se elemento não for igual a nulo, faça alguma coisa, por exemplo podemos dar o elemento play, que é o que queríamos fazer. Vou colocar esse código elemento.play de volta no nosso if e podemos testar na nossa aplicação. Então se o elemento não for nulo eu quero dar o play. Então eu vou passar aqui um hashtag, a hashtag de uma tag áudio que já existe, só para vermos se tá funcionando de volta a nossa funcionalidade, tocaSom ('#som\_tecla\_tim'), vocês vão ver o “tim” agora.

[07:51] Faltou dar o clique na nossa aplicação, então tem que dar um clique antes de executar a função, agora vai.

[07:59] Agora tocou. Agora se eu passar um elemento que existe e for um elemento que não tem o play, por exemplo, o button. Ainda temos o problema que play não é uma função desse tipo de elemento. Então temos que se certificar que o elemento que está sendo passado é do tipo áudio, é uma tag áudio para podermos aí sim executarmos o play nela.

[08:27] Então como poderíamos fazer isso? Ou seja, descobrir qual é o elemento que foi encontrado pelo querySelector. Então podemos usar o console.log para explorar a estrutura de um elemento. Então vou colocar o console.log(elemento) no elemento e vou comentar esse código do elemento play por enquanto só para não termos um erro. E vamos explorar o que tem dentro de um elemento. Vou salvar, vou recarregar e vou chamar função tocaSom.

[09:00] Eu estou no Mozilla Firefox e vocês vão perceber que no Firefox, quando eu dou o console.log de um elemento ele vem com esse triângulo aqui. Isso significa que eu posso expandir essa tag e olhá-la no formato de um objeto JavaScript que tem todas as propriedades e as funcionalidades que são possíveis executar para um elemento ou consultar, no caso de propriedade. Então por exemplo, as funcionalidades que já colocamos nos nossos botões, podemos verificar nos atributos.

[09:35] onkeydown, a função que fizemos, o onclick, a função que programamos também. E então também conseguimos ver propriedades que já são fornecidas pelo próprio JavaScript e HTML.

[09:51] Por exemplo, eu tenho uma propriedade chamada tagName, que tem o nome da tag “BUTTON”, porém está em maiúsculo. Se eu der mais uma olhada por tudo, tenho esse nodeName, também tem “BUTTON” em maiúsculo e tem essa localName, que também tem o nome, porém minúsculo. Como o JavaScript é case sensitive, ou seja, tem essa sensibilidade entre maiúsculo e minúsculo, eu prefiro usar essa localName, porque ela é toda minúscula.

[10:25] Eu também posso olhar no Chrome para ver como fica isso. Então eu abri o Chrome, o console, vou recarregar minha página e vou executar tocaSom passando tocaSom(‘button’) para ver o que acontece.

[10:36] No caso do Chrome ele não aparece esse triângulo, isso não quer dizer que não funcione explorarmos o objeto, ele está disponível para usarmos, porém o Chrome ele não oferece aquele triângulo para explorarmos e essa é uma diferença entre o Chrome e o Firefox na hora de implementarmos códigos com JavaScript. E o objeto do elemento é esse conjunto de propriedades e funcionalidades que estão por trás de um elemento.

[11:11] Então vamos agora no “main.js”, vamos dar um console.log(elemento.localName), e agora qualquer elemento que eu passar lá precisa exibir no console o nome desse elemento, ou seja, o nome da tag desse elemento. Então vou recarregar o Firefox e vamos passar tocaSom(‘button’).

[11:35] "button", porque o seletor é o próprio tag name. Agora se passarmos, por exemplo, o som do tecla “tim”, que é a tag name do elemento que está com ID, é áudio. Vamos ver se vai dar certo. tocaSom ('#som\_tecla\_tim')

[11:49] "audio". Deu certo. Mas vamos tentar a mesma coisa lá no Chrome só para vermos e ele vem como ‘button’, deu certo também. Então agora eu já descobri qual a propriedade aonde esse elemento guarda a informação do nome. E se for um elemento do tipo áudio, queremos que dê o play.

[12:13] Então lembra que temos aquela comparação == ‘audio’, queremos que fique play, console.log(elemento.localName == ‘audio’). Estou fazendo essa lógica dentro do console, então no momento eu vou imprimir no console true ou false. Então se eu se eu fizer isso no Chrome, vou passar tocaSom(‘button’).

[12:32] Falso, não é igual a "audio". Agora se eu passar o ID da tag “tim” vai tocar o som, ele é igual a nossa tag áudio. Então agora vamos seguir para o nosso código.

[12:49] Então se o elemento não for nulo e for igual a tag áudio, podemos colocar isso dentro de um outro if, vou tirar esse console, agora eu posso dar o play neste som.

if (elemento != null) {

if (elemento === 'audio') {

elemento.play();

}

} COPIAR CÓDIGO

[13:12] Então vamos executar o som do “tim”, então tocaSom(‘#som\_tecla\_tim’). E tocou. Agora vamos tentar com o piano e elemento não encontrado. E depois também temos aqui com o ‘button’. O Button por enquanto não colocamos nenhuma mensagem de erro caso seja um elemento inválido, mas vamos chegar lá.

Estruturas condicionais

PRÓXIMA ATIVIDADE

Uma empresa de transporte solicitou que você desenvolvesse uma função que entregue todos os pacotes vermelhos no Ceará, e os azuis no Rio de Janeiro.

Assinale a alternativa correta que corresponde a função de entrega do pacote:

Selecione uma alternativa

function entregaPacote(corDoPacote) {

if (corDoPacote == 'vermelho') {

mandaProCeara();

}

if (corDoPacote == 'azul') {

mandaProRioDeJaneiro();

}

}

Alternativa correta! Utilizando a estrutura condicional ifconseguimos saber o destino correto para entrega dos pacotes a partir das cores.

MELHORANDO O CODIGO

[00:00] Vamos primeiro só dar uma melhorada no nosso código. Ficou um monte de if. Podemos reduzir esse if em um único if, usando uma coisa muito parecida que fizemos nossos eventos de teclado, que juntamos dois ifs em um usando o “ou”. No nosso caso podemos fazer a mesma lógica, só que ao invés de usar “ou”, porque o elemento não pode estar nulo e\*\* tem que ser igual ao áudio, vamos usar o operador lógico \*and, que significa \*“e”, que soma “e” em português;

[00:36] Se o elemento não for nulo e aí ele é representado por dois E comercial &&, esse é o símbolo do and, vou até escrever and para vocês. E se o elemento não for nulo e elemento.localName === ‘audio’), então eu vou dar play. Então eu posso reduzir tudo isso em um único if.

[01:05] E também percebemos que esse if é a negação desse outro if, ou seja, se o elemento for nulo, então eu faço isso. Ou seja, se em algum momento o elemento não bater nisso, se o elemento for falso, eu vou ter uma outra condição falsa e eu posso no if usar uma estrutura conjunta que é o else.

[01:34] O else significa “senão” e ele precisa ser invocado imediatamente depois de um if, que vai ser a função que vai ser executada caso esta condição que seja falsa. E tão se o elemento não for nulo e o elemento for diferente do áudio, então eu vou cair no else.

[02:01] Então eu posso passar essa mensagem aqui de elemento não encontrado para dentro do else desse if, posso tirar esse == nulo, porque o else vai substituir dentro da nossa lógica. E agora essa mensagem vai cair para os dois erros, tanto para o button quanto para o piano.

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

elemento.play()

}

else {

console.log('Elemento não encontrado');

}COPIAR CÓDIGO

[02:21] Vamos testar? Então eu vou voltar no meu navegador, vou recarregar e vou testar o piano, “Elemento não encontrado”. E o button? Ele está dando a mesma mensagem para o button, então podemos colocar uma mensagem um pouco mais detalhada, “elemento ou seletor inválido”. “Elemento não encontrado” ou “Seletor inválido”, por exemplo. console.log('Elemento não encontrado ou seletor inválido');

[02:56] Podemos colocar a mesma mensagem para os dois. Então vou recarregar e vamos tentar agora com piano e com button. Temos a mesma mensagem, uma mensagem mais significativa. Então agora caso alguém use essa função tocaSom de maneira inapropriada, ela vai ter os detalhes do porquê não está funcionando a nossa função.

[03:23] Então vimos o if, o else e também o operador and e o operador de not equals. Apenas um detalhe "muito mágico", o JavaScript já é inteligente o suficiente para quando o valor for nulo ou for vazio dentro de uma referência.

[03:46] Então no caso, se isso (elemento) for nulo eu não preciso compará-lo com nulo, eu posso simplesmente retirar isso e ele vai fazer, o if já é inteligente para fazer essa verificação, ele pensa if elemento, se elemento existe, ou seja, tem alguma coisa que seja um valor que não seja nulo, que não seja vazio, que testou vazio ou zero ou false ou null ou undefined, se for diferente de tudo isso, então vai cair como uma condição verdadeira. Então eu posso deixar nossa lógica um pouco mais sucinta. E para essa aula é isso.

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

elemento.play()

}

else {

console.log('Elemento não encontrado ou seletor inválido');

}C

Nessa aula, você aprendeu:

Eventos no teclado:

O que são eventos do teclado e como usá-los: onkeydowne onkeyup. Como adicionar e remover classes em um elemento HTML através do JavaScript, com as funções add e remove do classList.

Condições no código e operadores lógicos:

O que é o objeto event, como declarar e acessar ele através do parâmetro de uma função atrelada a um evento. A estrutura condicional if e para que ele serve, além de conhecer o operador de igualdade ==, estritamente igual (===), e o operador or (||).

Mais condições:

A estruturas condicionais if e else juntas. O operador not equals (!=), operador lógico and (&&) e o valor null.