[00:00] Olá, o meu nome é Guilherme Lima e eu serei o seu instrutor nesse treinamento de métodos de array com JavaScript.

[00:05] Nesse curso vamos trabalhar no AluraBooks. Temos uma API que vai disponibilizar uma lista de objetos e o nosso desafio vai ser pegar as informações que estão naquela lista, exibir essas informações na tela, que começaremos com um projeto base, com uns arquivos estáticos.

[00:23] Criar filtros para livros de front-end, back-end, livros de dados, livros disponíveis, manipular elementos na tela através do DOM e o JavaScript. Ordenação de preços, tudo isso e vamos focar em alguns métodos especiais do JavaScript como: o método filter, forEach. map, reduce e sort.

[00:44] Esse curso voltado a pessoas que não tiveram contato com esses métodos especiais do JavaScript. E se você já tem contato também, eu recomendo que você faça esse treinamento, porque sempre tem algo muito interessante para aprendermos e analisarmos também.

[00:59] Esse curso está incrível, está com um leiaute muito bonito e ele está um curso bem leve. Espero que você goste, abriremos bastante a documentação, vamos deixar esses links da API para você conseguir praticar na sua casa também. E eu tenho a certeza de que vai ser um momento de aprendizado muito bacana entre nós. Vamos lá?

**Olá!**

É muito bom receber você neste curso de JavaScript: manipulação de array.

Espero que seja uma experiência de aprendizado incrível e que possamos vencer todos os desafios juntos. Neste curso, vamos aprender de forma prática como utilizar os métodos de array com JavaScript.

**Preparando ambiente**

Recomendo o uso do [VSCode](https://code.visualstudio.com/" \t "_blank) para editar o código e, para tornar o desenvolvimento ainda mais fácil, recomendo a [instalação da extensão do Live Server](https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer).

Para aprofundar nossos conhecimentos na linguagens JavaScript, vamos disponibilizar o [projeto base deste treinamento que está no GitHub](https://github.com/alura-cursos/alura_books/archive/refs/heads/projeto_inicial.zip).

A Alura é formada por pessoas que gostam de tecnologia e acreditam no poder da educação através dela. Somos uma comunidade que ama compartilhar conhecimento. Em caso de dúvida na instalação ou durante o curso, conte sempre com o fórum. Caso não tenha dúvidas, não deixe de participar do fórum ou interagindo no nosso servidor do Discord para ajudar outras pessoas e fazer da comunidade um lugar ainda melhor!

**Vamos começar?**

[00:00] Vamos iniciar os nossos estudos sobre métodos de *array* com JavaScript. Seu primeira passo é acessar o link "preparando ambiente" na atividade anterior a esse vídeo, lá vocês podem fazer *download* do projeto base que utilizaremos durante todo o nosso treinamento.

[00:14] Eu já fiz o download e o nome está "alura\_books - projeto\_inicial". Vou renomear esse download só para "alura\_books" e vou usar o Visual Studio Code para editar o nosso código. Então eu vou arrastar esse alura\_books para dentro e veremos o que é que temos para mexer nessa aplicação, nesse projeto.

[00:33] Então temos uma pasta, com várias imagens. Temos um "index.html". Logo mais entenderemos o que é cada etapa, o que são esses botões, como a estrutura está feita. Temos um arquivo que se chama "equisicao.txt", que eu já vou falar para você o que é. E o "style.css" para os estilos da nossa aplicação.

[00:52] Estou com o meu projeto base, com o *download*, o que eu preciso fazer agora é, de alguma forma, carregar esse projeto para conseguirmos visualizar o que mexeremos durante todo o nosso treinamento.

[01:03] Para isso, eu vou usar a extensão do Live Server, eu vou colocar *live*. Nesse primeiro link, Live Server. Eu vou instalar, o meu já está instalado, é só você instalar essa extensão. Eu vou clicar com o botão direito em cima - deixe-me fechar essa janela - do "index.html", eu vou colocar "abrir com o Live Server" ou "*open with Live Server*".

[01:24] Eu vou carregar essa aplicação. Carregou a aplicação, vamos ver o que é que temos. Temos um cabeçalho em cima com o título e subtítulo: "AluraBooks, melhores livros de programação, encontre na nossa estante o que você precisa para o seu desenvolvimento".

[01:39] E temos alguns botões: "Livros de front-end", "Livros de back-end", "Livros de dados". Eu clico nesses botões e não acontece absolutamente nada, daremos vida para esses botões com o JavaScript.

[01:50] E tem alguns livros, então tem o "Cangaceiro JavaScript", tem o outro "Cangaceiro", repara que ele está com uma opacidade diferente. Então ele está mais fraco para conseguirmos visualizar, provavelmente é porque esse livro não está disponível. Em baixo tem algumas informações.

[02:06] O título do livro, o autor, o valor desse livro e tem uma *tag* das categorias. Então tem *front-end*, *back-end*, repare que essas *tags* estão todas não verdadeiras. Por exemplo, temos uma *tag* para o livro de SASS, que é para os pré-processadores de CSS, com *back-end* também, não faz muito sentido. É meramente ilustrativo.

[02:28] E embaixo temos todos os livros disponíveis por 299 reais. Alura, mergulhe em tecnologia. Esse é o visual, todo esse visual que estamos vendo está estático. O que isso significa? Que ele está direto no HTML, olha só: temos os nossos botões e, quando tem a seção de livros, tem o "Cangaceiro JavaScript" no livro\_\_titulo, tudo isso está estático, ou seja, toda essa informação que estamos vendo na tela está diretamente no HTML.

[02:57] E no mundo real, não é isso que acontece. Geralmente buscamos essas informações de um banco de dados, ou de uma outra aplicação, ou de uma API. Então, o que é que faremos? No nosso treinamento, temos esse arquivo requisicao.txt. Repare que tem um endereço: https://guilhermeonrails.github.io/casadocodigo.livros.json.

[03:19] Eu vou clicar com o "Ctrl" e vou clicar em cima desse link para abrirmos e vermos o que é que acontece. Repara que, eu fazendo isso, ele me levou para uma outra página que tem: título, preço, autor, imagem, alt e categoria.

[03:32] Essas informações que estão aqui, elas estão do tipo json. Podemos até ver que tem esse nome livros.json. Então, geralmente, o que acontece? Um time focado no *back-end* vai criar todo o banco de dados, a comunicação. Vai falar: olha, vai ter uma tabela, chamada títulos, vai ter um autor que esse autor criou esse título desse livro, vai fazer toda aquela mágica no *back-end* conversando com o banco de dados.

[04:00] E para pegarmos essas informações exibirmos na tela, eu não quero exibir essas informações que já estão estáticas. Eu quero exibir tudo isso daqui, eu vou precisar fazer uma requisição. Então qual que vai ser o nosso desafio? Eu vou comentar todas essas linhas nossas de livro, então olha só, escrolando para baixo, quando começar a nossa seção, onde tem o id=livros, eu vou pegar do *id* <div class="livro">, vou clicar na linha 33, vou deixar a linha 33 clicada. Vamos escrolar.

[04:33] E repare que acabou todo meu conteúdo de livro naquele *id*. Eu vou apertar o "Shift" e vou selecionar a linha 264. Agora vem uma mágica: eu vou selecionar o "Ctrl + /" ou "Cmnd + / " para eu comentar todos esses livros estáticos.

[04:50] E se eu volto na minha aplicação do AluraBooks, olha só o que aconteceu? Não está nenhum livro sendo exibido. Por quê? O que queremos fazer agora é uma requisição para exibir, de fato, esses livros que estão aqui. Então eu quero pegar todos esses livros e construir, utilizando, manipulando o DOM, cada um desses livros no AluraBook. E é isso que faremos na sequência.

[00:00] Agora que comentamos os livros estáticos do HTML que tínhamos aqui, queremos realizar uma requisição para esse *endpoint* que está anotado em nosso "requisição.txt" e pegarmos esse json e colocar os livros dentro do nosso AluraBooks.

[00:20] Para isso, vamos fazer por etapas. Primeiro queremos realizar a requisição. Vamos criar uma pasta chama "app" e dentro dela vamos manter todo nosso código JavaScript. Durante o nosso treinamento vamos ter mais de um códido JavaScript, então vamos colocar todos os códigos dentro dessa pasta para mantermos o projeto organizado.

[00:40] Dentro dessa pasta, vamos criar um arquivo chamado "main.js". Esse arquivos será responsável por de fato realizar a requisição e dizer que temos a variável livros com base na requisição já feito.

[01:00] No primeiro passo será criar essa variável chamada livros que será igual a uma *array* vazia, let livros = []. Em seguida, criamos uma constante chamada endpointDaApi que será igual ao conteúdo que temos em "requisição.txt": const endpointDaApi = 'https://guilhermeonrails.github.io/casadocodigo/livos.json'

[01:36] Em seguida, fazemos um get para ele de fato fazer a requisição e buscar os livros da API. Então, getBuscarLivrosDaAPI(), ele será uma função.

[01:54] Queremos que essa função seja assíncrona. Para isso, precisamos definir a palavra async antes da palavra function. Em seguida, damos o nome da função: async function getBuscaLivrosDaAPI(). Não teremos parâmetros nessa função.

[02:07] Queremos que essa função seja assíncrona porque em nossa API estamos utilizando um projeto com alguns livros. Mas vamos supor que tenhamos onze milhões de livros cadastrados, essa função demoraria muito e provavelmente teríamos um erro.

[02:31] Para dizer que essa função precisa esperar uma promessa de receber todos os livros desse endpoint, vamos usar o await. Passamos uma requisição e obteremos uma resposta. Então vamos chamar res para termos uma resposta e deixaremos ela igual a await: const res = await. Ou seja: esper por uma promessa

[02:48] Nós sempre utilizamos para conseguirmos manipular as funções e essas requisições com HTTP esses APIs do fetch. Como queremos realizar uma função para o endpointDaAPI vamos passar ele como argumento: await: const res = await fetch(endpointDaAPI).

[03:04] Depois que pegarmos a resposta dessa aquisição, queremos atribuí-la a livros. E livros também será igual a await e a resposta dessa requisição queremos que seja transformada em json(): livros = wait res.json(). Em seguida, damos um console.log(livros) para exibirmos isso na tela.

[03:35] Como criamos um novo arquivo precisamos ir em "index.html" e adicionar o <script src="app/main.js"></script> antes de fechar o <body>, na linha 276.Assim, vinculamos esse HTML com o JavaScript que estamos criando.

[04:06] No navegador, clicamos com o botão direito do mouse e selecionamos "Inspecionar". Em "Console" temos uma *Array*(12), ou seja, os livros estão aqui. Porém, a forma como estamos visualizando esses livros não está muito legal.

[04:20] Podemos alterar a forma que vemos eles substituindo o console.log() por console.table(). Agora ele monta uma tabela com todos os livros que temos. Assim, conseguimos ver todos os campos que utilizamos e todas as outras informações.

[04:37] Para visualizarmos melhor, vou tirar o await de de livros. Ele vai realizar uma requisição, essa requisição vai esperar uma promessa. Ele vai terminar de fazer essa requisição, mas não vai dar tempo dele montar a resposta em json. Para conseguirmos de fato visualizar essa resposta dessa promessa na nossa variável de livros, vamos usar o await.

[05:05] Dessa forma temos os livros da API. Metade dos nossos problemas foram resolvidos. Trouxemos as informações que estavam no *endpoint* para nossa aplicação. Mas agora queremos montar na tela, manipular o DOM. Como faremos isso utilizando métodos especiais de *arrays* com JavaScript? Descobriremos na sequência!

[00:00] Nosso desafio agora é pegarmos as informações do json, que a colocamos na nossa variável de livros, e exibi-las na tela. A primeira ideia que nos vem à cabeça é a seguinte: já que temos a lista de livros, o que é a que podemos fazer? Poderíamos fazer um for para falar pegarmos o livro no índice 0, que tem o título 0, que tem o preço 0 e montar na tela, essa estrutura da classe de livro que comentamos.

[00:28] Então vamos criar uma nova classe de livro, essa classe vai ter uma imagem. Quando chegar no src, queremos pegar essa imagem que está dentro do src da nossa classe, esse HTTP que é um link que contém a imagem desse livro.

[00:41] Quando chegar na parte do nome do livro, temos o título do livro e assim por diante. Fazemos esse comportamento para todas as outras classes que temos. Só que, se eu faço isso na mão, isso vai ficar muito "verboso", principalmente utilizando o for.

[00:57] Assim que percebemos a utilidade e o benefício de utilizar métodos especiais de array. E eu quero pesquisar o primeiro método junto com vocês. Vou digitar forEach. Deixe-me colocar forEach mozilla, que conseguiremos até rodar um exemplo bacana dele.

[01:12] Vou clicar nesse primeiro link array.prototype.forEach(). Ele fala assim: o método forEach() executa uma função dada para cada elemento do array. Isso parece que faz sentido para nós. O que é que nós queremos pegar? Cada elemento do array e criar para ele uma classe de livro, com todas as características que ele tem.

[01:31] E o próximo elemento? Cria mais uma classe de livros e assim por diante. Então para nós parece que faz sentido. Vamos testar, só para vermos na prática acontecendo?

[01:41] Eu quero que você faça junto comigo. Na parte superior direita da tela, onde está escrito "Português (do Brasil)", clicaremos e selecionaremos a língua inglesa. Quando selecionamos a língua inglesa, aparece um *try it*, que seria para testarmos o nosso código.

[01:55] Eu vou escrever um código, se você quiser escrever junto comigo pode, mas eu recomendo que você assista e você preste atenção no que eu vou fazer.

[02:04] Primeiro de tudo, eu vou criar uma variável chamada livros e vou falar que ela vai ser igual a uma lista e vou atribuir já os valores. Eu tenho o livro de JavaScript, temos livros de Python e temos livros também de Go. Eu vou deixar em maiúsculo, para ficar certa a escrita e vou fechar.

[02:22] Tenho uma lista de livros com três elementos. O que é que eu faria da forma clássica, sem utilizar o forEach? Eu faria um for onde let i=0;) que é o clássico. Depois eu falaria assim: enquanto i for menor que livros.length, enquanto livros for menor que o tamanho de elementos, de itens que eu tenho nessa minha lista, você vai continuar incrementando.

[02:49] E aqui eu colocaria uma chave e outra chave fechando. E aqui eu faço, por exemplo, um console.log e aqui eu coloco livros, no plural, com o índice que ele está trabalhando e fecho o parênteses. Se eu for para baixo e rodar esse código, ele vai mostrar "Js", "Python" e "Go". Maravilha, conseguimos o resultado que queríamos.

[03:13] Só que, lembra que eu falei? Essa forma é muito verbosa, olha o tanto que eu tenho que escrever para conseguir esse resultado. O que eu vou fazer? Eu vou comentar essas três linhas e vou escrever agora utilizando esse método que estamos pesquisando, que é o forEach.

[03:28] Então temos alguns exemplos das sintaxes, como é que ele funciona, quais são os parâmetros que ele utiliza. E tem algumas formas. Eu posso fazer utilizando o arrow function, que é uma das formas mais comuns de encontrarmos nos códigos.

[03:42] Então eu vou passar o forEach, vou passar aqui dentro um elemento e o que eu quero que ele faça depois. Primeiro, eu vou pegar a minha lista, o meu array livros.forEach. Lembrando que forEach é uma palavra *case sensitive*, então vai fazer diferença eu escrever o "e" com maiúsculo ou minúsculo.

[04:00] Se você observar: "e" maiúsculo; em cima também: "e" maiúsculo. Então "for" minúsculo, "e" maiúsculo e "ach" minúsculo. forEach.

[04:12] Então dentro do forEach eu tenho uma lista de livros. O *for*, em inglês, significa para; *each* significa cada. Então para cada livro - eu vou escrever livro - dentro da minha lista de livros, eu quero fazer alguma coisa, eu quero realizar alguma ação.

[04:29] Então eu vou colocar uma arrow function abrindo => e vou colocar a nossa arrow function. E aqui eu posso colocar a mesma ideia que tínhamos ali, console.log(livro) que é a manipulação que já estamos utilizando.

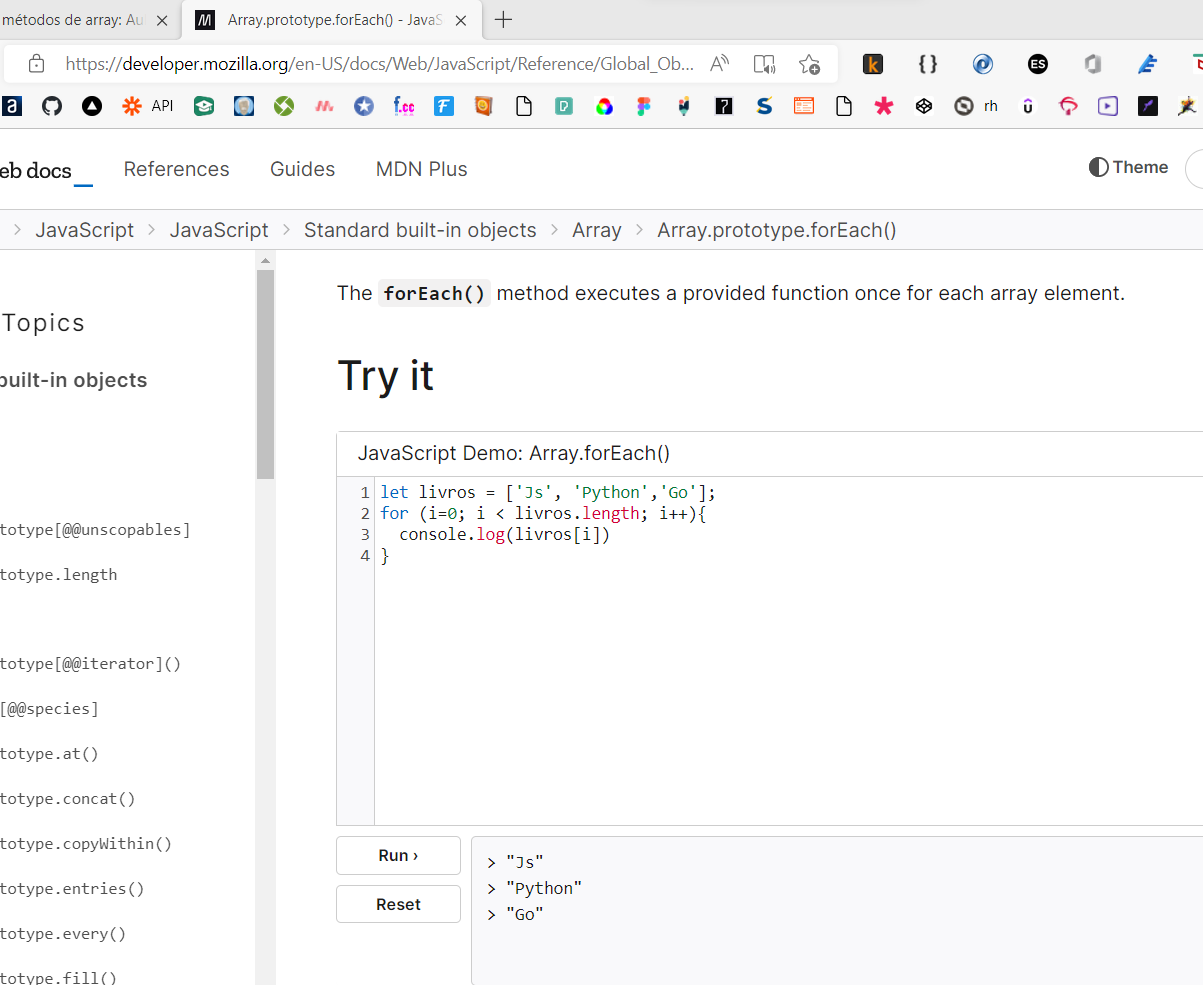
[04:43] Vou rodar esse código e temos o mesmo resultado. Puxa Gui, você fez alguma coisa, alguma mágica, eu vou fazer o meu código quebrar, para vocês verem que isso está funcionando. Olha, console.log, tirei o parênteses, eu vou rodar e ele vai falar "opa, você esqueceu de colocar um parênteses depois do argumento".

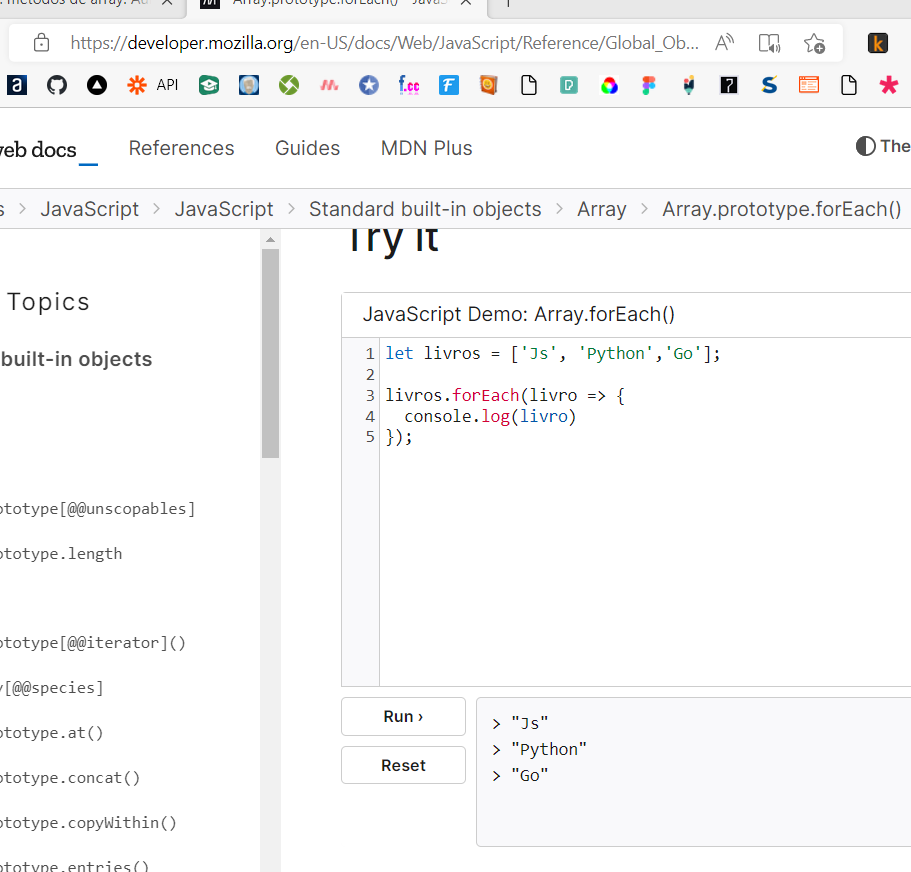
[04:58] Então eu vou colocar o parênteses, quando eu rodo temos o mesmo resultado. E embaixo tem algumas formas, podemos receber mais parâmetros como o elemento que estamos utilizando, o index e a própria array que iremos utilizar.

[05:14] Se precisarmos de algum caso específico, olha, quando tiver um index eu quero que tenha tal comportamento, conseguimos fazer isso também.

[05:23] Podemos utilizar arrowFunction ou funções callback, forEach e uma outra função call-back para utilizar essa ação em todos os elementos. O que eu quero deixar claro no coração de vocês é o seguinte: forEach, ele vai pegar cada elemento da nossa lista, do nosso array e vai executar uma determinada função que passaremos.

[05:45] Gui, deu para entender, criei livros.forEach(livro => { console.log(livro) }. Mas como é que fazemos agora para o nosso código, para criarmos, de fato, uma função com o forEach para exibir os livros na tela? Veremos isso no próximo vídeo.





[00:00] Vamos agora colocar os livros na tela? Primeira coisa que eu vou fazer vai ser pegar o elemento em que colocaremos os nossos livros. Então se observarmos, a seção 32, nós não comentamos essa section, deixamos ela aberta, livros está aberta, tanto que formos até o final, encontraremos section fechando. Olha, section fechando de livros.

[00:22] O que é que eu vou fazer? Eu vou pegar esse elemento, de section de livros e vou inserir cada livro dentro dessa section. O nosso código main para conseguirmos pegar essa section que estamos utilizando. O que é que eu vou fazer? Vou criar uma constante const elementoParaInserirLivros = document.getElementById('livros'), que é o *id* que temos no nosso HTML.

[00:56] Deixe-me abrir o HTML, que é esse *id* aqui, id="livros". Maravilha. Tenho esse *id* de livros nessa constante. Então agora o que precisamos fazer é de fato esse for, queremos exibir os livros na tela. Então o que é que eu vou fazer? Estamos utilizando o console.table, eu vou deixá-lo por enquanto, só para pegarmos algumas informações, mas depois eu vou tirá-lo.

[01:16] Então, da mesma forma que fazemos um console.table(livros), eu quero exibirOsLivrosNaTela, alguma coisa desse tipo, e eu vou passar quem? Vou passar os meus livros que eu peguei da minha requisição.

[01:31] Primeira coisa, vamos criar essa nossa função. Vai ser uma função, vai ter o nome de exibirOsLivrosNaTela, vou receber como parâmetro, vou colocar (listaDeLivros) para ficar mais claro o que eu quero fazer junto com vocês. Então vamos atacar essa função agora. O que é que eu tenho nessa função?

[01:53] Eu recebo uma lista de livros, esses livros que eu passo aqui, dentro dessa função vai chamar listaDeLivros. Então, o que eu quero fazer é pegar essa lista de listaDeLivros e fazer um forEach dele. Eu vou abrir os parênteses e vou criar, da mesma forma que eu mostrei no exemplo anterior.

[02:14] Eu vou pegar, para cada livro dentro dessa listaDeLivros, eu quero fazer uma determinada ação. Então => para criarmos a nossa arrowFunction, eu vou abrir e fechar chaves e a colocaremos o nosso código. O que é que temos? Temos o elemento para inserirmos os livros.

[02:35] Então eu vou pegar ele, elementoParaInserirLivros.innerHTML, ou seja, vai no elemento section class="livros wrap" id="livros" e o conteúdo dentro dele, quando eu coloquei inner.HTML é esse conteúdo dentro dele, eu quero colocar alguma coisa.

[02:51] O que é que eu vou fazer? Está vendo que eu tenho div class="livro"? Esse é um livro novo. Tudo isso faz referência a esse livro "Cangaceiro JavaScript". Então a imagem, o preço do livro, quem criou. Esse é um outro livro. Então se observarmos, todos os livros terão: um nome, que é o livro\_\_titulo; um alt; uma imagem; uma descrição, um preço e assim por diante.

[03:19] Então o que é que eu vou fazer? Vou dar um "Ctrl + c" dessa linha toda, vou no nosso código e vou falar: innerHTML pega os livros e insere cada livro, cada *id* que eu tenho, eu começo do 0 e vou até o 11, esses 12 livros, dentro desse elemento inserirLivros.

[03:37] Para isso, eu vou colocar o sinal de += *template string*, com a crase invertida, vou dar um "Tab" dentro e "Ctrl + v". Vou voltar no nosso código e olha o que aconteceu? Aconteceu um negócio interessante. Ele pegou e colocou 12 vezes o livro "Cangaceiro JavaScript". Vou até ocultar para vermos.

[03:57] Olha, tem "Cangaceiro JavaScript" quatro, oito, 12. Ele colocou 12 vezes dentro dessa *tag*, só que não é muito bem isso que eu quero. Eu quero as informações que estão contidas no livro que utilizamos.

[04:10] Vamos pegar então essas informações que temos no livro. Então olha só, deixe-me abrir um pouco para o lado. Vou atualizar a página. Temos algumas informações como título, preço, autor, imagem, alt, quantidade e categoria.

[04:30] Então vamos começar a alterar essas informações. Primeiro temos src da imagem do livro. Como que chama no nosso json? Chama imagem. E para eu pegar a imagem do livro específico que eu estou trabalhando, o que é que eu vou fazer? Eu vou precisar alterar, de alguma forma, esse valor. Eu preciso alterar esse conteúdo.

[04:49] Como que eu faço isso? Eu vou pegar, vou utilizar o sinal ${} e dentro das chaves eu vou escrever livro, no singular. Por quê? Não se trata da minha lista de livros, se trata do livro que eu estou utilizando naquele momento. No nosso caso, na primeira volta que ele fizer, a primeira vez que ele passar nesse *loop*, vai ser o índice zero.

[05:09] Então eu quero que ele pegue o livro.imagem. Vou voltar na nossa aplicação e repara o que vai acontecer. Temos o "Vue", o "Postgre", o "Sass" e todas as outras informações também. E tem uma parte interessante no nosso código, que ele está mostrando os livros como indisponíveis.

[05:26] Trataremos isso mais para frente. Eu vou remover esse indisponivel vou deixar só assim: img class = "livro\_\_imagens" src=${livro.imagem}". E visualizaremos cada livro de uma forma melhor. Só eu fazendo isso, já estamos vendo diversos livros diferentes. Não são todos os livros o "Cangaceiro JavaScript".

[05:47] Isso ficou incrível, ficou muito legal. Outra coisa muito importante, na API temos o alt dela. O alt das imagens que utilizamos. Tem o alt. Então eu posso fazer a mesma coisa: alt="${livro.alt}" e já temos o alt das imagens corretamente.

[06:05] Podemos até verificar, vou colocar nas imagens para vermos, nos elementos. Deixe-me atualizar, eu vou selecionar essa imagem clicando na seta, selecionando essa imagem, temos o alt: "Capa do livro de front-end com Vue.js", que é esse. Então está maravilha, está correto. Todos os alt estão corretos.

[06:29] O que é que temos? Esse é onde fica o título do livro. Então se é o título do livro ${lviro.titulo}. Descrição, está relacionado ao autor. Então ${} - deixe-me lembrar como está escrito, inspecionar, no console, eu esqueci de fechar, vou chutar - livro.autor, eu não lembro se é livro.autor. É livro.autor, então: o Caio, o Vinícius, o Nathan e assim por diante, maravilha.

[07:05] O preço? Temos ${livro.preço}. E para finalizar, temos a nossa classe de *tags*. Qual a *tag* que estamos utilizando? No nosso projeto, na API, ela está escrita como categoria. Então vamos colocar ${livro.categoria}. Vamos testar isso? Olha que incrível que ficou.

[07:35] Então já fizemos a requisição e utilizamos o forEach para criar cada um desses elementos de livro. Ele criou o livro de "Vue", o livro de "Postgre", o livro de "Saas", de "HTML e CSS" e todos os livros assim por diante. Então aprendemos como fizemos a requisição e como usamos o forEach para criar cada um desses elementos na tela.

# O que aprendemos?

[**PRÓXIMA ATIVIDADE**](https://cursos.alura.com.br/course/javascript-metodos-array/task/109847/next)

## Nesta aula:

* Carregamos o projeto base que vamos usar nesse treinamento;
* Realizamos o uma requisição para pegar todos os livros da API de exemplo;
* Entendemos que o método forEach é bastante parecido com o for antigo, basicamente faz o mesmo que ele, mas numa forma bem mais elegante;
* Exibimos os [livros da API manipulando o DOM](https://github.com/alura-cursos/js-metodos-array/blob/aula_1/app/main.js) através do método forEach.

## Na próxima aula:

Vamos aprofundar nosso conhecimento e aprender que podemos criar um novo array, derivando-se do array atual através de um novo método chamado map!

[00:00] Nosso próximo desafio é aplicar um desconto nos valores dos preços de cada um dos livros. Para isso, utilizaremos o método map.

[00:11] O método map() invoca a função callback passada por argumento para cada elemento do array- e aqui um ponto muito importante - e devolve um novo array como resultado.

[00:22] Vamos testar isso, cliquei na opção "Português do Brasil", na parte superior direita da tela e vou selecionar a língua inglesa, temos a mesma tradução para o map e tem um teste que podemos visualizar.

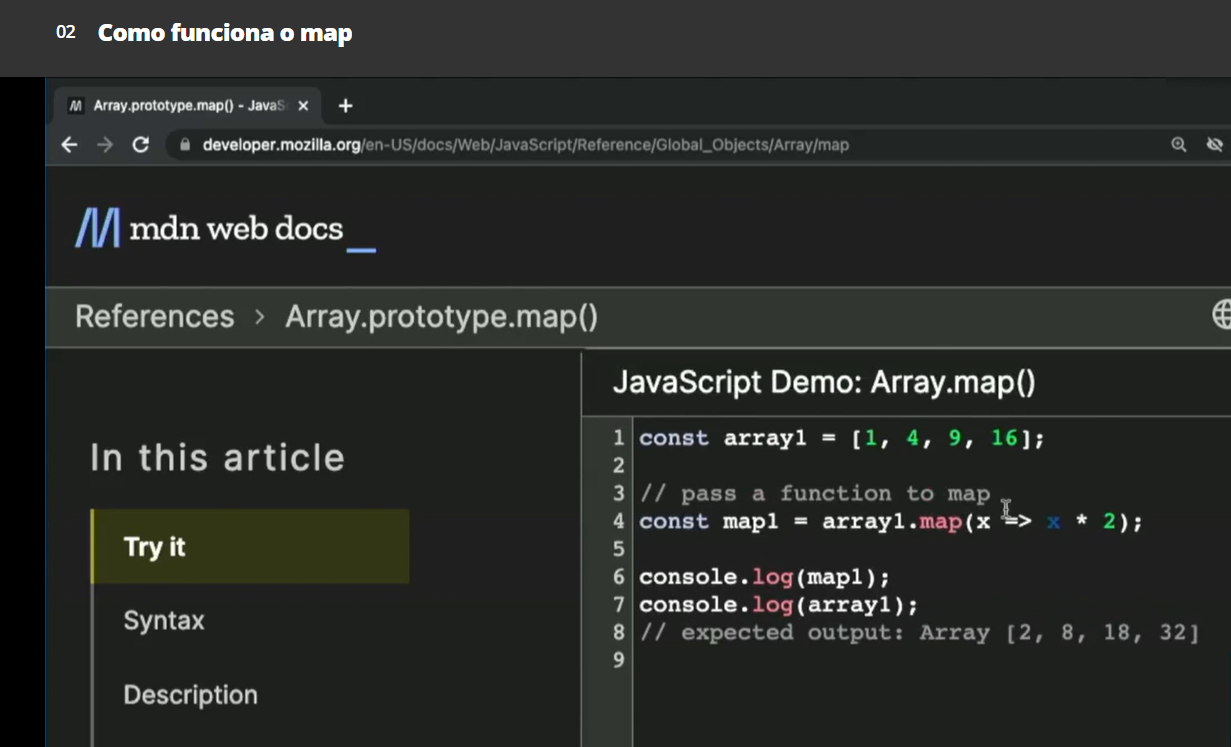
[00:32] Esse teste fala que ele tem um array e uma constante const array1 = [1, 4, 5, 16]. E ele vai passar para uma função map e ele cria uma nova constante const map1 = array1.map, pega cada valor, cada elemento do array e faz ele vezes dois.

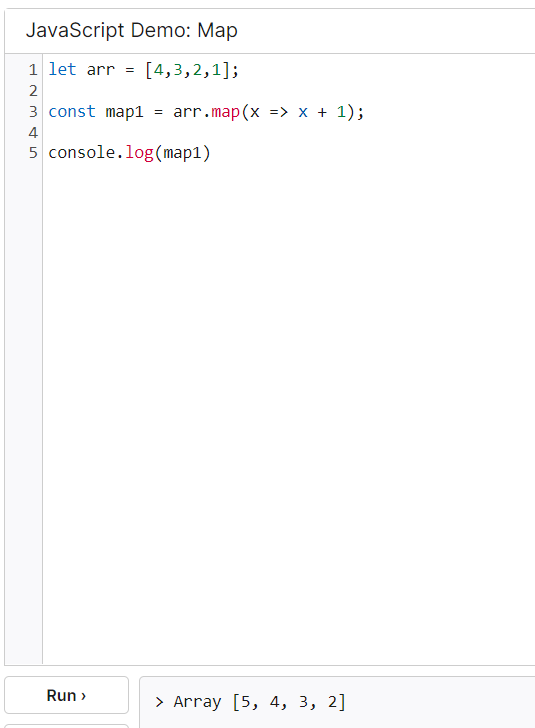
[00:54] Então, no console.log() o que é esperado é 2, 4, 18, 32, que é o dobro de cada um desses valores. Vamos rodar esse código para vermos e realmente temos isso. Só que tem um ponto importante que pode passar despercebido, que é o seguinte: esse método, assim como o método forEach, ele não altera o valor da array original, ele faz uma cópia para conseguir utilizar essas ações.

[01:18] Ou seja, ele passa por cada elemento do nosso array, que é invocado, executa uma determinada função, o comportamento que esperamos e retorna um novo array contendo os novos valores retornados.

[01:30] Essa função map é invocada da mesma forma que a função forEach. Só que, a função map, diferente do forEach, ela retorna um valor, geralmente. Se eu colocarmos console.log(array1), por exemplo, teremos os mesmos valores 1, 4, 9, 16 embaixo. Então o map vai alterar os valores para a nova array que utilizaremos, mas ele não altera a array principal. E diferente do forEach, teremos um retorno para essa nossa função.

[02:17] No próximo vídeo, quero começar a botar a mão na massa no nosso projeto, para alterarmos cada um desses valores embaixo.





[00:00] Vamos organizar um pouco melhor o nosso código. Reparem que criamos a nossa função main, que faz a busca de livro e já colocamos a função com o forEach embaixo.

[00:15] Vamos a função forEach para ficar mais fácil conseguirmos identificar o nosso código. Eu vou criar um novo arquivo JavaScript dentro de app chamado "metodoForEach.js".

[00:33] E dentro desse metodoForEach, vamos pegar a constante do elemento que buscaremos e a função, da linha 12 até a linha 29. Dar um "Ctrl + X" para recortar e "Ctrl + V" para colar.

[00:48] Dessa forma, você vai ter em cada um dos arquivos, métodos específicos para você conseguir pesquisar depois essas informações. Então temos o metodoForEach, vou criar já um novo arquivo chamado "metodoMap.js". E em "index.html" importaremos esses dois.

[01:10] Temos o nosso arquivo main, teremos um novo script que vai ser o app/metodoForEach.js. Depois vai ter mais um script que vai ser o app/metodoMap.js. E vamos começar a mapear os nossos preços, aplicando um desconto.

[01:36] Como que faremos isso? Na nossa função que exibe os livros, lá no main, exibirOsLivrosNaTela, eu quero colocar uma outra informação. Eu quero aplicar esses descontos e exibir, de fato, os livros na tela já com o desconto. Vamos supor que isso faça muito sentido para o nosso negócio, para o que estamos criando.

[01:56] Vamos criar uma nova variável aqui dentro, eu vou tirar esse console.log também, não precisamos mais dele. No lugar desse console.log, depois que pegamos a resposta com o json, eu vou fazer o seguinte: vou criar uma nova variável que eu vou chamar de livrosComDesconto vai ser = uma função que criaremos agora aplicarDesconto. E passaremos para essa função todos os livros.

[02:33] Depois, exibiremos na tela todos os livros já com os descontos. Essa vai ser a nossa sacada. Vamos criar uma função que criará o desconto em todos os livros em metodoMap. Então function aplicarDesconto(livros).

[02:59] Em seguida, vamos definir o quanto de desconto queremos, eu vou colocar const desconto = 0.3 por exemplo, um valor. E agora passaremos esses livros com o desconto para cá.

[03:17] Então eu vou fazer o seguinte, vou criar a variável, vou usar livrosComDesconto = livros.map e vamos aplicar desconto em cada livro. Então, nossa arrowFunction, {} e vamos mandar ver nessa função. Primeira coisa, a função map vai precisar retornar alguma coisa, então eu vou colocar um return.

[03:50] Queremos pegar todos com todas as informações sem alterar nenhuma delas além do preço. Para isso, podemos usar no return e duas chaves {} para indicar que retornaremos um objeto. Vamos utilizar um operador especial do JavaScript que é ...livro. Quando colocamos as reticências ... e escrevoemos livro, ele vai fazer uma cópia de todo o nosso array, ou do nosso objeto existente, para outro objeto.

[04:27] Então faça uma cópia de tudo o que você tem de livro, mas eu vou precisar alterar o preço dele. Então faça uma cópia com esse *spread operator*, *operator* com JavaScript e eu vou alterar o preço dele.

[04:41] Para isso, vamos dizer que o livro.preco vai ser igual a livro.preco \* desconto. E conseguiremos ver melhor. Está tudo em uma linha, eu só quebrei para visualizarmos corretamente, vou até tirar o *zoom* para vocês verem, o meu código está assim.

[05:04] Vou aumentar o *zoom* para vermos bem. Então pega tudo o que eu tenho de livro, faz uma cópia desse livro. Só que o preço eu vou fazer o livro.preco -.

[05:21] Em sequência, fazemos o return livrosComDesconto. Em nosso método main, pegamos os livros, depois aplicamos o desconto nesses livros e ele vai retornar o objeto.

[05:45] Pega esse objeto e guarda dentro dessa nossa variável de livros com desconto e exibe os livros com desconto na tela. Vamos atualizar e temos embaixo os livros com um desconto bizarro, está com muitos números decimais, precisaremos fixar esses valores.

[06:01] Podemos fazer isso ou no nosso código main, ou lá no nosso método forEach, podemos escolher. Eu posso colocar, por exemplo, no livro.preco um . toFixed(2) só para ele fixar em duas casas decimais, isso já resolveria o problema.

[06:25] Isso já resolveria a parte visual e eu acho que é o menos custoso para nós. Então, para arrumarmos aquela quantidade de valores periódicos da nossa dízima, eu vou fixar na exibição que ele vai estar em duas casas decimais.

## Nesta aula:

* Aprendemos que o método map executa uma função para cada elemento do array e devolve um novo array como resultado;
* Aplicamos um desconto através do método map nos [preços de cada livro do nosso projeto](https://github.com/alura-cursos/js-metodos-array/tree/aula_2/app) e entendemos a diferença entre os métodos forEach e map.

## Na próxima aula:

Vamos aprender um novo método chamado filter para exibir os livros por categoria!

[00:00] O nosso projeto está muito legal, está incrível. Aplicamos desconto nos livros, isso ficou incrível, está muito bonito! Porém, temos alguns botões que não demos tanta ênfase até o momento, mas começaremos a trabalhar nisso agora.

[00:13] Temos livros e determinadas categorias, como *front-end*, dados, livros de *back-end* como esse de Python. Eu quero clicar em um desses botões, "Livros de front-end", quando eu clicar, apenas os livros de *front* vão ser exibidos. Alteraremos esse conteúdo.

[00:35] Cliquei em "Livros de back-end" ou em "Livros de dados", eu quero visualizar só por determinadas categorias. E para esses três botões, nós utilizaremos um método especial chamado filter. Vamos pesquisá-lo também, então vou colocar array prototype filter e vou clicar nesse primeiro link do Mozilla, para entendermos.

[00:59] Então ele diz assim: o método filter() cria um novo array com todos os elementos que passam no testes implementado pela função fornecida. Legal, mas não entendi muito bem. Vamos tentar entender no exemplo que está em inglês? Então inglês, ele fala a mesma coisa, de uma função fornecida e ele tem um exemplo.

[01:19] Tem algumas palavras e ele quer filtrar const words e ele tem algumas palavras. Ele quer filtrar essas palavras e armazenar essas palavras em um novo resultado, ou seja, o método filter vai gerar, para nós, um novo array, com palavras que tenham mais de seis caracteres. E ele devolve esse resultado, então vamos ver se isso é verdade.

[01:42] Exuberante, mais de seis. Ok, tudo mais de seis. Exemplo legal, mas vamos criar um exemplo nosso, que faça sentido, para entendermos bem o que é que essa função faz.

[01:52] Então o exemplo que eu quero criar com vocês é: eu vou criar uma lista const idades e eu quero saber idades que podem dirigir ou não. Então eu tenho [10, 22, 42, 16, 50], por exemplo. Tenho algumas idades.

[02:21] O que eu quero fazer? Vou criar um novo array, que o método filter vai gerar um novo array, então eu vou escrever esse array: const podeDirigir = idades.filter e aqui a gente vai passar a nossa função.

[02:42] Lembrando que se quisermos, essa função que ele fala, função callback, seria exatamente isso, uma função que não passamos o nome e passaremos aqui dentro as idades, por exemplo, a idade. E a gente abre e fecha {} aqui, algo desse tipo. Mas eu vou já passar a nossa função com base na arrowFunction.

[03:05] Então eu vou pegar uma idade e essa idade eu vou fazer alguma coisa, que vai me retornar um valor, essa comparação vai me retornar um valor. Minha arrowFunction, faltou fechar um parênteses desse. A minha arrowFunction vai ser assim: ela vai ter um return, porque eu vou gerar uma nova lista e eu vou verificar se a idade é >= 18.

[03:31] E embaixo eu vou dar um console.log nos dois, tanto na podeDirigir, como nas idade para vermos que ela não mudou. Vamos escrolar o mouse um pouco para baixo, quando eu executo, pode dirigir: 223 42 e 50; e todas as idades: 10, 22, 42, 16 e 50.

[04:01] Então repara que o método filter, a principal diferença dele, para entendermos como ele funciona, é o seguinte: temos um array, com determinados valores, sejam eles idades, categorias, alguma coisa e queremos filtrar, criando um novo conjunto de elementos, uma nova array.

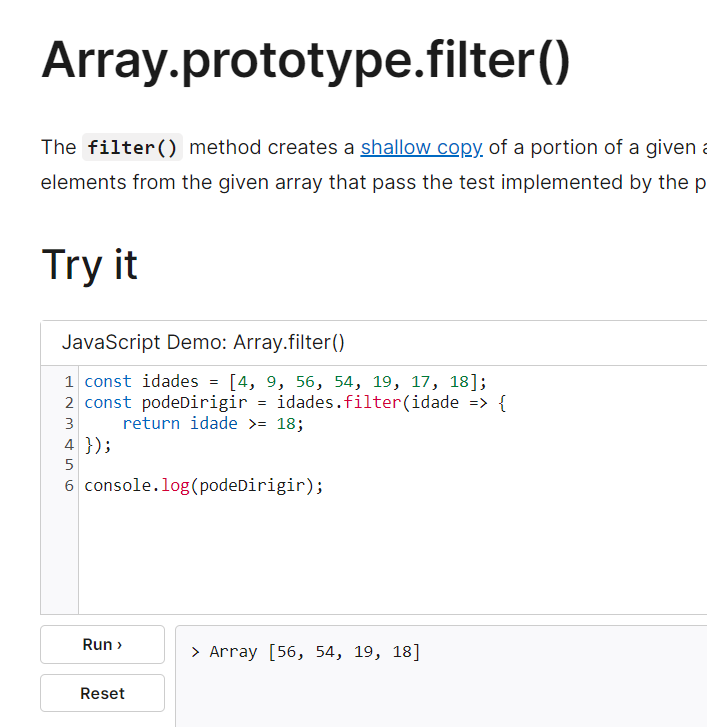
[04:21] Então essa função, sempre que passamos por ela, a função vai retornar um valor booleano e esse é um ponto interessante. Repara que eu falei idade >= 18 o valor booleano, vai ser verdadeiro ou falso, *true* ou *false*.

[04:35] Então, se o valor for verdadeiro, aí sim ele vai pegar esse valor e vai colocar nessa nossa outra array. Vou dar um *zoom* bem grande para vermos vem. Então, se o valor da idade for de fato maior ou igual a 18, ele vai armazenar na variável podeDirigir. Se não for, ele descarta e ele não utiliza aquela verificação, aquela idade.

[05:00] E o que é que faremos na sequência? Exatamente verificaremos, quando clicarmos no botão de *front-end*, se a categoria desse livro é *front-end*, se a dessa é *front-end*, se desse é *front-end*. E tudo o que for *front-end* eu vou ter um novo array, com todos aqueles elementos daquela categoria.

[05:19] E o mesmo para os outros, eu quero livros de *back-end*. Então todos os livros que são de *back-end*, que têm a categoria como *back-end*, serão utilizados lá.

[05:28] Então temos um exemplo bem lúdico, bem ilustrativo para mostrar como escrevemos esse método filter e um exemplo mais realista, que começaremos a atacar na sequência.



[00:00] Vimos como o método filter funciona, o que eu quero fazer agora é aplicar aquele conhecimento do filter, desse método de array, para esses botões. Então eu quero clicar no botão de *front-end* e visualizar só os livros de *front-end*. Vamos para o nosso código para fazermos isso acontecer.

[00:16] Primeira coisa, na nossa posta de *app*, eu vou criar um novo arquivo, que eu vou chamar de metodoFilter.js. No nosso index.html, no final dele, eu vou dizer que esse *script* faz parte do nosso projeto, então script src="app/metodoFilter.js"></script>.

[00:40] E no metodoFilter começaremos o nosso desafio. Primeira coisa, temos um botão que vai disparar essa ação. Então o que faremos é de fato pegar esse botão. Se observarmos, esse botão tem um *id*: btnFiltrarLivrosFront. Eu vou buscar através desse *id*.

[00:59] Vou criar uma variável chamada btn, pode ser uma constante, melhor, const btnFiltrarLivrosDeFront = document.getElementById('btnFiltrarLivrosFront'). Isso está em uma linha só, mesma coisa, porque está com vários caracteres e ele corta.

[01:31] Na linha dois, o que eu vou fazer? Já que eu preciso clicar e nesse clique precisa acontecer algum evento, alguma cosia, eu vou adicionar, no clique, para que uma função seja executada. Eu vou fazer isso como? Vou selecionar o btnFiltrarLivrosDeFront.addEventListener().

[01:54] Entre os parênteses, eu vou falar qual é o tipo de evento, vai ser o evento de click; quando eu clicar, eu quero de fato realizar alguma coisa. Essa alguma coisa vai ser filtrarLivros, por exemplo. Vou deixar filtrarLivrosDeFront para ficar mais claro ainda.

[02:11] E embaixo vamos começar a criar a nossa função, que vai function filtrarLivrosDeFront() e não temos nenhum parâmetro. Primeira coisa, filtraremos esses livros e esses livros serão armazenados em uma variável. Por quê? Lembra que quando filtramos, o que for verdadeiro ele vai guardar em algum lugar, vai ter um retorno esse filtro.

[02:40] Então, primeira coisa teremos uma variável que eu vou chamar de let livrosFiltrados vai ser = eu preciso de algum jeito de pegar todos os livros que eu tenho. E se observarmos, olha que interessante, não à toa, criarmos uma variável de escopo global chamada livros, que é uma array vazia, que quando executamos, faz o get dos livrosDaAPI, do json, ele coloca todos os livros dentro dessa nossa variável de livros, então eu posso usá-la aqui também.

[03:13] Eu vou falar livros.filter() eu quero filtrar todos os livros, então vai ter uma ação específica para cada um dos livros. Todo livro eu quero uma ação específica. O que eu quero fazer? Eu quero pegar todos os livros que tenham a categoria de *front-end*, que tenham a categoria igual escrito *front-end*, que é esse nome que aparece aqui, *front-end* ou dados ou *back-end*, temos três categorias.

[03:44] Eu quero que o livro.categoria == 'front-end'. E depois que ele pega esses livros filtrados, eu vou exibir uma mensagem no console, com o console.table(livrosFiltrados). Volto na nossa aplicação, atualizo a página, deixe-me inspecionar para visualizarmos isso acontecendo.

[04:15] Quando eu clico no botão "Livros de front-end", localizado na parte superior esquerda da tela", aparece "Vue", "Saas", "HTML", todos os livros de *front-end*. "Postgre" não apareceu, o "Python" não apareceu e o "Spring Boot" também não apareceu.

[04:25] Beleza Gui, mas será que isso está funcionando mesmo? Só para visualizarmos, eu vou passar como categoria dados, que é o que está escrito ali, ou seja, quando eu clicar no botão "Livros de front-end", localizado na parte superior esquerda da tela, que não é o que vai acontecer, vai acontecer agora só, mas depois eu vou voltar para *front-end*, ele vai exibir no console só os livros de dados.

[04:43] Então cliquei e ele exibiu dois livros dados: o "Postgre" e o "Pandas". Vamos ver os livros de *back-end*? back-end. Então quando eu clico no livro de back-end, aparece "Python", "Spring" e "Django". Beleza, apareceram os três. Eu vou voltar isso para *front-end* porque não faz sentido.

[05:05] E a minha pergunta é a seguinte: olha só, vimos que para mudar os livrosFiltrados de uma categoria para a outra, alteramos muito pouco, alteramos só essa categoria dentro do filtrarLivrosDeFront.

[05:20] Vou mudar, vou deixar isso só como filtrarLivros(), não vai chamar filtrarLivrosDeFront, vai se chamar filtrarLivros e eu quero que pensemos como é que aplicaremos esse mesmo comportamento para esses outros botões: para o "Livros de back-end" e para o "Livros de dados"?

[05:36] O que é que fizemos? Trouxemos um botão e aplicamos um novo evento de clique para eles, chamado de filtrarLivros. Será que a deixaremos o nosso código legal se criarmos esse filtrarLivros ou realizar um document.getElementById para todos os botões que temos? E adicionar, na mão, para todos os botões, esse evento de clique para filtrar os livros?

[06:03] Isso vai ficar um pouco ruim, o nosso código não vai ficar legal. E na próxima aula, no próximo vídeo, eu quero mostrar para vocês como faremos de uma forma que o nosso código fique bonito, fique elegante, utilizando coisas e conceitos que já aprendemos nesse curso, especificamente.

[00:00] Eu quero mostrar para vocês uma forma de pegar mais de um botão e atribuir nesse mesmo botão a mesma função para otimizarmos o nosso código e evitar a duplicidade de código. Então temos três botões principais, que são: "Livros de front-end", "Livros de back-end" e "Livros de dados".

[00:17] Se formos em nosso código, observe que temos uma li para cada livro e tem um botão com uma classe. Essa classe é btn btn-back, classe btn btn-dados, btn btn-disponiveis e btn btn-ordenacao.

[00:30] O que é que eu posso fazer? Existe uma forma de eu falar assim: olha, vá lá no documento e pegue todos os botões que eu tenho nesse meu documento. Vou mostrar isso para vocês acontecendo no console, ok?

[00:43] Deixe-me minimizar, porque o console está gigante, nem eu estou conseguindo enxergar. Agora sim, no console. Então o que é que eu quero fazer? Vá lá no document.querySelectorAll, não apenas querySelector.

[00:57] Se eu colocar apenas querySelector, eu vou selecionar, quando ele encontrar o primeiro elemento, ele já vai pegar. Não é isso que eu quero. Eu quero que ele pegue todos os botões da minha tela. Então eu vou falar assim: pegue lá tudo o que for .btn. Vou fechar a aspa simples e fechar o parênteses.

[01:15] Repare que ele me devolveu uma lista. Então, quando eu dei o "Enter", ele devolveu btnFiltrarLivrosFront, btnFiltrarLivrosBack, btnFiltrarLivrosDados, btnLivrosDisponiveis e btnOrdenarPorPreco. Isso ficou incrível, é exatamente isso que eu quero.

[01:30] Então vamos lá no nosso código e ver como que podemos utilizar essa nossa função. Aqui temos o código anterior, que funcionava para essa função que faz o filtro dos livros. Eu vou remover essa função que faz esse filtro.

[01:43] E vou criar uma nova variável, vou chamar de const botoes. E essa const botoes vai ser do tipo document.querySelectorAll. Não é querySelector. Pessoal, não é querySelector só. É querySelectorAll.

[02:00] De todos, não quero só o primeiro elemento que ele encontrar, eu quero todos os elementos, entre aspas, que tenham o .btn. E guarda nessa nossa variável botoes. Beleza Gui, eu tenho minha lista de botões e eu guardei toda essa lista aqui dentro. O que é que eu vou precisar fazer?

[02:16] Eu vou precisar pegar o primeiro botão e falar: olha, quando tiver um evento de clique, você vai chamar filtrarLivros. Pegar o segundo botão e falar: olha, quando você tiver um evento de clique, você vai chamar a função filtrarLivros. O terceiro botão e assim por diante.

[02:30] Essa questão de pega o primeiro elemento, faz alguma coisa; pega o segundo elemento, faz alguma coisa; pega o terceiro elemento, faz alguma coisa, executa uma função, isso me lembra algo que já vimos, que se chama forEach.

[02:41] Olha que interessante, então eu vou usar uma solução que já aprendemos no curso, que é botoes.forEach. E o que é que eu quero fazer com cada botão? Então eu vou falar assim: cada botão eu quero realizar uma determinada ação. Que ação é essa?

[02:57] A ação vai ser btn.addEventListener que vai ser o clique que vai chamar a função de filtrar livros. Olha que incrível isso, ficou muito legal. Olha só, então com duas linhas, praticamente, conseguimos resolver o problema de pegar todos os botões e colocar a função de filtrar livros.

[03:18] Só que agora temos um novo problema. Qual que é o problema? O problema é que qualquer um dos botões vai filtrar para livros de *front-end* e não é isso que queremos. Olha só: vou atualizar a página e vou clicar no botão "Livros de front-end", no canto superior esquerdo da tela, vou abrir o inspetor.

[03:34] Vamos no console e ele deu um Array de sete posições. Deixe-me minimizar só para conseguirmos ver melhor. Eu vou clicar no botão "Livros de dados", na parte superior da tela, ele me deu um Array de sete. Vou colocar outro livros, está me dando a tabela.

[03:50] Qualquer um dos botões que eu clico, ele está me devolvendo essa mesma Array. Se eu colocar, por exemplo, dados, ele vai me devolver uma lista com dois elementos, que são dois livros que eu tenho.

[04:00] Então eu vou clicar no botão "Livros disponíveis", na parte superior da tela, e ele vai me devolver uma lista com dois. Se eu clicar no botão "Ordenar por preço", no canto superior direito da tela, lista com dois.

[04:07] Se eu clicar no botão "Livros de front-end", no canto superior esquerdo da tela, ele vai me dar os dois livros de dados, que é o que eu tenho na minha lista. Isso não faz muito sentido. De alguma forma, precisaremos manipular esse conteúdo. Precisaremos identificar e alterar esse conteúdo.

[04:21] Como é que podemos fazer isso? Bom, uma das formas que temos é a seguinte: conseguimos saber exatamente qual é o botão que estamos clicando, para saber qual é o id desse nosso botão.

[04:32] Então, observando na index, temos o id="btnFiltrarLivrosFront”, o id="btnFiltrarLvirosBack", id="btnFiltrarLivrosDados" e assim por diante, ok? Então o que é que eu poderia fazer?

[04:42] Se eu tenho a informação de um botão específico, que eu vou pegar, deixe-me pegar os botões: botoes no índice zero, no índice um. Não, no índice zero, que é o button0btnFiltrarLivrosFront.btn.btn-front. Vou falar btn é igual a isso.

[05:00] Então se eu digito btn, ele vai me mostrar button0btnFiltrarLivrosFront.btn.btn-front. Se eu tenho esse botão, eu consigo saber exatamente qual é o valor desse botão. Então se eu colocar um btn.value ele me devolve exatamente a informação *front-end*.

[05:15] Se eu for no meu índice de botões e pegar o valor, o índice um, por exemplo, que é o livro de *back-end*, olha que interessante: se eu colocar botoes[1].value, olha o que ele me devolve: *back-end*.

[05:36] Isso já me dá uma ideia. E se eu usar o .value para conseguir identificar exatamente qual é essa categoria que eu estou querendo encontrar?

[05:45] Então eu vou fazer o seguinte: eu vou criar uma const, que eu vou chamar de const elementoBtn = document.getElementById exatamente do botão que estamos trabalhando. Então eu vou chamar aqui (this.Id), que vai ser para eu conseguir saber qual é o botão que eu estou clicando, ok?

[06:14] Dessa forma eu falo assim: houve um clique, houve um evento de clique, chame essa função. Primeira coisa, primeira instrução que essa função vai fazer? Qual é o botão que está sendo clicado? Puxa, se eu sei o botão que está sendo clicado, eu consigo pegar o valor dele, a categoria dele.

[06:28] Vou criar uma nova variável, const categoria = elementoBtn.value. E o que eu posso fazer é colocar a categoria. Falar, olha o livro filtrado vai ter que ser igual a categoria que eu estou passando. E aqui eu faço um - nossa, eu escrevi errado, deixe-me arrumar isso. Que coisa, em? - console.table.

[06:58] Aqui eu fecho, eu não escrevi errado não, eu esqueci o parênteses, parecia que estava errado. É isso aí. Então o que é que eu fiz? Pega o id do botão que estamos, pegamos o elemento todo.

[07:08] Peguei o elemento? Eu vou pegar o valor que está contido em cada um desses elementos, que temos "front-end", "back-end" e "dados", por exemplo e você vai buscar, vai fazer um filtro com base nessa categoria.

[07:20] Vamos lembrar, se eu faço um filtro de uma lista vazia, eu não vou ter nenhum retorno, não vai ter nenhuma informação. Então o que é que acontece? Com esse código, se eu clicar no console, eu vou conseguir ver cada um desses livros.

[07:32] Então olha: "Livros de front-end", apareceram os livros de *front-end*; "Livros de dados", apareceram só os livros de dados, olha "Post", "CRISP" e o "Pandas"; "Livros de back-end", apareceram três, "Python", o "Springboot" e o livro de "Django.

[07:46] Então temos o comportamento que queríamos. E mesclamos duas informações: com o método forEach, para conseguir colocar a função específica de filtrar livros; e o método filter, para de fato realizarmos o filtro.

[07:59] Se eu clico, por exemplo, no botão "Livros disponíveis", na parte superior da tela, ou no botão "Ordenar por preço", no canto superior direito da tela, nada vai acontecer. Não receberemos nenhum erro e nada acontece. Olha, "Livros disponíveis" e "Ordenar por preço", não acontece absolutamente nada.

[08:10] Esses três estão exatamente como a queríamos. Nosso próximo desafio é mostrar esses livros que estão no console, na tela.

[00:00] Quando clicamos em um dos botões, por exemplo, "Livros de dados", aparece no console todos os livros. Mas não é esse o comportamento que eu quero. Eu quero que, os livros que eu clique nesses botões apareçam na tela, ou seja, que os elementos sejam exibidos na tela.

[00:14] Para isso, o que é que faremos? No método forEach, se você navegar até ele, olha que interessante: temos uma função chamada exibirOsLivrosNaTela. Vamos usar essa função e ver o que é que vai acontecer com o nosso código?

[00:26] Então, no lugar de usarmos um console.table na linha oito do nosso método filter, eu colocaremos a função exibirOsLivrosNaTela(livrosFiltrados). Vamos ver o que é que vai acontecer? Eu vou fechar o meu console, para focarmos só no nosso código agora.

[00:42] E eu vou clicar no botão "Livros de dados", na parte superior central da tela. "Livros de dados", cliquei. Parece que não aconteceu nada de diferente, mas quando escrolamos o mouse, olha que interessante: os livros de dados já estavam aparecendo, que são o PostgreSQL e o livro de Python Pandas, Pandas Python, esses dois.

[00:59] E eles aparecem embaixo. Vou fazer o seguinte, vou clicar de novo no botão "Livros de Dados", na parte superior central da tela. Olha que louco, ele está colocando esses elementos, conforme eu vou clicando, embaixo. Então, nós temos todos os elementos, conforme clicamos nos botões para filtrar os novos livros, eles estão aparecendo embaixo.

[01:15] Não é esse comportamento que eu quero. Se eu clicar, por exemplo, no botão "Livros de front-end", na parte superior esquerda da tela, olha o tamanho que vai ficar agora a estrutura. Eu vou clicar agora no botão "Livros de Dados", na parte superior central da tela, ele está repetindo os livros. Não é isso que eu quero.

[01:25] Vamos lá, o que é que podemos fazer? Observando o nosso método forEach, temos esse elementoParaInserirLivros. E o que é que faremos? Assim que ele é clicado, que essa função é executada, ele pega a lista que ele recebeu, a lista de livros, não importa se são livros filtrados, se são livros da requisição da API, ele vai pegar e para cada livro ele vai inserir dentro desse elemento.

[01:46] O que poderíamos fazer é de fato pegar esse elemento, elementoParaInserirLivros, acessar o conteúdo dele .innerHTML e falar que o conteúdo dele vai ser sempre uma *string* vazia, ele vai ser limpo. Ou seja, sempre que clicarmos, ele vai limpar e vai mostrar apenas os livros que estamos visualizando.

[02:06] Então teremos livros de *front-end*, aparece só os livros de *front-end*. Podemos até validar isso com as categorias, olha só, todas as categorias de *front-end*. Livros de *back*, agora sim, *back-end*, *back-end* e *back-end*. E livros de dados, aparece aqui embaixo, dados e dados.

[02:22] Dessa forma, o nosso código ficou muito mais interessante e estamos reutilizando códigos que nós já tínhamos feito. Vimos também que podemos cruzar informações entre esses métodos de *array*, por exemplo, o forEach e o filter também, que utilizamos os dois para realizar o filtro dos botões e isso ficou incrível.

[00:00] O nosso próximo desafio vai ser ordenar os livros por preço. Então, no botão "Ordenar por preço", na área superior direita da tela, eu vou clicar nele e eu quero ordenar esses livros, por preço. Quando eu clico, ele some. Os erros porque estamos chamando aquela função de filtrosLivros, mas não tem nenhuma ordenação ali.

[00:16] O que é que faremos? Vamos pesquisar o método de *array* responsável por essa tarefa de ordenação. Então eu vou colocar *array prototype sort*. *Sort*, em inglês, significa ordenar. Vou clicar aqui, ele está mostrando essa página está em português.

[00:32] Vamos ler essa em português, depois passaremos em inglês, para realizarmos alguns testes. Então vamos lá. O array.prototype.sort(), o método *sort* ordena os elementos do próprio *array* e retorna o *array*, retorna o *array* ordenado. A ordenação não é necessariamente estável.

[00:46] Quando clicamos nesse estável, vamos para um link do Wikipedia, falando o que é uma ordenação estável. Preserva a ordem dos registros de chaves iguais, então ele dá até um exemplo e fala enquanto a ordenação não estável está sujeita a ter um comportamento diferente.

[01:02] Em outras palavras, não vai ser uma ordenação que esperamos necessariamente, entenderemos o porquê. Ele fala também sobre o tempo de execução, depende da quantidade de memória que utilizaremos para realizar essa operação, também.

[01:18] E aqui tem um ponto muito interessante. Quando passamos uma *array* e fala para ela ser ordenada, conseguimos passar uma função de comparação para ela. Essa função vai definir exatamente a ordem dos elementos.

[01:30] E por que é que ela não é uma ordenação estável? Olha que interessante: o *array* é ordenado de acordo com a pontuação de código *unicode* de cada um dos caracteres, de acordo com a conversão de cada elemento para string.

[01:45] O que é que vai acontecer? Quando chamarmos essa função, ela vai transformar os nossos elementos em string, para depois - com base no código *unicode* - sim realizar essa ordenação. E tem um ponto interessante, ele até fala: note que o *array* é ordenado de acordo com a pontuação de código *unicode*.

[02:03] Então aqui embaixo, se escrolarmos, encontramos alguns exemplos interessantes. Ele tem uma sequência de frutas e ele faz a ordenação. Então tem "cherries", "apples" e "bananas". E ele fala para ordenar, aparece "apples", "bananas" e "cherries".

[02:15] Depois - esse é o ponto chave - var scores = [1, 10, 2, 21]. Se fôssemos ordenar isso, sem olhar o código, seria 1, 2, 10 e 21. Só que, quando ele pede para ordenar, olha o resultado que é: 1; não é o 2, é o 10; 2 e 21. Observe que o 10 vem antes do 2, porque o 10 vem antes do 2 em pontuação de código *unicode*.

[02:40] Certo, então teremos um super desafio. Por quê? No nosso código - eu vou voltar em todos, atualizando a página - percebe que temos vários valores? Como que ordenaremos esses valores, de forma que faça sentido, para os elementos virem ordenados de uma maneira estável?

[02:59] Precisaremos passar uma função de comparação para ela. Então eu quero realizar um teste com vocês. Alterei, cliquei na língua e selecionei inglês americano e tem alguns exemplos que podemos rodar. Então vamos lá.

[03:13] Ele ordenou alguns meses, ordenou alguns valores também. Lembra aqui 1 e apareceu - deixe-me ver que número é esse, 1, 2, 3, 1, 2, 3 - 100 mil, 21, 30, 4. Não faz muito sentido, não é isso que queremos.

[03:28] Vamos fazer então uma ordenação para nós, assim, um exemplo prático desse daqui. Então vamos pensar. Se a ordenação vai precisar de uma função de comparação para colocar exatamente os itens em uma ordenação estável, precisaremos realizar um teste. E o que é que eu quero fazer? Vamos realizar o seguinte teste.

[03:47] Então, eu vou ter uma variável chamada precos, eu vou passar alguns valores, então vai ser [29.98,1,101,110]. Eu vou pedir para serem ordenados esses valores, então eu vou criar uma nova variável, que eu vou chamar de precosOrdenados = precos.sort() para executarmos essa nossa função.

[04:26] Eu quero visualizar os dois, eu quero visualizar, eu quero um console.log(precos) e um console.log(precosOrdenados). Vamos rodar esse código, para vermos o que é que aconteceu. Rodamos e temos os preços e os preços ordenados, não está fazendo muito sentido.

[04:50] Estamos, deixe-me tirar, deixe-me deixar só o precoOrdenado, acho que vai ficar mais fácil, conseguimos visualizar ali em cima. Então vou voltar, vou rodar mais uma vez, olha o preço ordenado: [1, 101, 11, 110, 29.98], não está fazendo sentido nenhum isso.

[05:03] De alguma forma pediremos para ordenar nos valores que esperamos. O que é que esperamos dessa ordenação? Vou colocar aqui, esperamos que seja o 1 o primeiro elemento; depois esperamos que seja o valor 11; depois o valor 29.98; depois o valor 101 e depois o valor 110. É isso que esperamos.

[05:29] Como é que faremos isso? Passaremos uma função de comparação. Então function() {} e escreveremos a nossa função. Nessa função, passaremos dois valores: o valor de a e b. O que é que vai ser o valor de a e b? Vai ser o primeiro elemento e o segundo elemento. E ele vai fazer essa comparação.

[05:49] O que eu quero que ele faça? Eu quero que ele retorne na ordem estável, então eu vou colocar um return e vou falar que ele vai ser na ordem estável do menor para o maior a - b. E vou rodar isso mais uma vez, para visualizarmos se está tudo certo. Deixe-me ver se o código está certo.

[06:06] Parece que está certo, se não estiver certo ele vai nos alertar, então ele voltou: [1, 11, 29.98, 101, 110]. É exatamente o que precisamos. Gui, eu preciso de uma ordenação ao contrário, eu preciso do menor para o maior.

[06:24] E só inverter: eu vou fazer b - a. Vou ordenar, vou pedir para executar mais uma vez e temos uma ordenação contrária: 110, 101, 29.98, 11, 1]. E aqui tem um ponto interessante, que eu quero mostrar para vocês também, que é uma curiosidade.

[06:41] Sempre que ordenarmos e tiver algum elemento que é *undefined*, ele vai ser classificado no final da nossa matriz. Então eu vou colocar, no lugar do 1, eu vou colocar *undefined*. *Undefined*, escrevi certo.

[06:53] Então eu vou rodar isso, repare o que aconteceu: o 1 estava lá no meio. Se temos um elemento que é *undefined*, ele vai lá para o final da nossa matriz. Qual que é a sacada então, Gui? Sempre que pensarmos na ordenação e precisarmos de uma ordenação estável, nós vamos precisar de uma função.

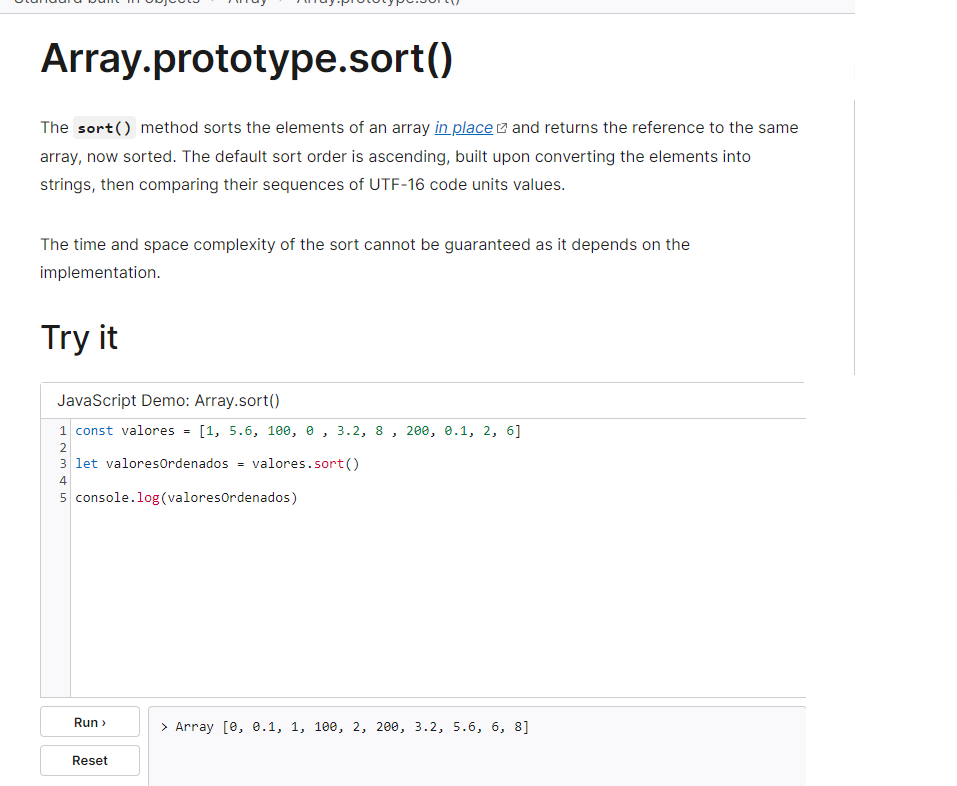
[07:08] Puxa, mas eu tenho que escrever dessa forma? Não, podemos usar uma *arrow function* também. Então eu vou tirar isso, eu vou passar dessa forma para ele. Olha, se eu rodar esse código, a teremos o mesmo resultado. E podemos deixar até sem o return, porque estamos fazendo uma ação, apenas.

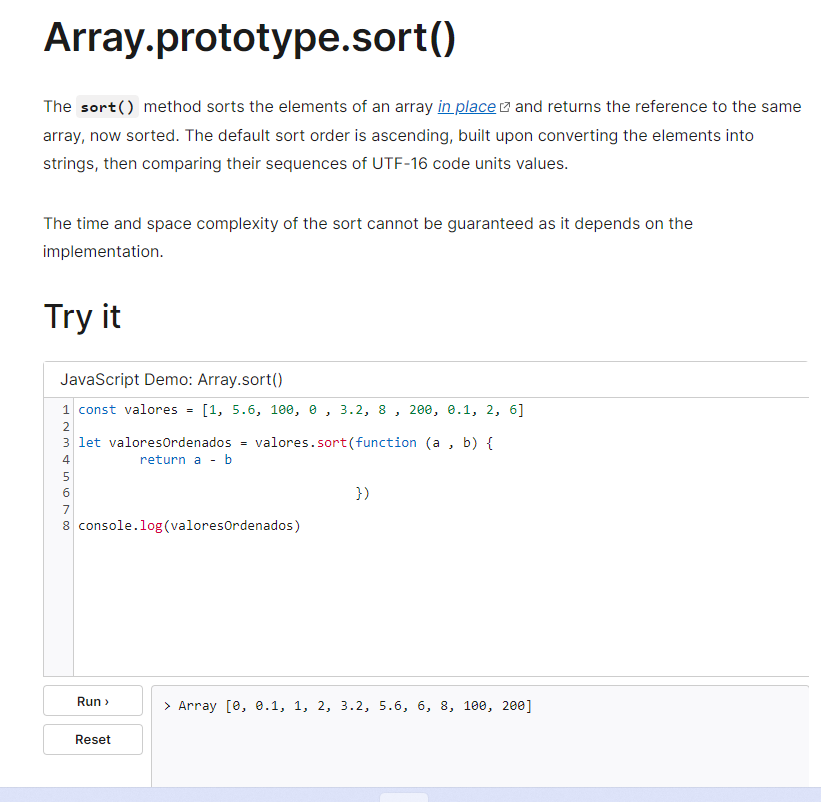
[07:22] Então eu vou tirar essas chaves e deixar o nosso código assim. Se eu rodar, também teremos o mesmo resultado. Então lembra: o *sort* vai classificar os elementos do *array* e retornar um *array* ordenado. Quando chamarmos esse *sort* com elementos que sejam *undefined*, esse *undefined* vai sempre para o final da nossa lista e eles são ordenados com base no *unicode*.

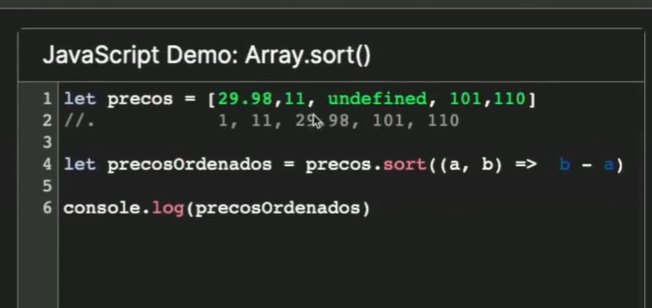
[07:44] Então tem que lembrar, ele transformou tudo para *string*, a ordenação dele não vai ser estável. Puxa, eu preciso de uma ordenação que seja estável, aí criamos uma função para fazer essa ordenação de comparação.

[07:56] Lembrando, eu preciso do menor para o maior: a - b. Então eu vou ordenar, ele vai mostrar 11, 29.98, 101, 110 e "undefined" lá no final. Eu preciso do maior para o menor, então eu coloco b - a e aí sim temos o resultado certo.

[08:12] Nosso próximo desafio é clicar nesse botão de ordenar e aplicar todo esse conhecimento do *sort* no nosso projeto.







[00:00] Vamos implementar essa ordenação por preço no nosso projeto? Primeira coisa que eu vou fazer vai ser acessar a pasta de *app* e criar um novo arquivo js, que eu vou chamar de metodoSort.js.

[00:15] Vou no index.html, nos *scripts* e vou falar: esse *script* também faz parte dessa nossa página <script src="app/metodosSort.js"></script>. No nosso metodoSort, o que é que eu vou fazer?

[00:28] Se observamos o nosso index, temos um *id* desse botão. Podemos pegar o *id* desse botão, do btnOrdenarPorPreco, ele até tem uma classe btn btn-ordenacao, mas eu vou pegar esse *id* dele.

[00:45] Eu vou criar uma variável que eu vou chamar de let btnOrdenarPorPreco = document.getElementById('btnOrdenarPorPreco). Vou adicionar nesse btnOrdenarPorPreco um evento de clique: btnOrdenarPorPreco.addEventListener('click', ordenarLivrosPorPreco) e vou criar essa função agora.

[01:32] function ordenarLIvrosPorPreco não temos parâmetros. Primeira coisa, pegaremos esses livros de algum lugar. Graças ao nosso método main, temos uma variável de livros disponível, que podemos usar.

[01:49] Eu vou chamá-la: livros.sort(). Eu vou ordenar e essa ordenação vai me devolver alguma coisa. Essa alguma coisa eu preciso armazenar em algum lugar. Então vou criar uma nova variável, que eu vou chamar de let livrosOrdenados = livros.sort().

[02:07] Só eu fazendo isso, será que já vou conseguir exibir na tela os livros com o preço correto? Não. Mas eu quero mostrar isso para vocês acontecendo. Olha só, no nosso método forEach, temos uma função chamada exibirOsLivrosNaTela.

[02:25] Vamos chamar essa função exibirOsLivrosNaTela(livrosOrdenados). Vamos ver o que é que vai ser exibido apenas com esse livros.sort, para vermos o que acontece?

[02:37] Eu estou na tela, atualizei a página, nem precisava atualizar, porque os livros já estavam aparecendo. Vou deixar o inspecionar para ficarmos de olho se receberemos alguma mensagem de erro ou alguma notificação.

[02:45] Vou colocar "Ordenar por preço" e ele mudou, ele piscou, aconteceu alguma coisa, só que os livros não estão ordenados. Repara que tem um 9,90 embaixo, não é o que precisamos.

[02:56] Lembrando: eu preciso de uma ordenação de valores que não seja no sentido alfabético, usando *unicode*, vou precisar passar uma função. Essa função eu posso passar da seguinte forma: vou falar que eu tenho o valor "a" e o valor "b", dos livros e eu quero comparar, quero fazer alguma coisa com esses valores.

[03:15] Então eu vou ter o livro "a" vai ter um preco e eu quero ordenar pelo maior preço para o maior, então vai ser "a" menos o valor do livro "b", e ele vai fazer essa comparação para todos livros.sort((a,b) => a.preco - b.preco).

[03:26] Voltando no nosso código, vou fechar o console para visualizarmos melhor. Vou clicar, repara os valores que temos: 20, 62, 55. Pedi para ordenar por preço, agora sim: 9,90, 11,90, 19, 29, 29, 49, 79, 89, 94, 139. Maravilha.

[03:45] Putz, quero ordenar no sentido oposto, então eu só inverto: livros.sort((a,b) => b.preco - a.preco). Quando eu volto no nosso código, eu peço para ordenar, ele vai mostrar o livro caro. Pessoal, isso é só fictício, esses valores não são os valores reais dos livros, tanto que se você quiser entrar na casa do código, lá estarão os valores corretos.

[04:02] Isso é meramente ilustrativo. Então vou voltar, quero ordenar por preço do menor para o maior livros.sort((a,b) => a.preco - b.preco) e exibir todos os livros na tela. Maravilha, isso ficou muito bacana.

[00:00] Nós temos um desafio muito interessante agora. Se a gente formos até embaixo da nossa página, notem como em todos os filtros que fazemos, temos esse elemento "todos os livros disponíveis por R$ 299,00".

[00:13] Então não importa a categoria que eu vou, sempre temos essa seção. Vamos olhar no nosso código onde que essa seção está? A seção no arquivo index.html, antes do footer, temos essa seção de valor\_total\_livros\_disponiveis, tem a classe livros\_\_disponiveis e tem a informação com o valor 299.

[00:34] Se eu alterar, por exemplo, para 199 ele vai alterar ali para 199 em todas as páginas. E qual que é o nosso desafio? Selecionaremos livros disponíveis e exibiremos essa informação aqui, vai mostrar essa seção disponível.

[00:49] Quando clicarmos em "Livros de front-end", "Livros de back", "Livros de dados" ou "Ordenação por preço", não exibiremos isso, manipularemos isso com JavaScript.

[00:59] Primeira coisa que faremos vai ser acessar o nosso arquivo main. Repara que temos aqui, buscamos o endpoint e depois, no forEach, temos o elemento da função que exibe, de fato, os livros na tela.

[01:14] O que é que faremos? Vamos pegar esse elemento, selecionar esse elemento dessa seção valor\_total\_livros\_disponiveis, com esse *id*. Então vou chamar const, vou chamar aqui em cima, const elementoComValorTotalDeLivrosDisponveis = document.getElementById('valor\_total\_livros\_disponiveis').

[01:50] O que é que faremos? Primeira coisa, limparemos esse elemento. Não queremos, sempre que o livro for exibido na tela, queremos acessar o conteúdo desse elemento e não exibir nada. Manipularemos isso lá quando for com os nossos livros do JavaScript.

[02:06] Não está aparecendo. Nem quando clicamos em "Livros disponíveis". O que é que faremos? Vamos no nosso metodoFilter e faremos o seguinte: temos a categoria. Podemos perguntar se a categoria desse livro for disponível, aí sim queremos mostrar aqueles livros.

[02:27] Eu posso fazer um filtro, exibe os livros na tela, desenha aqui, não importa em qual categoria e qual botão eu clique, ele vai colocar os livros. Depois, eu vou fazer uma pergunta, vou fazer assim: if (categoria == 'disponivel') o que eu quero fazer é exibir os livros, exibir essa seção na tela.

[02:54] Então vou criar uma função, que vou chamar de exibirValorTotalDosLivrosDisponiveisNaTela() e vou criar essa função agora, junto com vocês.

[03:09] function exibirValorTotalDosLivrosDisponiveisNaTela() e essa função vai fazer o que? Pegaremos aquele elemento que trouxemos, o elementoComValorTotalDeLivrosDisponiveis, vou colocar ele aqui e vou acessar o conteúdo desse elemento, .innerHTML = e eu vou fazer o seguinte, eu vou no nosso código HTML e vou copiar essa div aqui dentro dessa seção.

[03:35] "Ctrl + c" , volto no nosso metodoFilter e "Ctrl + v". Vamos ver se isso já resolveu o nosso problema. Vou atualizar a página. "Livros de front-end": não aparece nada. Na página principal, também não aparece. "Livros de back": não aparece. "Livros de dados": também não. "Ordenação por preço": também não.

[03:55] Quando clicamos em "Livros disponíveis" aí sim ele aparece. Está aparecendo, porém ele está aparecendo com um valor fixo, não está somando de fato o valor desses livros disponíveis que temos na nossa API.

[04:08] Então precisamos de alguma forma reduzir os valores, somar todos os valores dos livros disponíveis e manipular esse valor aqui dentro. Isso faremos na sequência. Antes de fazermos isso, vamos deixar o nosso código ainda mais limpo?

[04:23] Quando fazemos essa verificação na linha sete, dos livros disponíveis, fazemos assim: se a categoria for igual a disponível, faz o filtro pela quantidade. Então o que é que eu posso fazer? Vou selecionar tudo isso e ele vai dar uma opção para nós.

[04:39] Dar um "Option + z", "Ctrl + z" só para ele ver. Repara que tem uma lâmpada amarela do lado, eu vou clicar nela e vou colocar assim, selecionar essa opção: extrair para uma função global, de escopo global. E vou dar um nome, vou escrever assim: filtrarPorDisponibilidade, uma coisa assim.

[05:02] O que é que aconteceu? Ele tirou todo aquele código que estava aqui e passou para baixo, uma função que é filtrar por disponibilidade, que tem exatamente o filtro. E o outro, filtraremos por quê? Filtraremos por categoria.

[05:14] Então eu vou selecionar, voltando, selecionar todo esse livro, clico na lâmpada, extrair para uma função de escopo global e eu vou chamar de filtrarPorCategoria. Dar um "Enter" e olha que interessante, ele já extraiu para filtrar por categoria passando.

[05:32] Se voltarmos no nosso código, atualizar a página, repara que tudo está funcionando. "Livros de front"; "Livros de back"; "Livros de dados"; "Livros disponíveis" aparece lá embaixo; "Ordenação por preço" não aparece ali e está tudo uma maravilha agora.

[05:45] Gui, mas qual que é a vantagem de termos o nosso código assim? Repara que a leitura do entendimento do nosso código, a legibilidade, está muito melhor. livrosFiltrados = categoria. Se a categoria for disponível, exibe, faz o filtro por disponibilidade.

[06:01] Se não, faz o filtro por categoria, passando a categoria que você já tem como parâmetro. Fechou. Acho que essa parte de filtro ficou incrível, está legal. Clicando nos livros disponíveis, ele exibe aquela seção do valor total, só que precisaremos, de alguma forma, somar de verdade os valores dos livros que temos.

[06:22] E para isso, usaremos um método especial de array também.

[00:00] O nosso próximo desafio é cliquei no botão "Livros disponíveis", localizado na parte superior da tela, soma todos esses valores dos livros que estão disponíveis e exibe dinamicamente em baixo.

[00:09] Para isso, utilizaremos um método especial, chamado reduce e eu quero abrir a documentação dele junto com vocês. Então eu vou pesquisar no Google array prototype reduce.

[00:24] Clicando nesse primeiro link, do Mozilla, ele fala: o método reduce executa uma função reduce fornecida por você para cada elemento do array, resultando em um único valor de retorno.

[00:36] E ele tem um exemplo de como essa função reduce acontece, funciona. Eu convido você a ler o resto da documentação para você entender melhor os outros detalhes, mas eu queria botar a mão na massa mesmo, para começarmos a entender como que essa função funciona.

[00:53] E para isso, eu quero fazer um pequeno questionamento: se ela vai reduzir tudo em um único valor, qual seria a diferença da função map ou da função filter, por exemplo?

[01:05] E tem uma imagem, que eu gosto muito, que dá para entendermos bem o que cada um desses métodos fazem. Então sempre que pensarmos no map, ele vai criar um novo array, transformando cada elemento em um array individual. Ele vai criar uma nova lista, um novo array.

[01:22] Então temos uma maçã, uma laranja e uma banana, ele vai criar um novo array com esses elementos, ele vai transformar esses elementos.

[01:29] O filter vai fazer justamente o que? O filter vai remover tudo o que não faz parte daquele filtro que fizemos e vai retornar apenas aquele filtro que estamos, de fato, aplicando nele.

[01:42] E o reduce vai fazer justamente o oposto. Ele vai pegar todos os elementos do array e produzir um único valor, a exemplo do que temos aqui. Esse exemplo é muito legal.

[01:51] Na internet, se você pesquisar, por exemplo, como *map filter reduce image*, vai aparecer um monte de imagem, tem até aprendizado com meme. Vou abrir esse só para darmos uma olhada. Claro que isso é meramente ilustrativo, mas só para conseguirmos entender.

[02:06] Olha que interessante: nesse exemplo, ele está mostrando *map, filter and reduce* com emoji, a explicação com emoji. Então temos alguns itens, quando eu coloco para cozinhar ele me devolve esses valores.

[02:16] Quando eu tenho um filtro, o que é que ele vai fazer? Ele vai me retornar apenas conteúdos que são pertencentes a quem é vegano. E não importa se você come hambúrguer, batata, pipoca ou frango, quando você comer ele vai reproduzir um único valor. Nesse caso, o valor é um cocô.

[02:37] Isso é meramente ilustrativo, para vermos, tem várias outras imagens na internet também. Vamos botar a mão na massa e escrever um código para entendermos como que essa função reduce funciona na prática?

[02:47] Então voltando para o nosso cenário, eu vou criar uma variável chamada precos. Dentro de precos, eu vou passar três valores: 10, 20 e 70, por exemplo. E vou criar uma outra variável, chamada precoTotal, que vai ser igual a esse meu valor de precos.reduce e entre parênteses eu vou começar a minha função.

[03:12] Essa função vai receber, geralmente, um parâmetro que vai ser o responsável por acumular todos os valores. Se escrolarmos temos o primeiro parâmetro da função reduce que é o acc, o acumulador.

[03:23] Eu vou chamar com esse mesmo nome , acc. O segundo valor é o item de cada elemento, o valor atual de cada elemento. Então vamos lá, eu vou passar o valor atual, vou chamar de atual.

[03:37] E como essa função vai funcionar? Vou criar uma => e vou fazer o seguinte: eu quero somar todos esses valores que eu tenho dentro de precos e reduzir essa minha lista em apenas um único valor.

[03:48] Para isso eu vou fazer o seguinte: eu vou pegar o acc + atual e ele vai devolver esse valor reduzido para mim. Vou colocar um console.log só para vermos esse precoTotal. Vamos rodar esse código?

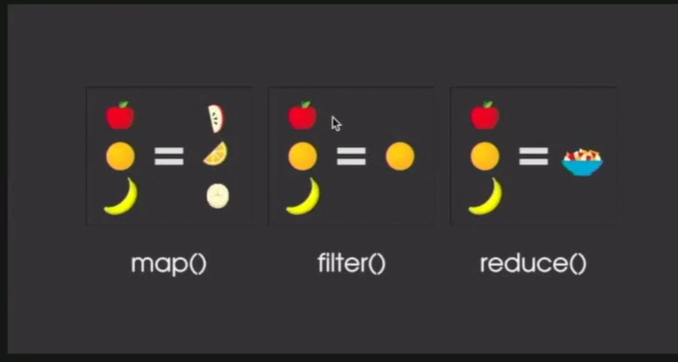
[04:10] Rodando e nos mostrou 100. Puxa Gui, legal. Eu só consigo sempre somar então? Eu uso a função reduce sempre para somar? Não, eu posso usar também para fazer uma verificação para, por exemplo, pegar qual é o maior valor dessa lista. Eu quero reduzir, pegar toda a minha lista e pegar, por exemplo, o maior valor.

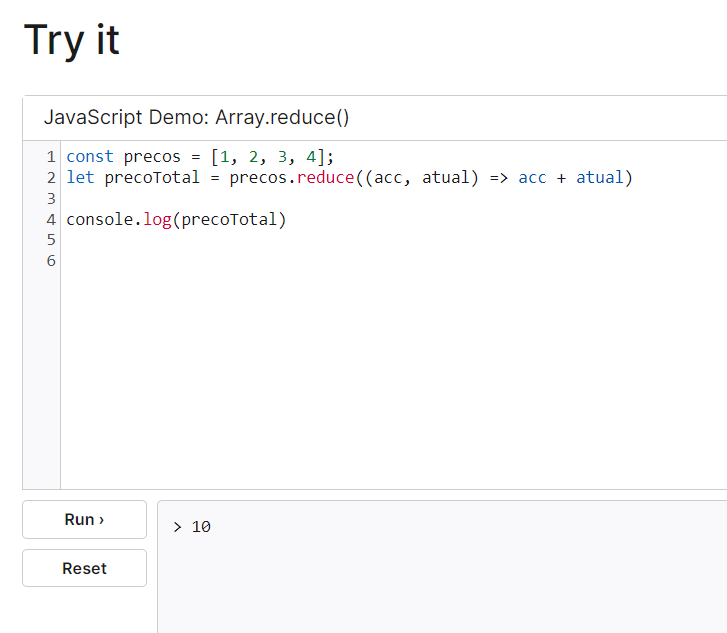
[04:26] Eu posso fazer assim: se o reduce for maior que o atual eu vou perguntar, eu quero devolver, se o acumulador foi maior que o atual, eu quero devolver o acumulador. Senão, eu quero devolver o atual.

[04:37] Olha que interessante, quando eu mandar rodar ele devolveu o maior valor. Puxa e se eu quisesse o menor valor? Eu posso colocar o sinal de < e ele vai ficar fazendo essa comparação do valor que eu estou no acumulador em relação a cada item do meu array.

[04:51] Legal. Eu estou mostrando esses exemplos com preço porque eles fazem sentido para o projeto que utilizaremos. Podemos usar o reduce também para outros elementos.

[04:59] Então a sacada, o que eu quero que você guarde no seu coração, é o seguinte: sempre usaremos a função de redução para combinar dois valores e produzir um único valor. E na sequência faremos isso aplicando esse conceito no nosso projeto.





[00:00] Vamos implementar então o método reduce para somar o valor de todos os livros quando esse botão de "Livros disponíveis" for clicado e mostrarmos esse resultado do valor total nessa seção?

[00:12] A primeira coisa que eu vou fazer vai ser, lá na pasta de *app*, criar um novo arquivo chamado metodoReduce.js. No index.html, vamos colocar esse *script* no nosso documento app/metodoReduce.js.

[00:32] O que eu vou fazer agora vai ser abrir o nosso método filter. Nós fazemos o filtro de livros disponíveis, se essa categoria for disponível, e ele exibe os livros na tela.

[00:42] Essa está feita. Quando clicamos no botão "Livros disponíveis", localizado na parte superior da tela e exibiu os livros disponíveis na tela, o que é que queremos fazer? Queremos calcular antes de colocar essa seção na nossa página.

[00:53] Essa seção é colocada aqui exibirValorTotalDosLivrosDisponveisNaTela(). Então o que é que eu quero fazer? Eu quero criar uma constante chamada de valorTotal e ela vai ser igual ao resultado de uma função, o retorno de uma função, que eu vou chamar de calcularValorTotalDeLivrosDisponiveis().

[01:16] E para eu conseguir calcular os livros disponíveis, eu vou passar a minha base de livros filtrados. Então essa função calcularValorTotalDeLivrosDisponiveis(livrosFiltrados) eu vou utilizar o método reduce.

[01:29] Então eu vou chamar a function, vou passar os valores calcularValorTotalDeLivrosDisponíveis(livros) para cá, que vai calcular o valor e aqui eu vou chamar só de livros, para ficar mais fácil o nosso entendimento.

[01:39] Essa nossa função vai ter um retorno. Então vamos lá. Qual que vai ser o retorno dela? Eu quero que pegue todos os livros e aplique a função de reduce. Para aplicar essa função de reduce, teremos um acumulador e vai ter o livro.

[01:53] Então queremos fazer alguma coisa. Eu quero que o acumulador some com o valor do livro, só que não o valor do objeto e sim o valor que eu tenho em preço.

[02:04] Some o valor que eu tenho em preço, vou colocar uma , aqui e o meu primeiro índice eu quero que seja um índice 0, que comece com o valor 0. E ele vai começar a fazer essa redução.

[02:15] Lá no nosso método filter, o que é que eu quero fazer? Antes de passarmos esse valor total para outra função, eu quero visualizar ele na tela para ver se ele está certo.

[02:23] Eu vou colocar console.log(valorTotal) só para visualizarmos. Voltando ao nosso código, vou abrir o "Inspecionar elemento", venho no "console" e vou clicar no botão "Livros disponíveis", localizado na parte superior da tela.

[02:34] Cliquei e ele mostrou 714.2 e está certo. Se somarmos, vai dar esse valor mesmo. O que eu vou fazer, só para deixá-lo um pouco melhor? Eu vou colocar um .toFixed(2) para colocar duas casas decimais nesse valor.

[02:50] Então olha só, livros disponíveis: 714.20, maravilha, já temos esse valor. Realizamos a função reduce, temos o valorTotal, vou tirar esse console.log, não precisamos mais dele, é só para testarmos mesmo.

[03:05] Eu coloquei um caractere errado. Agora sim. O que eu quero fazer agora é pegar esse valorTotal e passar ele para a nossa função exibirValorTotalDosLivrosDisponveisNaTela(valorTotal). Então passei para ela, botei o valorTotal e, no lugar do 199,00 eu vou passar quem? Com o sinal de ${}, eu vou passar o valor total.

[03:24] Olha só que interessante: carregando os livros, os livros de dados, não aparece nada. Livros de *back* também não aparece nada. Se eu clico no botão "Livros disponíveis", localizado na parte superior da tela, aí sim vem o valor total dos livros.

[03:37] Gui, se mudar os valores da API, por exemplo, esse livro agora custa 139, agora custa 9,90? Ele vai alterar sozinho esse valor total.

[03:46] Essa é uma ideia, se eu quero comprar todos os livros que estão disponíveis, esse é o valor do investimento que eu faria. E assim concluímos essa parte do reduce com o JavaScript`.

# JavaScript

### Método .concat( )

[ 🏀 , 🏀 , 🏀 ].concat( [ ⚾ , ⚾ ] ) => [🏀 , 🏀 , 🏀, ⚾ , ⚾ ]

O método concat junta dois (ou mais) arrays em um novo array, sem alterar os já existentes.

### Método .pop( )

[ 🏀 , 🏀 , 🏀, 🏀 , ⚽ ].pop( ) => [🏀 , 🏀 , 🏀, 🏀]

O método pop remove o último elemento de um array.

### Método .push( )

[🏀 , 🏀 , 🏀].push(🏈) => [ 🏀 , 🏀 , 🏀, 🏈 ]

O método push adiciona um novo elemento no final do array, aumentando seu tamanho.

### Método .includes()

[ ⚾ , 🏈, ⚽, 🏀 ].includes( ⚽ ) => true

O método includes verifica se um elemento está contido em um array e retorna true se o elemento estiver contido ou false caso contrário.

### Método .fill( )

[ ⚾ , 🏈, ⚽, 🏀 ].fill( ⚽, 1 ) => [ ⚾ , ⚽, ⚽, 🏀 ] [ ⚾ , 🏈, ⚽, 🏀 ].fill( ⚽ ) => [ ⚽, ⚽, ⚽, ⚽ ]

O método fill preenche os elementos especificados em um array com um determinado valor.

## Método .indexOf( )

[ ⚾ , 🏈, ⚽, 🏀, ⚽ ].indexOf( ⚽ ) => 2

O método indexOf retorna o primeiro índice encontrado de um valor especificado. Se o valor não for encontrado o método retorna -1.

### Método .reverse( )

[ ⚾ ,🏈, ⚽, 🏀 ].reverse( ) => [🏀, ⚽,🏈, ⚾]

O método reverse inverte a ordem dos elementos de um array e substitui o array original.

### Método .slice( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].slice( 1, 3 ) => [ 🏈, ⚽ ]

O método slice retorna elementos de um array, selecionados de determinada posição de início até determinada posição final. O elemento na posição final não é incluso.

### Método .some( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].some( ( 🏐 ) ⇒ { return bola === 🏐 } ) => true

O método some verifica se algum elemento do array passa em um teste. Esse teste é feito através de uma função callback. O método executa a função de callback para cada elemento uma vez e retorna true se o teste for true para um dos elementos, e false se o teste for false para todos os elementos. Além disso, o método não executa a função callback para arrays vazios e não altera o array.

### Método .join( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].join() ⇒ ⚾,🏈,⚽,🏀,🏐 [ ⚾ , 🏈, ⚽, 🏀, 🏐 ].join(‘ ’) ⇒ ⚾🏈⚽🏀🏐 [ ⚾ , 🏈, ⚽, 🏀, 🏐 ].join(‘-’) ⇒ ⚾-🏈-⚽-🏀-🏐

O método join puxa elementos de um array e lista no formato de string, o resultado da operação puxou as propriedades do array e as listou de acordo com o que foi determinado.

### Método .shift( )

[⚽, 🏐 , 🏐 , 🏐, 🏐].shift( ) ⇒[🏐 , 🏐 , 🏐, 🏐]

O método shift é parecido com o método .pop() mas ao invés de remover o último elemento do array, ele é usado para remover o primeiro elemento do array.

### Método .unshift( )

[🏀 , 🏀 , 🏀].unshift(🏐) ⇒ [ 🏐, 🏀 , 🏀 , 🏀]

O método unshift é parecido com o que método .push() realiza, mas ao invés de adicionar no final do array, ele é utilizado para adicionar um elemento no início de um array.

### Método .splice( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].splice( 1, 2, 🎱 ) ⇒ [⚾ ,🎱, 🏀, 🏐 ] [ ⚾ , 🏈, ⚽, 🏀, 🏐 ].splice( 2, 3) ⇒ [⚾, 🏈 ]

Com o método splice conseguimos escolher um índice inicial e final para substituirmos valores no lugar deles. E também podemos remover itens, no segundo exemplo, foram removidos três elementos a partir da posição dois.

### Método .length( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].length(3) ⇒ ⚾ , 🏈, ⚽ [ ⚾ , 🏈, ⚽, 🏀, 🏐 ].length() ⇒ 5

O método length define ou retorna o número de elementos em um array.

### Método .sort( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].sort() ⇒ ⚽,⚾,🏀,🏈,🏐

O método sort ordena os elementos do próprio array e retorna o array. A ordenação padrão é de acordo com a pontuação de código unicode.

### Método .toString( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].toString ⇒ ⚾,🏈,⚽,🏀,🏐

O método toString retorna uma string com todos os valores do array separados por vírgulas.

### Método .findIndex( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].findIndex(emoji => emoji === “⚽”) ⇒ 2

O método findIndex retorna o índice no array do primeiro elemento que satisfizer a função de teste provida. Caso contrário, retorna -1, indicando que nenhum elemento passou no teste.

### Método .find( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].find( bola ⇒ bola.formato === “redonda” ) => ⚾

O método find é utilizado para procurar um elemento dentro do array que atenda a condição atribuída a ele, que retornará o primeiro elemento encontrado. Ele percorre todo o array buscando o elemento que atenda a condição e retorna o primeiro que foi encontrado, caso não encontre retorna undefined. Este método não executa caso o array esteja vazio e não altera o array original

### Método .at( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].at(-1) ⇒ 🏐 / [ ⚾ , 🏈, ⚽, 🏀, 🏐 ].at(1) ⇒ 🏈

O at acessar os elementos de um array usando um índice inteiro positivo e negativo, sendo que o índice negativo enumera os itens de trás para frente, portando o 🏐 está posicionado no índice -1.

### Método .isArray( )

var bolas = [ ⚾ , 🏈, ⚽, 🏀, 🏐 ] Array.isArray(bolas) ⇒ true

O método Array.isArray verifica se a variável é um array e retorna true ou false.

### Método .every( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].every( ( bola ) ⇒ { return bola === 🏐 } ) => false

O método every verifica se cada elemento do array passa em um teste. Esse teste é feito através de uma função callback. O método executa a função de callback para cada elemento uma vez e retorna true se o teste for true para todos os elementos, e false se o teste for false para pelo menos um elemento. Além disso, o método não executa a função callback para arrays vazios e não altera o array.

### Método .filter( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].filter( ( bola ) ⇒ { return bola !== 🏈 } ) => [ ⚾, ⚽, 🏀, 🏐 ]

O método filter usa uma função callback de teste e executa ela para cada elemento do array. Ele retorna um novo array com os elementos que passarem no teste.

### Método .map( )

[🏀, 🏀,🏀,🏀 ].map( console.log(( bola ) ⇒ { return 🏈 }) ) => [ 🏈, 🏈, 🏈, 🏈 ]

O método map usa uma função callback de teste e executa ela para cada elemento do array, retornando um novo array modificado. Não executa a função de callback para arrays vazios.

### Método .forEach( )

[🏀, 🏀,🏀,🏀 ].forEach( ( bola ) ⇒ { console.log( bola + ⚽)} ) => 🏀⚽🏀⚽🏀⚽

O método forEach usa uma função callback e executa ela para cada elemento do array. Não executa a função de callback para arrays vazios e não retorna um novo array, diferente do método map.

### Método .copyWithin( )

[ ⚾ , 🏈, ⚽, 🏀 ].copyWithin(2,0) ⇒ [ ⚾ , 🏈, ⚾ , 🏈 ]

O método copyWithin copia os elementos do array para outra posição no array, ele não adiciona itens apenas substitui os valores existentes.

### Método .lastIndexOf( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].lastIndexOf(⚽) ⇒ 2

O método lastIndexOf retorna o valor do último índice especificado, se o valor não for encontrado ele retorna -1.

### Método .valueOf( )

[ ⚽, ⚾ , 🏀, 🏐 ].valueOf() ⇒ [ ⚽, ⚾ , 🏀, 🏐 ]

O método valueOf é usado para retornar o array. É um método padrão do objeto Array. Este método retorna todos os itens na mesma matriz. Ele não altera o conteúdo original da matriz e não contém nenhum valor de parâmetro.

### Método .reduce( )

[ 🥦, 🍅, 🥕, 🍆 ].reduce((valor, elemento) => valor + elemento, 0)) ⇒ 🥗

O método reduce executa uma função redutora para o elemento array, ele retorna um único valor que é o resultado acumulado da função. Ele não executa a função para elementos de array vazios e não altera a matriz original.

### Método .keys( )

[ ⚾ , 🏈, ⚽, 🏀, 🏐 ].keys() => ["0","1","2","3"]

O método keys retorna um novo array composto pelas chaves (posições) do array o qual ele foi aplicado.

[00:00] Se você chegou até aqui, parabéns! Você está finalizando mais um treinamento da Alura. Nesse curso, aprendemos muito sobre métodos de array com JavaScript.

[00:09] E utilizamos um projeto do mundo real para aprender como aplicar ordenação, como aplicar filtro, como aplicar método forEach, map, reduce, a diferença entre eles.

[00:21] Realizamos uma série de atividades e exercícios para reforçar ainda mais o nosso conhecimento. E eu queria parabenizar você por finalizar mais esse treinamento.

[00:29] Lembrando que, se você ficou alguma dúvida, recorra ao fórum. Se você não tem dúvida, vai lá no fórum, ajuda alguém, reforça o seu conhecimento e faça cada vez mais a comunidade da Alura ser integrada e ser um ambiente bem legal de aprendizado para todo mundo.

[00:43] Eu queria deixar o meu abraço e nos encontraremos no nosso próximo treinamento. Tchau!