***JavaScript – Manipulando o DOM***

Agora que vocês já configuraram sua página do seu jeito, vamos começar efetivamente com o JavaScript. Para começar com o JavaScript você não precisa de nada, não é preciso um servidor ou uma grande estrutura. Só com o navegador é possível usar o JavaScript e fazer as interações funcionarem.

Aqui estou usando o Google Chrome. Para fazer isso, vamos abrir as ferramentas do desenvolvedor apertando o "F12". Em "Elementos" temos a nossa árvore do HTML com o head, o body e o main, tudo da forma como construímos.

<!DOCTYPE html>

<html lang="en">

<head>\_</head>

<body> = S0

<main>\_</main>

</body>

</html>

Temos também uma aba que vamos usar muito. Quando estivermos programando em JavaScript, essa é aba "Console" será a principal. É nela qua vamos inserir os comandos de JavaScript especialmente para teste e também onde veremos os erros acontecendo.

Antes de usar isso, voltaremos lá no nosso HTML. Nele temos nossa página inteira consrtuída e um input no fim do nosso HTML. Esse input tem um tipo (type) submit, um valor (value) de "Iniciar produção" e uma classe (class) producao.

Vamos criar aqui um identificador (id) chamado producao para que possamos usar esse identificador no JavaScript.

<input type="submit" value="Iniciar produção" class="producao" id="producao">

É com esse elemento que vamos interagir agora. No console, vamos digitar document.getElementById(). Percebam que forma interessante de escrever aqui, escrevemos a primeira palavra *get* com a letra minúscula e cada palavra seguinte começamos com a letras maiúscula para fazer a divisão.

Aqui estamso buscando o elemento pelo id e o id que criamos foi producao. Notem o que acontece quando damos o Enter.

document.getElementById('producao')

<input type="submit" value="Iniciar produção" class="producao" id="producao">

O JavaScript foi até o HTML e nos trouxe o elemento. Reparem que carregando o HTML ele é exatamente igual ao que vemos na página. Com a seta para cima, podemos voltar à último linha digitada. Podemos mudar qualquer elemento, podemos colocar no console value="Teste de JavaScript" e dar um Enter.

document.getElementById('producao').value="Teste JavaScript"

'Teste JavaScript'

Com isso, o botão da nossa página passa a ter texto "Teste JavaScript" em vez de "Iniciar Produção" como era anteriormente. Nós buscamos um elemento pelo identificador e manipulamos esse elemento com o JavaScript.

Ao longo de todo o seu aprendizado nós vamos usar o JavaScript para interagir com a nossa página, nosso HTML. Adicionar e retirar informações, manipular dados, evento e comportamentos, tudo isso com o JavaScript. É isso que veremos em toda essa formação e vamos começar pelo mais simples: interagir com um único elemento.

Mexemos no id do elemento, buscamos um elemento pelo identificador, mas é muito raro adicionarmos identificadores nos elementos. Geralmente ao criarmo uma página usamos muitas classes. Como a classe cria um conteúdo reaproveitável no CSS, ela é muito mais usada. Por exemplo, a classe estatistica.

Se formos ao console, e colocarmos document.getElements o console já sugere algumas opções, dentre elas getElementsByClassName(), buscando pela classe. Então, vamos buscar pela classe estatistica.

document.getElementByClassName('estatistica')

HTMLCollection(4) [div.estatistica, div.estatistica, div.estatistica, div.estatistica]

Repare que ele nos retornar uma coleção de elementos do HTML que tem quatro elementos. Ele nos retorna um *array* com quatro elementos. Ele me retorna esses dados e esses dados eu posso manipular

É preciso gravarmos quando é um id ou uma classe para conseguirmos buscar esse elementos? É preciso gravar os comandos diferentes? Não obrigatoriamente. Se olharmos nosso css, podemos interagir com os elementos pelo nome da tag, com uma classe usando o ponto ., podemos navegar no css e buscar um elemento depois do outro, enfim, é possível fazer qualquer busca aqui. Se quisermos mudar um identificador, colocamos # e o identificador, podemos colocar o identificador e dentro dele uma tag, não tem problema nenhum. O css navega na página e busca esse elemento para colocar o estilo.

O JavaScript faz o mesmo usando uma função uma pouco diferente, que é querySelector(). Com ela uma busca é feita no seletor que configurarmos.

document.querySelector()

Se realmente quisermos o id de produção, colocamos #producao. E como no css, ele nos retorna o input cujo id é producao.

document.querySelector('#producao')

<input type="submit" value="Teste JavaScript" class="producao" id="producao">

Então não é preciso grava getElementById, getElementByClassName ou getElementByTagName, que busca o elemento pela sua tag, nada disso é preciso. Usando o querySelector chegamos em todos os elementos baseados na mesma estrutura da nossa navegação pelo css.

Vocês viram que usamos aqui a palavra "document" por várias vezes. O document é onde o JavaScript vai buscar o que estamos pedindo, ou seja, no nosso HTML. Mas para mergulhar um pouco mais no document e entendermos como funciona o **DOM**, convido vocês aos próximos vídeos.

**Sobre o DOM**

Saber manipular elementos no DOM é essencial para uma pessoa desenvolvedora Web. Dessa maneira, assinale a alternativa abaixo que contenha informações corretas acerca dessa API.

O DOM é uma interface de linguagem neutra que permite que o documento HTML seja atualizado dinamicamente.

Alternativa correta! O DOM permite atualizar, excluir e modificar dinamicamente elementos do documento HTML!

Usamos a palavra "document" no console do navegador e entendemos que ela tem alguns poderes. Mas, afinal, o que é essa palavra?

Sempre que o JavaScript vai agir com o navegador, ele navega em uma coisa chamada **DOM**. DOM é a sigla de Document Object Model, que em português significa **modelo de objeto do documento**. O nosso HTML é um documento e para lê-lo o navegador o transforme em um objeto. Um objeto manipulável, um objeto na estrutura do JavaScript.

Se chegarmos em nosso console e digitarmos apenas document vamos entender um pouco mais.

> document

#document

<!DOCTYPE html>

<html lang="en">

<head>\_</head>

<body>\_<body>

</html>

Na estrutura do document elenos retornou o html. Aqui no Chrome, ele exibe a árvores do HTML limpa para nós. O document é, portanto, o nosso arquivo de html. Porém, ele é o arquivo html com super poderes. Vamos abrir aqui o Firefox para vermos uma representação um pouco diferente do document. Com a mesma página aberta digitamos document no console.

document

HTMLDocument file:///User/pedromarins/code/alura/robotron-200/index.html

Enquanto no Chrome ele apresenta um identificador e um document aqui ele diz que é um documento do tipo HTML. Quando abrimos ele, todos os "super poderes" que o HTML possui são apresentados e todos esses poderes que o JavaScript ao ler a página também passa a ter.

Se por exemplo, digitarmos no console do Chrome document.URL ele nos retornará qual é a url que estamos usando no navegador.

document.URL

'file://Users/pedromarins/code/alura/robotron-200/index.htmlCOPIAR CÓDIGO

Se digitarmo document.title, ele retorna o título que está dentro da tag <title>.

document.title

'Robotron 2000'

Se digitarmo document.body ele retorna todos o body criado.

document.body

<body>

<main>\_</main>

</body>

O mesmo acontece se digitarmo document.head, ele vai retornar tudo.

document.head

<head>

Agora, podemos usar o body e podemos usar o head então por que não podemos usar document.producao que nós criamos anteriormente?

document.producao

undefined

O body e o head são tags padrão da nossa página, elas sempre estarão presentes, ou seja, esse métodos já existem no JavaScript e no DOM. Agora, quando queremos procurar uma classe ou um id que nós criamos, usamos os métodos próprios de busca como o document.getElementById('producao') ou o querySelector('#producao') e o resultado é o mesmo.

document.getElementById('producao')

<input type="submit" value="Teste JavaScript" class"producao" id="producao">

querySelector('#producao')

<input type="submit" value="Teste JavaScript" class"producao" id="producao">

Vimos como apenas dois navegadores se comportam para que vocês entendam que duas ferramentas diferentes estão olhando e manipulando a mesma página da mesma forma e usando a mesma estrutura de JavaScript. Lógico que alguns comportamentos podem ser diferentes de alguns navegadores para outros, mas em sua a maioria é o JavaScript manipulando a páginas por meio dessa busca avançado dos elementos, dos eventos e de melhorias em nosso código.

Para fazer a primeira melhoria e começarmos a manipular de fato a nossa página com JavaScript, nosso desafio será montar o nosso robô para a batalha. Vamos adicionar braço, blindagem, e núcleos. Mas isso é um desafio para a próxima aula.

**Para saber mais: selecionando elementos**

O DOM (Document Object Model) e o JavaScript, juntos, possuem grande poder de modificar dinamicamente a estrutura de um documento HTML. Sendo possível, por exemplo:

* Adicionar/modificar/remover tags, textos, imagens e qualquer elemento no HTML.
* Alterar estilos CSS da página.
* Criar novos eventos HTML.

Vamos conferir formas de realizar os itens listados no tópico acima.

**Métodos para selecionar elementos no HTML**

* document.getElementByID(id) - Selecionar um elemento pelo ID.
* document.getElementsByTagName(name) - Selecionar um elemento pelo nome.
* document.getElementsByClassName(name) - Selecionar um elemento pelo nome da classe.

**Propriedades e métodos para alterar elementos no HTML**

* element.innerHTML - Esta propriedade obtém ou altera qualquer elemento no HTML, inclusive tags.
* element.innerText - Esta propriedade permite inserir textos no HTML.
* element.attribute - Esta propriedade altera o valor de um elemento HTML
* element.setAttribute(atributo, valor) - Este método altera o valor de um atributo de um elemento HTML.

**Adicionando e excluindo elementos**

* document.write() - Escreve no fluxo de saída do HTML.
* document.appendChild() - Adiciona um elemento HTML.
* document.removeChild() - Remove um elemento HTML.
* document.replaceChild() - Substitui um elemento HTML.
* document.createElement() - Cria um elemento HTML.

**Consolidando o seu conhecimento**

Praticar é fundamental para fixar o conhecimento adquirido, para isso, temos o primeiro desafio! Utilizando JavaScript, substitua a frase dentro das tags <p> </p> do projeto disponibilizado abaixo:

OBS: PROJETO DENTRO DA PASTA TESTE.

Como fazer:

* Utilize o escritor de código de sua preferência;
* Crie dois arquivos index.html e style.css;
* Copie o código HTML e cole no arquivo index.html e o código CSS no arquivo style.css;
* Abra o projeto no navegador, aperte o atalho F12 e realize a mudança escrevendo os comandos no console.

VER OPINIÃO DO INSTRUTOR

### Opinião do instrutor

document.getElementById('texto').innerText="Escreva o texto aqui"

Para inserir um texto é necessário utilizar a propriedade innerText.

# O que aprendemos?

## Nessa aula:

* Desenvolvemos os primeiros passos do projeto **Robotron 2000**;
* Definimos o que é o DOM;
* Aprendemos como manipular elementos no DOM.

**Eventos com JavaScript**

Agora que entendemos como o DOM funciona, vamos começar a interagir de verdade com nossa página usando o JavaScript! Para criarmos um código JavaScript que interaja com nossa página temos algumas opções e entre elas usar diretamente o console.

Nele, vamos criar um alerta alert com a mensagem Oi e dar Enter.

alert('Oi')

Esse alerta aparece na nossa página com a mensagem "This page says oi" e temos que dar um "Ok" para ele sumir. Toda vez que estivermos interagindo com a página, é bom deixarmos o console aberto.

O alerta esstá funcionando, mas não é exatamente isso que queremos. Nós queremos que esse código esteja no JavaScript porque se recarregarmos a nossa página, o alerta some. Quando essa cpodigo está no HTML, podemos recarregar a página e o código está salvo.

Em nossa img class="robo" vamos adicinar a função onclick. O JavaScript trabalha com eventos, toda vez que algo acontece, ele se manifesta. Nesse caso queremos escutar o evento do click: toda vez que ele acontecer, nós fazemos algo. Portanto, no onclick damos um alert com oi.

<img class="robo" src="img/robotron.png alt="Robotron" onclick="alert('oi')">

Quando voltramos ao navegador, recarregarmos a página e clicarmos no robô, o alerta "oi" vai aparecer. O Robotron começou a se comunicar conosco, é a evolução das máquinas!

Agora, queremos que esse código esteja repetível. O grande problema de fazermos um código inline é que todas as vezes que quisermos alterar o código JavaScript, precisaremos mexer no arquivo HTML. Isso não faz muito sentido porque se estamos mexendo com Java Script, deveríamos mexer exclusivamente no arquivo de JavaScript.

Como fazemos para retirar essa manifestação do JavaScript do HTML? Vamos criar um arquivo JavaScript! Vamos criar uma pasata chamada "js" e dentro dela, criarmos um arquivo chamado "main.js", que ´o nome convecionalmente usado para arquivo padrão da JavaScript. Assim como usamos convencionalmente "style.css" para o arquivo de css.

No "main.js" fazemos o alert('oi'). Mas se recaarregarmos a página logo em seguida não vai acontecer nada porque não chamamos esse arquivo em nosso HTML. Assim como no css, é preciso fazer um link entre esses dois arquivos. Porém, diferente do arquivo de css, esse link não é feito com a tag link. Para linkar um arquivo de script, usamos a tag script com o source src para descobrir a localizaão desse arquivo. Nesse caso, o arquivo está dentro da pasta js com o nome main.js.

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

Agora o alerta funciona em nosso navegador! Porém, ele travou o carregamento da página de modo que todo o restanto ficou em branco.

Primeiro, como desenvolvedores, nós não usamos o alert para dar erros, nós usamos o console.log(). Com ele, além de não travar o carregamento da página, ele aparece direto em nosso console.

O carregamento da página foi travado porque o carregamento do arquivo JavaScript faz isso! Toda vez que nosso navegador encontra um arquivo JavaScript na leitura do html, ele para tudo até ler aquele aqruivo inteiro e executá-lo antes de passar para a próxima linha.

Para evitar problema, nós sempre colocamos o arquivo JavaScript na última linha possível, isso garantirá que nosso html seja lido inteiramente e carregado antes do js. Assim, quando ele estiver lendo o arquivo js, a página já vai estar dispnível para o usuário.

Agora que temos um console funcionando, precisamos adicionar novamente o evento de um click seja mapeado. Para isso, usamos a função addEventListener. Essa função é um "escutador de eventos" e ela receberá um evento do tipo click.

.addEventListener("click")

Reparem, porém, que o VSCode já está apontando um erro porque sempre que um evento de click é adicionado, ele precisa ser adicionado a um elemento. No index.html, adicionaremos ele no robotron. Por isso, vamos adicionar esse id a robo.

<img class="robo" src="img/robotron.png alt="Robotron" id="robotron">

No js buscamos ele por meio de document.querySelector() buscando pelo id #robotron.

document.querySelector("#robotron").addEventListener("click")

Agora podemos ter um evento de click nesse elemento! Contudo, a função addEventListener() espera algo além do click, ela espera receber uma operação para executar. E podemos passar aqui um console.log dando oi.

document.querySelector("#robotron").addEventListener("click", console.log('oi'))

Entretanto, quando carregamos a página essa função já foi executada e o "oi" já está no console e quando clicamos, não acontece mais nada!

Quando queremos que algo execute um pedaço de código, é preciso extrair isso para uma função. No JavaScript a função serve para armazenarmos pedaços de código.

Por esta razão, vamos criar a função dizOi. A função tem essa grafia, ela começa com function seguido pelo nome da função que abre e fecha parênteses () e abre e fecha chaves {}.

function dizOi() {

}

E dentro dela mandaremos executar o nosso console.log().

function dizOi() {

console.log('oi')

}

Agora o que precisamos é que na hora de clicar, ele execute dizOi.

document.querySelector("#robotron").addEventListener("click", dizOi)

function dizOi() {

console.log('oi')

}

Recarregamos a página e ao clicar no botão, o console soma todas as vezes que a página diz "oi".

Não temos apenas essa operação para salvar o código, ainda podemos salvar elementos, criar variáveis. Por exemplo, vamos criar uma constante const, que é um tipo de variável que não é alterada, chamada robotron. Essa constante receberá toda a busca que fizemos do nosso elemento.

const robotron = document.querySelector("#robotron")

Agora o addEventListeiner() ficou perdido, ele não está mais sendo executado em nada. Por essa razão, vamos colocá-lo para ser executado no robotron.

const robotron = document.querySelector("#robotron")

robotron.addEventListener,("click", dizOi)

function dizOi() {

console.log('oi')

}

Vamos salvar esse código e recarregar nossa página no navegador. Ao clicar no robô, ele continua dizendo "oi".

Nesse vídeo nós separamos o nosso código. Criamos um arquivo JavaScript para podermos criar códigos JavaScript. Entendemos como funciona o click do elemento por meio de um escutador de eventos e separamos código JavaScript quando armazenado para ser usado muitas vezes. Aprendemos que nesse caso ele precisa estar dentro de uma função e que os elementos buscados pelo JavaScript podem ficar dentro de uma variável.

Não é simples, mas com a prática e a continuidade dos nossos exercícios, isso vai ficando muito claro para vocês. Vamos explorar um pouco mais das funções do JavaScript no próximo vídeo!

Código referente a aula acima:

JavaScript:

*const* robotrom = document.querySelector("#robotrom")

robotrom.addEventListener("click", dizOi)

*function* dizOi() {

    console.log ("oi")

}

**EventListener & querySelector**

Na última aula, vimos como utilizar os métodos addEventListener() e querySelector(). Em conjunto, eles possibilitam receber e retornar informações à pessoa usuária.

Tendo como base o conhecimento adquirido até aqui, assinale a alternativa que contenha um conjunto de código que retorne a palavra “Confirma” com evento de clique:

function myFunction() {

document.querySelector("#retorna").innerHTML = "Confirma";

}

const elemento = document.querySelector("#botao");

elemento.addEventListener("click", myFunction);

Alternativa correta! Mesmo a const sendo declarada após a função, o código funciona sem problemas. Isso acontece devido ao comportamento de hoisting no JavaScript.

**Funções com JavaScript**

O JavaScript e toda a linguagem de programação tem a opção de armazenar dados. Gosto de explicar a função como uma forma de armazenarmos códigos que serão repetidos várias vezes. A função é usada para que esse código seja chamado mais de um vez.

Vamos supor que ao carregar nossa página nós queiramos executar esse bloco de códigos.

const robotron = document.querySelector("#robotron")

robotron.addEventListener("click", dizOi)

function dizOi() {

console.log('Oi')

console.log("Bem-vindo ao Robotron 2000")

}

Duas linhas de código, duas impressões no colole. Queremos que todas as vezes que a página for carregada, a função dizOi seja executada.

Para isso, nós chamamos esse função colocando o nome dela e abrindo e fechando parênteses, assim a função será executada no momento exato em que ela for lida.

dizOi()

Recarregando a página, ao ler o arquivo JavaScript o código já foi executado e ao clicar no robô o código é executado novamente. Uma função nos ajuda a repetir blocos de código. Essa é uma função nomeada, chamada de **declaração** de uma função. Contudo, essa não é a única forma de criar um função no JavaScript.

Nessa lingagem, podemos criar uma função anônima, ou seja, uma função sem nome. Se ao clicar no Robotron, nós quisessemos que esse código fosse executado apenas uma vez, poderíamos usar uma função anônima.

Para isso, declaramos ela em addEventListener(). Não colocamos nome nela, colocamos o parênteses, abrimos chaves e executamos o código.

robotron.addEventListener("click", function() {

})

Em seguida colocamos outro console.log que será Cliquei no robô.

robotron.addEventListener("click", function() {

console.log('Cliquei nno robô.')

})

Assim que executamos a página, a função dizOi é executada, ele leu a linha e executou e agora todas as vezes que clicarmos no robô ele executa a função anônima.

A diferença da função anônima para a declarada é que a declara pode ser chamada mais vezes por ter um nome. Uma função anônima só pode ser executada uma única vez, quando um gatilho for disparado, nesse caso, quando o evento do click for disparado.

As novas versões do JavaScript a partir do ES6 ainda tem uma forma diferente de declaramos uma função anônima. No lugar de usarmos function(), nós abrimos parênteses e colocamos uma seta formada por um sinal de igual e um maior =>.

robotron.addEventListener("click", () => {

console.log('Cliquei nno robô.')

})

Com isso, economizamos na escrita de uma função, mas continuamos identificando-a claramente como um função anônima. Essa é uma **arrow function** (em tradução livre: função de setas) para declarar a função que não será nomeada novamente.

Se recarregarmos a página, ele novamente executa dizOi e se clicamos no robô ele está interagindo com nossa página.

Vimos a criação de uma função, a criação de uma função anônima e a criação de uma arrow function. Porém, uma das coisas mais importantes sobre uma função é podermos manipular os dados, mandar uma função dizer "oi" para qualquer pessoa. Em vez de dizermos "Bem-vindo ao Robotron 2000" podemos especificar o que será dito.

Vamos dar uma olhada no console. Se nele executarmos a função dizOi ele só executa.

dizOi()

Oi

Bem-vindo ao Robotron 2000

Mas e se quisermos falar "Oi, Pedro" ou só "Pedro, bem-vindo ao Robotron 2000"? Precisaremos que essa função receba um parâmetro. O parâmetro vem dentro dos parênteses da criação de uma função. Aqui, o nome desse parâmetro será nome.

function dizOi(nome) {

console.log('Oi')

console.log("Bem-vindo ao Robotron 2000")

}COPIAR CÓDIGO

Essa função vai receber um nome e vai imprimir esse nome.

function dizOi(nome) {

console.log(nome)

console.log("Bem-vindo ao Robotron 2000")

}

E todas as vezes que chamarmos essa função é preciso dizer o que queremos imprimir, nesse caso Pedro.

dizOi("Pedro")

Se recarregarmos a página, veremos "Pedro. Bem -vindo ao Robotron 2000" aqui no nosso console.

Pedro

Bem-vindo ao Robotron 2000

Vamos reparar no que aconteceu: nós passamos um texto para uma função, ela recebeu esse texto e fez algo com ele. Se quisemos somar algo, por exemplo "Oi + nome", seria possível.

Oi Pedro

Bem-vindo ao Robotron 2000

Podemos passar, inclusive, outros valores como dizOi('Renata') ou dizOi('Paulo')

dizOi("Renata")

Oi Renata

Bem-vindo ao Robotron 2000

dizOi('Paulo')

Oi Paulo

Bem-vindo ao Robotron 2000

Passamos um parâmetro para essa função e essa função a executou!

Entendemos aqui como passar um texto como parâmetro, mas já existem parâmetros que são padrão da nossa página. Por exemplo, toda vez que um evento acontece, podemos manipular esse evento. E em uma arrow function, colocamos entre os parênteses o que usaremos, que é o evento. E colocamos no corpo da função do console.log o evento.

robotron.addEventListener("click", (evento) => {

console.log(evento)

})

Recarregamos a página e toda vez que clicarmos, o console retorna tudo o que aconteceu, além de executar tudo que queremos.

PointerEvent {isTruste: true, pointerId: 1, width: 1, height: 1, pressure: 0, \_}

Por que manipular o evento é importante? Porque agora podemos clicar em qualquer um dos itens para somar as peças em nosso robô! Montar um robô ainda mais incrível para começar a produção e combater a invasão alienígena.

Contudo, para combater a invasão alienígena primeiro temos que manipular os eventos de click em nosso formulário de peças. E isso é assunto para o próximo vídeo.

Código ultilizado em aula :

JavaScript:

*const* robotrom = document.querySelector("#robotrom")

robotrom.addEventListener("click", (*evento*) => {

    console.log(evento)

})

*function* dizOi(*nome*) {

    console.log (nome + "" + "Bem vindo ao Robotrom 2000")

}

dizOi ("Marcos")

# Hoisting no Javascript

Quando uma equipe trabalha em algum projeto, muito provavelmente o código será lido e alterado por diversas pessoas, dessa maneira, é necessário entender diferentes formas possíveis de escrever o mesmo código, obtendo uma boa produtividade na sua performance de desenvolvimento.

Contudo, é importante para uma **pessoa programadora** entender o **que é Hoisting** e como se comporta em diferentes casos nesta linguagem.

Vamos analisar este **código de function declaration**, e o que retorna:

console.log(soma(2, 5))

function soma(a, b) {

return a + b

}

Este código retorna o valor: 7

Repare que a função consegue ser chamada antes mesmo de ter sido declarada. **Hoisting** é o termo que explica essa situação, em português ele significa “içamento”, ou “elevação” e foi citado pela primeira vez no [**ECMAScript® 2015 Language Specification**](https://262.ecma-international.org/6.0/). O Hoisting permite que você execute funções antes das suas declarações. Na prática, inicialmente as **declarações de funções** são colocadas na memória durante a [fase de compilação](https://www.alura.com.br/artigos/o-que-e-compilacao) e, mesmo assim, permanecem no mesmo lugar que estão digitadas.

A razão pela qual o código anterior funciona é que os mecanismos JavaScript movem a função soma para o início do escopo, como ilustra o código abaixo:

function soma(a,b){

return a + b

}

console.log(soma(2,5))

Entretanto, será que Hoisting funciona em outros tipos de código? Vamos conferir!

console.log(alura)

var alura = ‘cursos’;

Este código retorna:

undefined

A utilização de **Hoisting em var não é indicada**, pois a variável criada é elevada para o escopo, mas sem o seu valor, e com isso, retorna valor undefined.

O comportamento é parecido utilizando também var function:

function testaHoisting() {

console.log(‘testaHoisting’, alura)

var alura = ‘cursos’

}

Este código retorna: undefined.

**Hoisting também não é indicado utilizando let**, pois acontece um outro tipo de **comportamento não desejado:**

console.log(alura)

let alura = ‘cursos’;

Este código retorna:

script.js:1 Uncaught ReferenceError: Cannot access 'alura' before initialization```

Esta mensagem em português significa “Não é possível acessar ‘alura’ antes de sua inicialização'', ou seja, a linguagem JavaScript reconhece que ‘let = alura’ existe, porém não consegue acessar sua declaração. Esse comportamento de Hoisting com let é o mesmo ao utilizar const.

**Tipos de funções**

Saber identificar, e utilizar corretamente função de uma função anônima é essencial para uma pessoa desenvolvedora em JavaScript.

Assinale a alternativa que contenha informações corretas em relação a estes tipos de funções:

Toda arrow function é também uma função anônima. Uma mesma função anônima não pode ser declarada mais de uma vez, apenas funções.

Alternativa correta! Como uma função anônima não possui nome, ela precisa sempre ser declarada onde será executada, diferente de funções que podem ser declaradas mais de uma vez.

# O que aprendemos?

## Nessa aula:

* Aprendemos a selecionar um ou mais elementos com querySelector()/querySelectorAll();
* Aprendemos como utilizar o método addEventListener();
* Definimos as diferenças entre função anônima e função;
* Entendemos o comportamento de hoisting no JavaScript;

**Alterando um item na página**

Até aqui já prendemos a buscar um elemento, adicionar um evento de click e executar uma função! Agora, temos tudo que precisamos para fazer o montador de peças do Robotron funcionar.

Começaremos adicionando e subtraindo braços no Robotron. Primeiro, nosso html. Em nosso html temos aqui o Robotron, as estatístitcas e equipamentos. Temos o montador de casa uma das peças.

Nesse momento, não quero interagir com todas as outras, quero interagir só com o controle das peças dos braços. Então, temos uma classe em um botão que é o controle-ajuste que é o negativo e tem uma classe que é o positivo. Ambos na classe controle-ajuste.

<div class="controle">

<buttom class="controle-ajuste">\_</buttom>

input type="text" class="controle-contador" value="00">

buttom class="controle-ajusta"></buttom>

</div>

Para facilitar nossa vida, vamos começar com um identificador para subtrair id="subtrair" no primeiro buttom class e um id="somar" no segundo buttom class. No input, colocaremos o id de braco.

<buttom class="controle-ajuste" id="subtrair">\_</buttom>

input type="text" class="controle-contador" value="00" id="braco">

buttom class="controle-ajusta" id="somar"></buttom>COPIAR CÓDIGO

Pronto! Temos nossa estrutura do HTML. Agora, vamos para o javaScript. Temos o subtrair, o somar e o brcao então vamosa salvar essas três variáveis. Em "main.js" vamos esquecer a variável Robotron e vamos trocar por essas três por enquanto.

const subtrair = document.querySelector("#subtrair")

const somar = document.querySelector("#somar")

const braco = document.querySelector("#braco")[

Já temos os três elementos. O que nós queremos? Toda vez que tiver um clique no subtrair, queremos que tire um. Sempre que tiver um clique no somar queremos que adicionae um. Vamos começar pelo somar.

somar.addEventListener("click", (evento) => {

})

No somar toda vez que clicarmos queremos pegar o braco e no value dele incrimentar 1.

somar.addEventListener("click", (evento) => {

braco.value += 1;

})

Vamos ver o que acontece agora ao recarregarmos nosso navegador. Ao apertarmos o botão "+" de "Braços" estamos somando "1", mas como um incremento.

Existe um grande problema quando estamos manipulando os dados de um formulário que é o valor que vem para nós vem como uma string, um texto. Vamos fazer um experimento no console. Aqui, temos um 00 - porque é uma string - e toda vez que somamos 1, veja o resultado.

"00" + 1

'001'

E se pegarmos esse resultado e somarmos mais um?

"001" + 1

'0011'

No JavaScript toda vez que temos uma soma entre uma string e um número, ele vai pegar a string que é o tipo mais forte e vai tranformar tudo em string. O resultado será uma string modificada, adicionada àquele número.

Não queremos que esse número dinamicamente vire uma soma ou uma subtração. Para fazer isso, precisamos tranformar o valor de braco.value em um número. A operção parseInt() faz isso.

somar.addEventListener("click", (evento) => {

parseInt(braco.value);

})

Vamos dar um console.log() nela para que o resultado fique visível.

somar.addEventListener("click", (evento) => {

console.log(parseInt(braco.value));

})

Quando clicamos no botão "+" de "Braço" ele imprime um 0, que é um número. Agora braco.value pode receber o parseInt() do braco.value mais 1

somar.addEventListener("click", (evento) => {

braco.value = parseInt(braco.value) + 1;

})

Agora quando carregamos o navegador, ele está somando! Podemos somar 10, 12 braços, quantos quisermos. Vamos, então, fazer o subtrair! E para subtrair, não tem nenhum detalhe extra, é só pegarmos o subtrair e diminuir 1. O mesmo código vai fazer a subtração e a adição.

subtrair.addEventListener("click", (evento) => {

braco.value = parseInt(braco.value) - 1;

})

E agora conseguimos adicionar e remover braços do nosso robô! Com nossos elementos de mais e menos funcionando, precisamos apenas que todos os outros itens tenham o mesmo comportamento. Porém, não é saudável ficarmos criando códigos que serão repetidos diversas vezes. Por isso, vamos aprender a melhorar essa função a fim de deixarmos o código dinâmico. Este será o assunto do nosso próximo vídeo.

**Reaproveitando código**

Criamos um código capaz de adicionar e subtrair os itens do nosso montador do Robotron. A validação de braço negativos, ainfla um robô não pode ir com braços negativos para batalha, nós veremos no Módulo 2. Por enquanto, esse não é o assunto que queremos tratar.

O que vamos ver agora é como reaproveitamos esse código com uma função.

const subtrair = document.querySelector("#subtrair")

const somar = document.querySelector("#somar")

const braco = document.querySelector("#braco")

somar.addEventListener("click", (evento) => {

braco.value = parseInt(braco.value) + 1;

})

subtrair.addEventListener("click", (evento) => {

braco.value = parseInt(braco.value) - 1;

})

Vamos entender agora na prática para que servem as funções. Temos esse código parseInt(braco.value) que se repete e nós não precisamos jogar essa validação no elemento. Podemos terceirizar essa validação para um outro lugar.

Criaremos uma funcão chamada manipulaDados() que recebe um parâmetro que será a operacao.

function manipulaDados(operacao)COPIAR CÓDIGO

Se essa operação for, por exemplo, subtrair, então queremos tirar um item do nosso contador de braco.value senão queremos somar.

function manipulaDados(operacao) {

if(operacao === "subtrair") {

braco.value = parseInt(braco.value) - 1

} else {

braco.value = parseInt(braco.value) + 1

}

}

Agora trouxemos toda a lógica de manipulação dos dados para um único lugar e só precisamos mandar o dados para aquele lugar que toda a operação será realizada. Assim, separamos as responsabilidades. A única responsabilidade do evento do click é a avisar para o maniúlaDados() que aconteceu um clique. E a função o manipulaDados() é interagir e realmente mudar o valor.

Em addEventListener() não precisamos do parâmetro evento, precisamos apenas chamar a função manipulaDados() passando o que é esperado, somar e subtrair, respectivamente.

somar.addEventListener("click", () => (manipulaDados("somar"))

subtrair.addEventListener("click", () => (manipulaDados("subtrair"))

Salvando isso e recarregando nossa página, conseguimos somar e subtrair os elementos! Vamos mergulhar um pouco mais nessa lógica porque queremos somar e subtrair todos os elementos (braços, blindagem, núcleos, pernas e foguetes). E no html nós só adicionamos um identificados. Será que podemos fazer isso usando a classe controle-ajuste?

Para buscar esse elemento vamos chamar isso de controle que receberá o document.querySelector() com a classe .controle-ajuste.

const controle = document.querySelector(".controle-ajuste")

Em seguida damos um console.log(controle) para vermos o que esse querySelector() conseguiu capturar.

<buttom class="controle-ajuste id="subtrair">-</buttom>

Ele pegou um botão que tem o id subtrair. Notem que o querySelector() buscou apenas um elemento, mas nós queremos buscar todos. Para isso, usamos a operação querySelectorAll().

const controle = document.querySelectorAll(".controle-ajuste")

Todos os elementos que tiverem essa classe, agora serão buscados.

NodeList(10) {

buttom#subtrair.controle-ajuste,

buttom#soma.controle-ajuste,

buttom.controle-ajuste

buttom.controle-ajuste

buttom.controle-ajuste

buttom.controle-ajuste

buttom.controle-ajuste

buttom.controle-ajuste

buttom.controle-ajuste

buttom.controle-ajuste

Temos aqui uma lista de nós, ou seja, um array com todos esses elementos! Ou seja, todos esses elementos agora estão manipuláveis nessa função do controle.

Como fazemos para criar um evento para todos esses elementos sem que precisemos criar dez código diferentes? Vamos manipular esse array para que ele faça isso pra mim. Mas isso é assunto para o próximo vídeo.

Código referente a aula acima:

JavaScript:

*const* subtrair = document.querySelector("#subtrair")

*const* somar = document.querySelector("#somar")

*const* braco = document.querySelector("#braco")

somar.addEventListener("click", () => {manipulaDados("somar")}) *//função que ao clicar irá chamar a função de manipulaDados para acrescentar ou diminuir dependendo do botao*

subtrair.addEventListener("click", () => {manipulaDados("subtrair")}) *//função que ao clicar irá chamar a função de manipulaDados para acrescentar ou diminuir dependendo do botao*

*function* manipulaDados(*operacao*) { *//Função que adiciona valores , recebe como parametro operacao*

    if (operacao === "somar") { *//Se operacao for igual a "somar" adiciona + 1 no .value do braco*

    braco.value = parseInt(braco.value) + 1

} else {  *//Se não no caso for clicado o outro botao com o intuito de diminuir subtrai dele -1 .value do braco*

    braco.value = parseInt(braco.value) - 1

    }

}

**Alterando vários componentes**

Temos uma função que manipula os dados a partir de um parâmetro e temos uma variável com todos os nossos controles. O que precisamos agora é adicionar em cada controle um escutador de eventos.

Existem algumas operações muito legais para manipularmos arrayss, por exemplo, podemos fazer os loops de interação com o for ou podemos usar o while. Mas o array tem uma operação própria chamada **forEach()**. Com essa função do Array, conseguimos fazer esse loop sem inventar muito.

O forEach() é muito simples de ser utilizado, toda vez ele é usado temos um elemento e um índice. O elemento é aquilo que foi clicado e o índice é o indíce do array. Então controle.forEach() faz algo.

Dentro dessa operação, vamos criar uma função de seta que vai executar algum código. Nesse momento queremos descobrir qual elemento foi clicado, então em foEach() temos a opção de pegar um elemento e vamos dar um console.log() no elemento para entender se está funcionando.

controle.forEach( (elemento) => {

console.log(elemento)

)}

No console, ele retorna todos os dez elementos que queríamos. Precisamos agora adicionar um escutador de evento como aquele da função addEventListener() novamente recebendo o click fazendo algo () =>. E damos um console.log() de funcionou.

controle.forEach( (elemento) => {

elemento.addEventListener('click', () => {

console.log("funcionou")

)}

)}

Recarregamos o navegador e funcionou! Em "Braços" clicamos em subtrair e em somar e funcionou, na "Blindagem" também, subtrair funcionou e somar funcionou. Muito bom!

Agora, em vez de dizer apenas que funcionou ele nos diga o que aconteceu, qual foi o evento. Já vimos que todo evento de click tem o prâmetro do evento com tudp que aconteceu ali, então vamos dar um console.log() no evento.

controle.forEach( (elemento) => {

elemento.addEventListener('click', () => {

console.log("evento")

)}

)}

Agora, quando clicamos em algum botão, ele nos traz o evento do click! Tem algo aqui do evento, que é o target, ou seja, onde o click aconteceu,. Com evento.target ele vai nos trazer isso.

controle.forEach( (elemento) => {

elemento.addEventListener('click', () => {

console.log(evento.target)

)}

)}

Agora quando clicamos ele traz o botão e quando somo, ele subtrai. A blindagem, o núcleo, as pernas e os foguetes ainda não estão fazendo isso pelo simples motivo que a função manipulaDados ainda não está entendendo isso tudo.

Precisamos passar o somar de o botão for "+" e subtrair se o botão for "-". Temos o evento.target e se olharmos no hmtl, o conteúdo desse buttom é um - e o conteúdo do outro é +.

<div class="controle">

<buttom class="controle-ajuste" id="subtrair">-</buttom>

<input type="text" class="controle-contador" value="" id="braco">

<buttom class="controle-ajuste" id="somar">+</buttom>

</div>

Não precisamos ficar entendendo se é somar ou subtrair, podemos simplesmente pegar aquele conteúdo .textContent. Toda vez que é um input é um value e toda vez que é um texto, é um textContent.

controle.forEach( (elemento) => {

elemento.addEventListener('click', () => {

console.log("evento.target.textContent")

)}

)}

Agora quando clicamos no mais, vem o mais + e quando clicamos no menos vem o menos -.

Nosso código está muito bom, porém estamos dando console.log()nisso! Queremos chamar a função manipulaDados, mas essa função espera um texto chamado subtrair ou um texto chamado somar e o resultado é + ou -.

Então, vamos rafatorar nossa função e melhorar a lógica do nosso código para em vez de subtrair ele esperar - e em vez de somar esperar +.

somar.addEventListener("click", () => (manipulaDados("somar")))

subtrair.addEventListener("click", () => (manipulaDados("subtrair")))

function manipulaDados(operacao) {

if(operacao === "subtrair") {

braco.value = parseInt(braco.value) - 1

} else {

braco.value = parseInt(braco.value) + 1

}

}

Quando somamos agora, ele está somando de dois em dois! Isso porque continuamos com somar.addEventListener e subtrair.addEventListener, não faz sentido eles estarem portanto vamos removê-los.

Agora, em Braços tanto o botão de somar quanto de subtrair funcionam, porém, os botões de "Blindagem" também estão somando e subtraindo em "Braços".

Reparem que em nossa função manipulaDados estamos interagindo somente com braco.value! Precisamos que esse dado também seja dinâmico a partir do elemento que nós clicarmos.

Vamos refatorar ainda mais esse código e deixá-lo melhor, mas isso é assunto para a próxima aula!

Código referente a aula acima:

*const* subtrair = document.querySelector("#subtrair")

*const* somar = document.querySelector("#somar")

*const* braco = document.querySelector("#braco")

*const* controle = document.querySelectorAll(".controle-ajuste") *//const que irá pegar todos os elemento que possuem a classe de ajuste de controle responsavel por conter o sinais de + e -*

controle.forEach( (*elemento*) => {  *//função que vai varrer o array de elementos que para cada elemento dentro irá executar algo*

    elemento.addEventListener("click", (*evento*) => { *//Para cada elemento desses que for clicado*

        manipulaDados(evento.target.textContent)  *//executa a minha função manipulaDados e pega o evento de click que está ocorrendo em x alvo de acordo com o conteudo de texto dentro dessa classe executa a manipulaDados no caso tem os sinais + e -*

    })

})

*function* manipulaDados(*operacao*) { *//Função que adiciona valores , recebe como parametro operacao*

    if (operacao === "+") { *//Se operacao for igual ao sinal "+" que e o conteudo de texto que tem dentro das classes .controle-ajuste adiciona + 1*

    braco.value = parseInt(braco.value) + 1

} else {   *//Se operacao não for igual ao sinal "+" quer dizer a outro sinal ou algo diminue -1*

    braco.value = parseInt(braco.value) - 1

    }

}

# Para saber mais: para que serve um Array?

Nesta aula, tivemos uma pequena introdução sobre Arrays. Quando aplicamos o código abaixo, uma lista é criada com todos os elementos HTML que possuem a classe .controle-ajuste:

const controle = document.querySelectorAll(".controle-ajuste")

Esses elementos ficam guardados na const controle, e podem ser manipulados posteriormente ao avançarmos com o projeto **Robotron 2000**.

Para um melhor entendimento do que são Arrays, e como manipulá-los, recomendo a leitura deste artigo escrito pelo instrutor Neilton Seguins:

[**Para que serve um array?**](https://www.alura.com.br/artigos/javascript-para-que-serve-array)

# Consultando uma lista

Suponhamos que temos um array chamado listaItens e precisamos conferir a quantidade de objetos que ele possui.

<ul class="lista">

<div class="item">Laranja</div>

<div class="item">Vermelho</div>

<div class="item">Branco</div>

<div class="item">Laranja</div>

<div class="item">Preto</div>

</ul>

Das opções abaixo, qual retorna corretamente esta informação?

Top of Form

* Alternativa correta



console.log(listaItens.length)

Alternativa correta! A forma mais apropriada de consultar a quantidade de itens em um array é utilizando a propriedade length.

Bottom of Form

# Consolidando o seu conhecimento

Chegou a hora de praticar! No desafio desta aula, temos uma lista de cores que serão utilizadas para pintar uma loja, porém o cliente informou que não deseja mais pintar com a cor "vermelha". Retire esta cor da lista utilizando um método do JavaScript. Utilize a lista abaixo como base:

var lista = ["Laranja", "Vermelho", "Branco", "Amarelo", "Rosa"];

Como fazer:

* Abra o inspecionar em qualquer página do navegador;
* Na aba “console” do inspecionar passe a lista de cores;
* Aplique o método para remover a cor “Vermelho”;
* Utilize console.log(lista); ao final para conferir o resultado.

### Opinião do instrutor

var lista = ["Laranja", "Vermelho", "Branco", "Amarelo", "Rosa"];

lista.splice(1,1)

console.log(lista);

Ao utilizar o método splice(), o primeiro parâmetro a ser informado é a posição do item a ser removido, e o segundo parâmetro informa quantos itens serão removidos a partir de sua posição.

# Array.prototype.splice()

## [Resumo](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array/splice#resumo)

O método **splice()** altera o conteúdo de uma lista, adicionando novos elementos enquanto remove elementos antigos.

# O que aprendemos?

## Nessa aula:

* Definimos o que é Array e como manipular seus objetos;
* Como utilizar arrow functions;
* Como utilizar o método splice() para remover itens.

**Alterando o elemento**

Temos nosso componente da calculadora funcionando de uma forma dinâmica. Agora, é possível adicionar e subtrair todos os itens do Robotroon 2000. Porém, quando adicionamos o Braço, ele funciona normalmente, mas quando adicionamos uma Blindagem, um Núcleo, Perno ou Foguete, ele está também adicionando um Braço!

Em nosso código, nós buscamos somente o elemento com identificador do braço e é só ele que estamos mexendo. Já deixamos o nosso controlador dinâmico buscando os elementos que estão sendo clicados, mas isso não está refletido no elemento em que o código é aplicado, em que a soma e a subtração são aplicadas.

Em nosso html temos o botão de controle de ajuste negativo e positivo.

<div class="controle">

<buttom class="controle-ajuste">-</buttom>

<input type="text" class="controle-contador" value="00" id="braco">

<buttom class="controle-ajuste">+</buttom>

Aqui eu já retirei os identificadores, então podemos tirar nosso somar e subtrair do nosso código java.

const braco = document.querySelecto("#braco")

cont controle = document.querySelectorAll(".controle-ajuste")

controle.forEach( (elemento) => {

elemento.addEventListener('click', (evento) => {

manipulaDados(evento.target.textContent)

})

})

function manipulaDados(operacao) {

if(operacao === "-") {

braco.value = parseInt(braco.value) - 1

} else {

braco.value = parseInt(braco.value) + 1

}

}

Em nosso html, o buttom e o input são irmãos, mas não sabemos qual vem antes e qual vem depois. Em compensação, sabemos que todos eles são filhos do controle. Na árvore do nosso html, todos eles são filhos da classe controle, dessa div.

Isso é muito bom porque no JavaScript a navegação na árvore do html, interagir com o DOM é fácil, nós já vimos. Dentro de uma operação e de um evento conseguimos mapear o que aconteceu. Se clicamos no menos, já vimos o texto desse elemnto, mas também podemos ver quem é o pai dele. A partir do momento que conhecemos o pai podemos interagir com o elemento filho.

Vamos fazer isso agora para vocÊs verem funcionando. Daremos um console.log() em evento.target, ou seja, no elemento que foi clicado. Porém, como queremos navegar pela árvore do html, vamos usar o parentNode porque queremos ir no pai, que é o parent.

controle.forEach( (elemento) => {

elemento.addEventListener('click', (evento) => {

manipulaDados(evento.target.textContent)

console.log(evento.target.parentNode)

})

})

Agora quando clicamos, vamos direto na div de controle certa! Como nós conhecemos a div controle, só precidamos alterar a peça. Vamos trazer essa busca do braço aqui para dentro. Mas em vez de braco queremos procurar a peca, vamos deixar esse código mais genérico. Com "Ctrl + D" conseguimos trocar todas os itens de uma só vez.

function manipulaDados(operacao) {

const peca = document.querySelector("#braco")

if(operacao === "-") {

peca.value = parseInt(braco.value) - 1

} else {

peca.value = parseInt(braco.value) + 1

}

}

Temos esse elemento dinâmico, portanto, não vamos mais buscar por braco e sim por controle-contador.

function manipulaDados(operacao) {

const peca = document.querySelector("#controle-contador")

if(operacao === "-") {

peca.value = parseInt(braco.value) - 1

} else {

peca.value = parseInt(braco.value) + 1

}

}

Porém, não queremos buscar isos no documento inteiro senão vamos encontrar cinco controle-contador, queremos buscar o controle-contador só do controle, do elemento que foi clicado. Para isso, é preciso passar o elemento como parâmetro para função manipulaDados().

controle.forEach( (elemento) => {

elemento.addEventListener('click', (evento) => {

manipulaDados(evento.target.textContent, evento.target.parentNode)

})

})

Agora estamos passando dois blocos de informação para manipulaDados(), um é o operacao e o segundo vamos chamar de controle, que é o pai do elemento clicado.

function manipulaDados(operacao, controle) {

const peca = document.querySelector("#controle-contador")

if(operacao === "-") {

peca.value = parseInt(braco.value) - 1

} else {

peca.value = parseInt(braco.value) + 1

}

}

O querySelector() não será mais no documento inteiro, será apenas em controle. Toda vez que clicarmos no controle, vamos buscar o controle-contador e adicionar ou subtrair um item.

function manipulaDados(operacao, controle) {

const peca = controle.querySelector("#controle-contador")

if(operacao === "-") {

peca.value = parseInt(braco.value) - 1

} else {

peca.value = parseInt(braco.value) + 1

}

}

Recarregando o navegador podemos ver que a soma e a subtração funcionam em todos os itens do montador do Robotron! Agora que já podemos ter o robô pronto para batalha, precisamos resolver um detalhe muito importante porque esse código não está muito dinâmico e robusto!

Imaginem que por uma decisão de design em vez de - escrevêssemos menos e em vez de + escrevêssemos mais: toda nossa lógica que está procurando pelo sinal de menos ou pelo sinal de mais vai quebrar! Isso não acontece apenas com texto, também aconteceria se mudássemos o nome da classe ou adicionássemos uma outra classe.

Temos uma ferramenta chamada **\*Data Attributes** para resolver isso, mas esse é o assunto do próximo vídeo!

Código referente a aula acima:

*const* controle = document.querySelectorAll(".controle-ajuste") *//const que irá pegar todos os elemento que possuem a classe de ajuste de controle responsavel por conter o sinais de + e -*

controle.forEach( (*elemento*) => {  *//função que vai varrer o array de elementos que para cada elemento dentro irá executar algo*

    elemento.addEventListener("click", (*evento*) => { *//Para cada elemento desses que for clicado*

        manipulaDados(evento.target.textContent, evento.target.parentNode)  *//executa a minha função manipulaDados e pega o evento de click que está ocorrendo em x alvo de acordo com o conteudo de texto dentro dessa classe executa a manipulaDados no caso tem os sinais + e -*

    })  *//evento.target.parentNode faz com que toda a vez que eu clique em um target eu busque o controle-contador daquele elemento buscando o pai dele o a class controle*

})

*function* manipulaDados(*operacao*, *controle*) { *//Função que adiciona valores , recebe como parametro operacao*

*const* peca = controle.querySelector(".controle-contador") *//busca o controle-contador dentro da classe controle que o pai dos elementos*

    if (operacao === "+") { *//Se operacao for igual ao sinal "+" que e o conteudo de texto que tem dentro das classes .controle-ajuste adiciona + 1*

    peca.value = parseInt(peca.value) + 1

} else {   *//Se operacao não for igual ao sinal "+" quer dizer a outro sinal ou algo diminue -1*

    peca.value = parseInt(peca.value) - 1

    }

}

**Código à prova de falhas**

Temos nosso código dinâmico do montador de robôs! Agora temos os recursos para montadar toda as peças do Robotron 2000, porém, existe um problema muito grave, porque nosso CSS está acoplado ao nosso JavaScript. Por sorte, toda vez que usamos JavaScript para interagir com os elementos, podemos usar os **Datta Attributes**, os **atributos de dados**.

Vamos começar apagando o id="braco" do nosso html e modicar o controle-ajuste negativo. Para fazer isso, vamos criar um data attribute, vamos colocar data-controle e dizer que o texto é -.

<button class="controle-ajuste" data-controle="-">-<</button>

Criamos uma propriedade! Assim como a classe já existe, criamos uma propriedade do tipo data com o nome "controle" e o valor negativo. Vamos fazer isso para todos os elementos do html e também para o controle positivo.

Agora no JavaScript em vez de buscar a classe controle-ajuste vamos buscar pelo data attributes usando o colchetes [].

cont controle = document.querySelector("[controle-sjuste]")

O código continua buscando o elementos! Porém, colocamos um valor no data-controle então não precisamos buscar o texto do elemento, o texto pode mudar a vontade porque estamos desaclopando o JavaScript do código HTML. Agora nós buscaremos o elemento data.

controle.forEach( (elemento) => {

elemento.addEventListener('click', (evento) => {

manipulaDados(evento.target.dataset.controle, evento.target.parentNode)

})

})

Assim passamos para operacao o valor do controle, assim estruturamos melhor o código. Mas continuamos pecando porque estamos buscando pela classe do controle-contador. Então, vamos inventar um novo dataset.

Vamos colocar data e inventar um novo nome, não quero mais controle agora eu quero contador.

<div class="controle">

<button class="controle-ajuste" data-controle="-">-</button>

<input type="text" class="controle-contador" value="00" data-contador>

<button class="controle-ajute data-controle="+">+</button>

</div>

Reparem que não colocamos ele igual a nenhum valor, só colocamos o nome. Se não quisermos, não precisamos atribuir um valor a essa propriedade.

Agora em no código java vez de buscarmos por controle-contador, buscaremos por data-contador.

function manipulaDados(operacao, controle) {

const peca = controle.querySelector("[data-contador]")

if(operacao === "-") {

peca.value = parseInd(peca.value) - 1

} else {

peca.value = parseInt(peca.value) + 1

}

}

Temos agora nosso contador de peças, nosso montador do Robotron extremamente dinâmico! Entretanto, não estamos fazendo o cálculo para saber se esse é um bom robô para a batalha, nós queremos atualizar força, energia, poder e velocidade. Para fazer isso é preciso manipular os dados da nossa calculadora de energia.

Queremos que um elemento manipulado altere um grupo de elementos, mas isso é assunto para a próxima aula! Te vejo lá!

Código referente a aula acima:

HTML:

<div *class*="controle">

                            <*buttom* *class*="controle-ajuste" *data-controle*="-">-</*buttom*>

                            <input *type*="text" *class*="controle-contador" *value*="00" *data-contador*>

                            <*buttom* *class*="controle-ajuste" *data-controle*="+">+</*buttom*>

                        </div>

                    </div>

                    <hr *class*="linha">

                    <div *class*="peca">

                        <label *for*="" *class*="peca-titulo">Blindagem</label>

                        <div *class*="controle">

                            <*buttom* *class*="controle-ajuste"*data-controle*="-">-</*buttom*>

                            <input *type*="text" *class*="controle-contador" *value*="00" *data-contador*>

                            <*buttom* *class*="controle-ajuste" *data-controle*="+">+</*buttom*>

                        </div>

                    </div>

                    <hr *class*="linha">

                    <div *class*="peca">

                        <label *for*="" *class*="peca-titulo">Núcleos</label>

                        <div *class*="controle">

                            <*buttom* *class*="controle-ajuste"*data-controle*="-">-</*buttom*>

                            <input *type*="text" *class*="controle-contador" *value*="00" *data-contador*>

                            <*buttom* *class*="controle-ajuste" *data-controle*="+">+</*buttom*>

                        </div>

                    </div>

                    <hr *class*="linha">

                    <div *class*="peca">

                        <label *for*="" *class*="peca-titulo">Pernas</label>

                        <div *class*="controle">

                            <*buttom* *class*="controle-ajuste"*data-controle*="-">-</*buttom*>

                            <input *type*="text" *class*="controle-contador" *value*="00" *data-contador*>

                            <*buttom* *class*="controle-ajuste" *data-controle*="+">+</*buttom*>

                        </div>

                    </div>

                    <hr *class*="linha">

                    <div *class*="peca">

                        <label *for*="" *class*="peca-titulo">Foguetes</label>

                        <div *class*="controle">

                            <*buttom* *class*="controle-ajuste"*data-controle*="-">-</*buttom*>

                            <input *type*="text" *class*="controle-contador" *value*="00" *data-contador*>

                            <*buttom* *class*="controle-ajuste" *data-controle*="+">+</*buttom*>

                        </div>

JavaScript:

*const* controle = document.querySelectorAll("[data-controle]") *//const que irá pegar todos os elemento que possuem o data atributos de controle responsavel por conter o sinais de + e -*

controle.forEach( (*elemento*) => {  *//função que vai varrer o array de elementos que para cada elemento dentro irá executar algo*

    elemento.addEventListener("click", (*evento*) => { *//Para cada elemento desses que for clicado*

        manipulaDados(evento.target.dataset.controle, evento.target.parentNode)  *//executa a minha função manipulaDados e pega o evento de click que está ocorrendo em x alvo de acordo com o conteudo de texto dentro dessa classe executa a manipulaDados no caso tem os sinais + e -*

    })  *//evento.target.parentNode faz com que toda a vez que eu clique em um target eu busque o controle-contador daquele elemento buscando o pai dele o a class controle*

})

*function* manipulaDados(*operacao*, *controle*) { *//Função que adiciona valores , recebe como parametro operacao*

*const* peca = controle.querySelector("[data-contador]") *//busca o controle de cada elemento que possui o data-contador*

    if (operacao === "+") { *//Se operacao for igual ao sinal "+" que e o conteudo de texto que tem dentro das classes .controle-ajuste adiciona + 1*

    peca.value = parseInt(peca.value) + 1

} else {   *//Se operacao não for igual ao sinal "+" quer dizer a outro sinal ou algo diminue -1*

    peca.value = parseInt(peca.value) - 1

    }

}

# Para saber mais: data-attribute

Data-attributes são utilizados para guardar valores em elementos HTML. Esses valores podem ser manipulados através do JavaScript. Também é possível estilizar elementos HTML com CSS referenciando o seu data-attribute. Essa funcionalidade é bem recente no mundo do desenvolvimento, sendo lançada na última versão do HTML(HTML5).

Data-attributes não devem ser utilizados para dados visíveis, pois tecnologias de acessibilidade podem não identificar seus valores.

Sua estrutura é dividida em duas partes:

* A primeira parte é o identificador deste dado, que consiste em data- + o nome de sua escolha que melhor identifica o tipo de dado a ser atribuído.
* A segunda parte é o valor atribuído à este data-attribute, este valor deve sempre estar entre aspas “”, e dessa maneira é lido como uma string.

Segue exemplo abaixo:

**HTML**

<h1>Lista de tintas:</h1>

<ul id="lista">

<li data-cor="laranja" data-tipo="tinta-exterior" onclick="mudaCores(this)" class="item">Tinta laranja</li>

<li data-cor="vermelho" data-tipo="tinta-interior" onclick="mudaCores(this)" class="item">Tinta vermelha</li>

<li data-cor="branco" data-tipo="tinta-interior" onclick="mudaCores(this)" class="item">Tinta branca</li>

<li data-cor="amarelo" data-tipo="tinta-exterior" onclick="mudaCores(this)" class="item">Tinta amarelo</li>

<li data-cor="rosa" data-tipo="tinta-exterior" onclick="mudaCores(this)" class="item">Tinta rosa</li>

</ul>

No JavaScript, podemos criar uma função que recebe esses atributos, permitindo manipulá-los:

**JS**

function mudaCores(elementos){

var cores = elementos.getAttribute("data-cor");

var tipoElemento = elementos.getAttribute("data-tipo");

}

O prefixo data- não é obrigatório para definir um data-attribute personalizado, podemos alterar o código para a seguinte forma:

**HTML**

<h1>Lista de tintas:</h1>

<ul id="lista">

<li cor="laranja" tipo="tinta-exterior" onclick="mudaCores(this)" class="item">Tinta laranja</li>

<li cor="vermelho" tipo="tinta-interior" onclick="mudaCores(this)" class="item">Tinta vermelha</li>

<li cor="branco" tipo="tinta-interior" onclick="mudaCores(this)" class="item">Tinta branca</li>

<li cor="amarelo" tipo="tinta-exterior" onclick="mudaCores(this)" class="item">Tinta amarelo</li>

<li cor="rosa" tipo="tinta-exterior" onclick="mudaCores(this)" class="item">Tinta rosa</li>

</ul>

**JS**

function mudaCores(elementos){

var cores = elementos.getAttribute("cor");

var tipoElemento = elementos.getAttribute("tipo");}

# Definindo data-attributes

Tendo em vista o conteúdo abordado sobre os data-attributes utilizados por pessoas desenvolvedoras, desde sua estreia a partir do HTML5.

Assinale abaixo a alternativa que contenha informações corretas acerca dos data-attributes.

Top of Form

* Alternativa correta



data-attributes permitem armazenar informações extras em elementos HTML. Sua utilização não é indicada para dados visíveis, pois tecnologias de acessibilidade podem não ler seus valores. Seus dados podem ser acessados e manipulados utilizando JavaScript, ou também para estilizar elementos HTML no CSS.

Alternativa correta!

Bottom of Form

# Consolidando o seu conhecimento

Apenas com teoria não é possível aprender código, então, vamos exercitar a utilização do data-attribute em funções neste desafio. Temos uma lista de cores disponibilizada no código abaixo. Você irá criar um botão que, ao ser clicado, faz aparecer esta lista, e também criará um segundo botão que, ao ser clicado, faz com que a lista de cores volte a sumir.

Você pode chegar ao mesmo resultado de diferentes formas. Então, fique à vontade para compartilhar a sua solução para este desafio no Fórum!

Como fazer:

* Utilize os códigos HTML e CSS disponibilizados;
* Crie o arquivo de código JS;
* Referencie o arquivo JS no HTML.

**HTML**

<!DOCTYPE html>

<html lang="en" >

<head>

<meta charset="UTF-8">

<title>Manipulando o DOM - Aula 3</title>

<link rel="stylesheet" href="./style.css">

</head>

<body>

<button onclick="mostraTintas(this)">Aluratintas em estoque:</button>

<ul class="lista">

<li cor="laranja" tipo="tinta-exterior" class="item">Tinta laranja</li>

<li cor="vermelho" tipo="tinta-interior" class="item">Tinta vermelha</li>

<li cor="branco" tipo="tinta-interior" class="item">Tinta branca</li>

<li cor="amarelo" tipo="tinta-exterior" class="item">Tinta amarela</li>

<li cor="rosa" tipo="tinta-exterior" class="item">Tinta rosa</li>

<li cor="preto" tipo="tinta-exterior" class="item">Tinta preta</li>

</ul>

<a href="https://alura.com.br/" target="\_blank"><img src="https://www.alura.com.br/assets/img/home/alura-logo.svg" alt="" class="alura-logo"></a>

</body>

</html>

**CSS**

\* {

margin: 0;

padding: 0;

}

body {

font-family: "Roboto Mono", monospace;

min-height: 400px;

min-width: 450px;

background-size: 80vh;

color: rgb(0, 0, 0);

background-image: url("https://www.uniabeu.edu.br/wp-content/uploads/2020/03/fundo-lilas.png");

background-size: cover;

background-repeat: no-repeat;

height: 100vh;

font-size: 24px;

font-weight: bold;

display: flex;

align-items: center;

flex-direction: column;

}

button {

font-size: 24px;

font-family: "Futura Lt BT", sans-serif;

background-color: black;

background-repeat: no-repeat;

cursor: pointer;

overflow: hidden;

outline: none;

padding: 8px 20px 8px 20px;

color: white;

box-shadow: 0px 0px 5px gray;

border-radius: 5px 5px 0 0;

border: none;

transition: 500ms;

opacity: 0.8;

margin: 30px 0;

}

button:hover {

color: white;

background-color: gray;

}

button:active {

color: black;

background-color: white;

}

.lista {

padding: 20px;

list-style-type: none;

border: 4mm ridge rgba(234, 122, 11, 0.992);

}

.alura-logo {

width: 150px;

position: absolute;

top: 2%;

right: 2%;

}

[cor="laranja"] {

color: orange;

}

[cor="vermelho"] {

color: red;

}

[cor="branco"] {

color: white;

}

[cor="amarelo"] {

color: yellow;

}

[cor="rosa"] {

color: pink;

}

[cor="preto"] {

color: black;

}

VER OPINIÃO DO INSTRUTOR

### Opinião do instrutor

const lista = document.querySelector("ul");

document.querySelector("#botao").addEventListener("click", () => {

lista.setAttribute("data-lista", "mostrar");

});

document.querySelector(".close").addEventListener("click", () => {

lista.setAttribute("data-lista", "esconder");

});

Minha maneira de fazer:

*//Cria a constante lista, pegando a classe da <ul>*

*const* lista = document.querySelector(".lista")

*const* botaoOn = document.querySelector(".botaoOn")

*const* botaoOff = document.querySelector(".botaoOff")

*function* mostrarLista () {

    botaoOn.onclick = *function* () {

        lista.style.display = "block";

    }

}

*function* esconderLista () {

    botaoOff.onclick = *function* () {

        lista.style.display = "none";

    }

}

esconderLista();

mostrarLista();

Outra maneira interessante de fazer.

//Cria a constante lista, pegando a classe da <ul>

const lista = document.querySelector(".lista");

//Cria o botão "Mostrar lista", colocando um texto nele e colocando ele no body

let botaoMostrar = document.createElement("button");

let textoMostrar = document.createTextNode("Mostrar lista");

botaoMostrar.appendChild(textoMostrar);

document.body.appendChild(botaoMostrar);

//Cria o botão "Esconder lista", colocando um texto nele e colocando ele no body

let botaoEsconder = document.createElement("button");

let textoEsconder = document.createTextNode("Esconder lista");

botaoEsconder.appendChild(textoEsconder);

document.body.appendChild(botaoEsconder);

//A função mostrarLista é criada. No evento de clique no botão, a <ul> vira display block

function mostrarLista () {

botaoMostrar.onclick = function () {

lista.style.display = "block";

}

}

//A função esconderLista é criada. No evento de clique no botão, a <ul> vira display none

function esconderLista () {

botaoEsconder.onclick = function () {

lista.style.display = "none";

}

}

//Funções são executadas

esconderLista();

mostrarLista();

**Interagindo com objetos**

Nosso componente da engrenagem está funcionando perfeitamente e o atualizamos para usar o dataset e não ser dependente de alguma classe ou conteúdo que possa ser alterado. Agora, queremos que as estatísticas de nosso robô seja atualizada constantemente conforme formos montando Robotron. Ao adicionar um braço ou uma blindagem, precisamos saber se o poder, a velocidade e a força do robô está aumentando.

Em nossa lista de links, abrindo o repositóro do Robotron vocês vão encontrar um arquivo de estatisticas "estatistica.js" onde tem a declaração de uma const. Vamos copiar e colar em nosso código "main.js" adicionando a variável pecas.

const pecas = {

"bracos": {

"forca": 29,

"poder": 35,

"energia": -21,

"velocidade": -5

},

"blindagem": {

"forca": 41,

"poder": 20,

"energia": 0,

"velocidade": -20

},

"nucleos":{

"forca": 0,

"poder": 7,

"energia": 48,

"velocidade": -24

},

"pernas":{

"forca": 27,

"poder": 21,

"energia": -32,

"velocidade": 42

},

"foguetes":{

"forca": 0,

"poder": 28,

"energia": 0,

"velocidade": -2

}

}

Essa variável é um objeto que tem dentro dele cinco itens, braco, blindagem, nucleos, pernas e foguetes e dentro de cada um deles temos mais quatro declarações, um outro objeto, com mais quatro valores.

Se voltarmos no navegador e recarregarmos, podemos usar no console a variável pecas, teremos os cinco itens, cada um com seus valores.

pecas

(bracos: {-}, blindagens: {-}, nucleos:{-}, pernas: {-}, foguetes{-})

A estrutura de um obejto é muito interessante por ter a chave-valor que nos deixa navegar pelo objeto usando as referências. Com pecas.bracos conseguimos entrar dentro do item e ter todo o valor, que são mais quatro itens.

pecas.bracos

{forca: 29, poder: 35, energia: -21, velocidade: -5}

"energia": -21,

"forca": 29,

"poder": 35,

"velocidade": -5

[[Prototype]]: Objetc

Podemos, inclusive, navegar mais um nível e descobrir o valor númerico daquela estrutura.

pecas.braco.energia

-21

Também podemos navegar em um objeto com a busca [] e por meio dela encontrar o valor.

pecas["bracos"]

{forca: 29, poder: 35, energia: -21, velocidade: -5}COPIAR CÓDIGO

Essas são formas interessantes de navegarmos no objeto porque elas nos deixam buscar dados de uma forma dinâmica e para buscar esses dados de uma forma dinâmica também vamos precisar alterar o hmtl.

Nele temos o box das estatísticas com todos os valores e vamos adicionar em cada um desses textos um outros dataset que será o data-estatistica. O primeiro data-estatistica é forca, depois é poder, energia e velocidade, repectivamente.

<section class="box estatistica">

<div class="estatistica">

<p class="estatistica-titulo">Força</p>

<div class="estatistica-valor"

<p class="estatistica-numero" data-estatistica="forca">768</p>

</div>

</div>

<div class="estatistica"

<p class="estatistica-titulo">Poder</p>

<div class="estatistica-valor"

<p class="estatistica-numero" data-estatistica="poder">630</p>

</div>

</div>

<div class="estatistica"

<p class="estatistica-titulo">Energia</p>

<div class="estatistica-valor"

<p class="estatistica-numero" data-estatistica="energia">286</p>

</div>

</div>

<div class="estatistica"

<p class="estatistica-titulo">Velocidade</p>

<div class="estatistica-valor"

<p class="estatistica-numero" data-estatistica="velocidade">597</p>

</div>

</div>

<section>

Com o data-estatistica já podemos buscar esse elemento e alterar seu valor. Mas, além disso, precisamos passar para frente o que estamos mexendo. Aqui, não temos o conteúdo de braco sendo enviado para nenhum lugar, só temos o mais e menos. Portanto, em nosso buttom vamos criar mais um data attribute data-peca. O primeiro data-peca será igual a braco, o segundo igual a blindagem, depois nucleos, pernas e foguete, respectivamente.

Agora temos toda a estrutura que precisamos para fazer a lógica dessa nova funcionalidade, temos as peças, os valores. Agora precisamos reestruturar o nosso código para que ao clicarmos em um desses itens, chamarmos alguma nova função para atualizar as estatísticas. Mas isso é assunto para o próximo vídeo, te vejo lá!

Código referente a aula acima:

HTML:

<label *for*="" *class*="peca-titulo">Braços</label>

                        <div *class*="controle">

                            <*buttom* *class*="controle-ajuste" *data-controle*="-" *data-peca*="bracos">-</*buttom*>

                            <input *type*="text" *class*="controle-contador" *value*="00" *data-contador*>

                            <*buttom* *class*="controle-ajuste" *data-controle*="+" *data-peca*="bracos">+</*buttom*>

                        </div>

                    </div>

                    <hr *class*="linha">

                    <div *class*="peca">

                        <label *for*="" *class*="peca-titulo">Blindagem</label>

                        <div *class*="controle">

                            <*buttom* *class*="controle-ajuste"*data-controle*="-" *data-peca*="blindagem">-</*buttom*>

                            <input *type*="text" *class*="controle-contador" *value*="00" *data-contador*>

                            <*buttom* *class*="controle-ajuste" *data-controle*="+" *data-peca*="blindagem">+</*buttom*>

                        </div>

                    </div>

                    <hr *class*="linha">

                    <div *class*="peca">

                        <label *for*="" *class*="peca-titulo">Núcleos</label>

                        <div *class*="controle">

                            <*buttom* *class*="controle-ajuste"*data-controle*="-" *data-peca*="nucleos">-</*buttom*>

                            <input *type*="text" *class*="controle-contador" *value*="00" *data-contador*>

                            <*buttom* *class*="controle-ajuste" *data-controle*="+" *data-peca*="nucleos">+</*buttom*>

                        </div>

                    </div>

                    <hr *class*="linha">

                    <div *class*="peca">

                        <label *for*="" *class*="peca-titulo">Pernas</label>

                        <div *class*="controle">

                            <*buttom* *class*="controle-ajuste"*data-controle*="-" *data-peca*="pernas">-</*buttom*>

                            <input *type*="text" *class*="controle-contador" *value*="00" *data-contador*>

                            <*buttom* *class*="controle-ajuste" *data-controle*="+" *data-peca*="pernas">+</*buttom*>

                        </div>

                    </div>

                    <hr *class*="linha">

                    <div *class*="peca">

                        <label *for*="" *class*="peca-titulo">Foguetes</label>

                        <div *class*="controle">

                            <*buttom* *class*="controle-ajuste"*data-controle*="-" *data-peca*="foguestes">-</*buttom*>

                            <input *type*="text" *class*="controle-contador" *value*="00" *data-contador*>

                            <*buttom* *class*="controle-ajuste" *data-controle*="+" *data-peca*="foguestes">+</*buttom*>

                        </div>

                    </div>

                </div>

JavaScript:

*const* controle = document.querySelectorAll("[data-controle]") *//const que irá pegar todos os elemento que possuem o data atributos de controle responsavel por conter o sinais de + e -*

*const* pecas = {

    "bracos": {

        "forca": 29,

        "poder": 35,

        "energia": -21,

        "velocidade": -5

    },

    "blindagem": {

        "forca": 41,

        "poder": 20,

        "energia": 0,

        "velocidade": -20

    },

    "nucleos":{

        "forca": 0,

        "poder": 7,

        "energia": 48,

        "velocidade": -24

    },

    "pernas":{

        "forca": 27,

        "poder": 21,

        "energia": -32,

        "velocidade": 42

    },

    "foguetes":{

        "forca": 0,

        "poder": 28,

        "energia": 0,

        "velocidade": -2

    }

}

controle.forEach( (*elemento*) => {  *//função que vai varrer o array de elementos que para cada elemento dentro irá executar algo*

    elemento.addEventListener("click", (*evento*) => { *//Para cada elemento desses que for clicado*

        manipulaDados(evento.target.dataset.controle, evento.target.parentNode)  *//executa a minha função manipulaDados e pega o evento de click que está ocorrendo em x alvo de acordo com o conteudo de texto dentro dessa classe executa a manipulaDados no caso tem os sinais + e -*

    })  *//evento.target.parentNode faz com que toda a vez que eu clique em um target eu busque o controle-contador daquele elemento buscando o pai dele o a class controle*

})

*function* manipulaDados(*operacao*, *controle*) { *//Função que adiciona valores , recebe como parametro operacao*

*const* peca = controle.querySelector("[data-contador]") *//busca o controle de cada elemento que possui o data-contador*

    if (operacao === "+") { *//Se operacao for igual ao sinal "+" que e o conteudo de texto que tem dentro das classes .controle-ajuste adiciona + 1*

    peca.value = parseInt(peca.value) + 1

} else {   *//Se operacao não for igual ao sinal "+" quer dizer a outro sinal ou algo diminue -1*

    peca.value = parseInt(peca.value) - 1

    }

}