Java script

# Selecionando elemento na página

Para selecionar um elemento da página usa:

document.querySelector=(“. Nome da classe ou # Id do elemento”);

exemplo:

var tdaltura = document.querySelector(“.infoaltura”);

Neste caso é selecionado somente um elemento mesmo que existam outros elementos com a mesma classe

Para selecionar todos temos que fazer

Document.çquerySelectorAll(“.infoaltura”);

Neste caso é criado um array com todos os elmentos da classe

Depois de selecionado podemos acessar as propriedades por exemplo texto e trocar esse texto etc:

]Para isso usamos .textContent “”;

# Percorrendo array:

Para percorrer esse array, temos que fazer um for:

Var alturas = document.querySelectorAll(“.infoaltura”);

For (var i = 0; i < alturas.length; I++) {

Altura = alturas[i];

}

Por fim podemos fixar o numero de casas decimais usando to.lfixed(2);

# Melhorando visualização

No arquivo css, podemos adicionar uma classe exemplo

.paciente-invalido {

background-color: lightcoral;

}

Nesta classe estamos alterando a cor de um elemento caso este não seja calculado

No entanto no Java script temos que pedir para ele adicionar essa classe como se ele estivesse adicionando no HTML da seguinte forma:

paciente.classList.add(paciente-invalido");

usamos para isso a propriedade classList.add();

# Escutando eventos:

Podemos escutar os eventos de uma página ou seja quando o usuário clica, rola a página, passa o mouse etc.

Para isso usamos a função:

addEventListener("click", mostraMensagem);

Essa função recebe dois parâmetros o primeiro que é qual evento deve ser escutado nesse caso estamos usando o click e o segundo a função que deve ser chamada nesse caso estamos chamando a função mostramensagem.

Observação: a função mostra mensagem deve ser chamada sem os parênteses pois como ela não retorna nada o código que será executado se passado com parêntese será igual a nada.

Pode ser também criada uma função anônima ou seja, é uma função que é criada no segundo parâmetro específico para o evento.

Para escutar um evento, passamos o nome da variável que contem o elemento que desejamos ouvir . e a função como mostrado abaixo:

titulo.addEventListener("click", mostraMensagem);

Exemplo de função anônima com o evento

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

console.log("Olha só posso chamar uma função anônima.")

});

Observe que a função ainda possui as chaves ({}), mas não recebe um nome.

# Evitando comportamento padrão

Num formulário, a função padrão de um botão é ao ser clicado, envia o formulário e recarrega a página.

No entanto como não queremos enviar o formulário devemos mudar o padrão desse botão adicionando a função

event.preventDefault();

para utilizá-la, receberemos como segundo parâmetro a função anônima, o event:

var botaoAdicionar = document.querySelector("#adicionar-paciente");

botaoAdicionar.addEventListener("click", function(event) {

event.preventDefault();

console.log("Oi eu sou o botao e fui clicado");

});

# Adicionando item na tabela após cflicar no botão

Primeiramente vamos trocar a função anônima que exibe a mensagem de que o botão foi clicado pela função que iremos criar para adicionar itens na tabela.

É necessário selecionar o formulárioi e faremos isso através do Id

var form = document.querySelector("#form-adiciona");

Através do formulário, temos acesso aos seus inputs, acessando através da sua propriedade name. É possível vermos no HTML que todos os inputs possuem uma propriedade name, bastando utilizá-lo para termos acesso ao input:

var nome = form.nome;

Mas não estamos interessados no campo em si, no input, e sim no seu valor. Para acessar o valor de um campo em JavaScript, utilizaremos a propriedade value:

var nome = form.nome.value;

Sabendo disso basta capturar o restante dos campos.

# Criando novo elemento

Até o momento vimos como selecionar e trocar textos dos elementos, agora para criar um elemento, devemos usar a função createElement, exemplo:

var pacienteTr = document.createElement("tr");

Veja que criamos uma variável, atribuímos a ela a função createElement e dentro das aspas passamos qual elemento queremos criar, no nosso caso uma TR.

Também criaremos as TDS.

var nomeTd = document.createElement("td");

Porém falta colocarmos o conteúdo que capturamos nos campos dentro da TD.

Para isso dissemos que a variável nomeTd ponto propriedade que troca conteúdo textContent recebe a variável nome que contém o valor capturado no campo nome.

nomeTd.textContent = nome;

faremos isso para todos os outros campos.

# Utilizando o JavaScript para adicionar um elemento dentro de outro

Agora devemos adicionar cada td dentro da tr. No JavaScript existe uma função chamada appendChild. Ela adiciona como filho o elemento passado a ela como parâmetro. Logo, vamos chamar essa função na tr, passando a td como parâmetro, uma por uma:

pacienteTr.appendChild(nomeTd);

Veja que lá em cima criamos a TR paciente e agora aqui em baixo estamos colocando dentro da Tr todas as Tds.Por fim vamos selecionar a tabela através do seu id e vamos adicionar a Tr nela através da mesma função.

var tabela = document.querySelector("#tabela-pacientes");

tabela.appendChild(pacienteTr);

# Criando objeto

Para criar um objeto em java script criamos uma variável colocamos igual e dentro da chave passamos as propriedades separados por vírgula.

Assim conseguimos obter as propriedades cujo o valor pegamos do formulário só que ao invés de passarmos com igual, colocamos dois pontos.

Exemplo:

var paciente = {

nome: form.nome.value

No exemplo a cima estamos criando o objeto paciente e a propriedade nome estamos pegando do campo nome do formulário.

Podemos acessar as propriedades fazendo paciente.nome, etc.

Para melhorar nosso código criaremos a função que obtém dados do formulário:

// form.js

function obtemPacienteDoFormulario(form) {

var paciente = {

nome: form.nome.value,

peso: form.peso.value,

altura: form.altura.value,

gordura: form.gordura.value,

imc: calculaImc(form.peso.value, form.altura.value)

}

return paciente;

}

Essa função pega todos os dados do formulário

# Melhorando o código

Iremos criar uma função monta tr e uma outra monta td além disso iremos adicionar as classes que já veio do html.

Na função monta td receberemos como parâmetro dado e classe que deverá ser passada toda vez que a função monta tr chamar a função monta td.

E para enxugar ainda mais o código faremos isso na função appendchild

O código ficará assim:

//função para montar td

function montaTd(dado,classe){

var td = document.createElement("td");

td.textContent = dado;

td.classList.add(classe);

return td;

}

//função para montar tr

function montaTr(paciente) {

//criando Tr

var pacienteTr = document.createElement("tr");

//adicionando classe paciente do html antigo

pacienteTr.classList.add("paciente");

//criando td chamando função montaTd

pacienteTr.appendChild(montaTd(paciente.nome, "info-nome"));

pacienteTr.appendChild(montaTd(paciente.peso, "info-peso"));

pacienteTr.appendChild(montaTd(paciente.altura, "info-altura"));

pacienteTr.appendChild(montaTd(paciente.gordura, "info-gordura"));

pacienteTr.appendChild(montaTd(paciente.imc, "info-imc"));

return pacienteTr;

}

RTambém iremos usar uma propriedade do form para limpar o formulário toda vez que clicarmos no botão adicionar.

# Melhorando parte de validação de peso e altura

Quando adicionamos um paciente na tabela este não é validado peso e altura igual ao que fazemos com os já existentes.

Por isso criamos a função de validação de peso e altura no form.js

No calculaimc.js ao invés de iniciarmos as variáveis pesoevalido e alturaevalida com true, chamamos a função que criamos então fica:

var pesoEhValido = validaPeso(peso);

var alturaEhValida = validaAltura(altura);

Além disso, caso o paciente tenha algum valor inválido, queremos que ele não seja adicionado na tabela. Então devemos sair da função caso a condição do if seja válida. Ou seja, se o código entrar no if, exibiremos a mensagem e retornaremos a função, adicionando-se o return vazio:

Este if é adicionado após a função montar.

var form = document.querySelector("#form-adiciona");

var paciente = obtemPacienteDoFormulario(form);

var pacienteTr = montaTr(paciente);

if (!validaPaciente(paciente)) {

console.log("Paciente inválido");

return;

}

Função que irá validar os campos caso algum não seja preenchido na hora que enviar o formulário, este apresentará erro

Está sendo criado um array e impresso esse array com os erros adicionados para cada campo.

# Função para validar erros

Vamos agora criar a função que irá validar o usuário quando submetermos o form. O nome desta serávalidaPaciente, que receberá o objeto paciente e retornará um array de erros:

// form.js

function validaPaciente(paciente) {

var erros = [];

if (paciente.nome.length == 0) {

erros.push("O nome não pode ser em branco");

}

if (paciente.gordura.length == 0) {

erros.push("A gordura não pode ser em branco");

}

if (paciente.peso.length == 0) {

erros.push("O peso não pode ser em branco");

}

if (paciente.altura.length == 0) {

erros.push("A altura não pode ser em branco");

}

if (!validaPeso(paciente.peso)) {

erros.push("Peso é inválido");

}

# Removendo componentes da tabela

Para resolver o problema da remoção de pacientes, vamos nos aproveitar de uma característica do JavaScript chamada Event Bubbling, ou "borbulhamento" de eventos. Quando escutamos um evento no JavaScript, ele na verdade não acontece só no dono do evento (no nosso caso, na linha do paciente), ele acontece também no pai do paciente, no pai do pai do paciente, e assim vai subindo.

Na nossa estrutura, ao darmos um duplo clique na <tr> do paciente, o pai (<tbody>) também escuta o evento, assim com a tag <table>, até chegar no <body>.

<thead>

<tr>

<th>Nome</th>

<th>Peso</th>

<th>Altura</th>

<th>Gordura Corporal(%)</th>

<th>IMC</th>

</tr>

</thead>

<tbody id="tabela-pacientes">

<tr class="paciente" id="primeiro-paciente">

<td class="info-nome">Paulo</td>

<td class="info-peso">100</td>

<td class="info-altura">-2.00</td>

<td class="info-gordura">10</td>

<td class="info-imc">0</td>

</tr>

</body>

Para solucionar o nosso problema selecionaremos a table visto que ela é o pai da tabela

var tabela = document.querySelector("table");

usaremos a funç~~]ao evgent pois ela tem a porpiredade .target que nos diz qual é o alvo que foi clicado ou seja qual a td que foi clicada.

tabela.addEventListener("dblclick", function(event) {

event.target

e aqui não podemos usar o remove se não ele irá remover o alvo que foi clicado então temos que usar a propriedade parentNode.

Que seleciona o pai de quem foi clicado então pedimos para remover.

Exemplo:

var tabela = document.querySelector("table");

tabela.addEventListener("dblclick", function(event) {

event.target.parentNode.remove();

});

# Efeito visual antes de deletar

Para que a linha não suma instantaneamente ao clicarmos ou seja, para que apareça um efeito visual podemos colocar este efeito criando uma classe no css

.fadeOut {

opacity: 0;

transition: 0.5s;

}

var tabela = document.querySelector("#tabela-pacientes");

tabela.addEventListener("dblclick", function(event) {

event.target.parentNode.classList.add("fadeOut");

setTimeout(function() {

event.target.parentNode.remove();

}, 500);

});

Além de adicionarmos a classe que adicioanamos no css usamos a função setTimeout que pede para o java script aguardar tempo em milissegundos antes de executar a próxima função.

# Solução para remover linhas da tabela pressionando enter

Será necessário mexer no html conforme exemplo abaixo

<html>

<body>

<table>

<tr tabindex="0">

<td>col11</td>

<td>col12</td>

<td>col13</td>

</tr>

<tr tabindex="0">

<td>col21</td>

<td>col22</td>

<td>col23</td>

</tr>

</table>

<script>

let linhasColuna = document.querySelectorAll("tr");

for (let i = 0; i < linhasColuna.length; i++) {

linhasColuna[i].addEventListener("keypress", function (event) {

if (event.keyCode == 13) {

event.target.remove();

}

});

}

</script>

</body>

</html>

Observação: O valor 0 da propriedade tabindex significa que o elemento irá ser acessível via tecla tab do teclado na ordem em que ele aparece na tela, levando em consideração os demais elementos acessíveis que vierem antes dele. Ou seja, se antes dessa tabela houvessem 3 elementos acessíveis via tecla tab, para chegar na primeira linha seria necessário apertar a tecla tab 4 vezes.

# Filtrando pacientes da tabela

No projeto iremos implementar no html o campo que iremos usar para fazer o filtro

Além disso serão implementadas classe no css para deixar o campo maior.

Para começar iremos selecionar o campo que iremos usar para escrever a palavra ou letra correspondente que deverá ser buscado na tabela

var campoFiltro = document.querySelector("#filtrar-tabela");

Vamos adicionar o evento de imput que é responsável por detectar quando o usuário está digitando.

campoFiltro.addEventListener("input", function() {

No arquivo css criaremos a classe invisível e usaremos o display none que esconderá o conteúdo da tabela que não fizer parte do filtro.

.invisivel {

display: none;

}

Temos acesso que o que foi digitado pelo usuário pode ser acessado através do this.value, afinal o this neste caso é o <input> e podemos ter acesso ao seu contéudo com a propriedade .value. Se detectarmos algo escrito no campo, vamos esconder todos os pacientes da tabela colocando a classe .invisel, caso contrário, vamos remover a classe invisivel e exibi-los novamente:

var campoFiltro = document.querySelector("#filtrar-tabela");

campoFiltro.addEventListener("input", function() {

var pacientes = document.querySelectorAll(".paciente");

if (this.value.length > 0) {

for (var i = 0; i < pacientes.length; i++) {

var paciente = pacientes[i];

paciente.classList.add("invisivel");

}

} else {

for (var i = 0; i < pacientes.length; i++) {

var paciente = pacientes[i];

paciente.classList.remove("invisivel");

}

}

});

Esta é uma forma manual porém vamos implementar usando expressão regular

# Usando expressão regular

O primeiro passo é extrair todos os pacientes para comparar com o que foi digitado no campo filtro

Selecionando campo filtro colocando evento imput para escutar quando o usuário está digitando e na função anônima selecionando todos os pacientes da tabela.

var campoFiltro = document.querySelector("#filtrar-tabela");

campoFiltro.addEventListener("input", function() {

var pacientes = document.querySelectorAll(".paciente");

if (this.value.length > 0) {

for (var i = 0; i < pacientes.length; i++) {

var paciente = pacientes[i];

var tdNome = paciente.querySelector(".info-nome");

var nome = tdNome.textContent;

var expressao = new RegExp(this.value, "i");

//comparando se o que foi digitado bate com o que tem na tabela

if (expressao.test(nome)) {

paciente.classList.remove("invisivel"); } else {

paciente.classList.add("invisivel");

}

}

} else {

for (var i = 0; i < pacientes.length; i++) {

var paciente = pacientes[i];

paciente.classList.remove("invisivel");

}

}

});

Outra forma de comparar, sem usar expressão regular

Aprendemos a como filtrar, comparar letra a letra, utilizando expressões regulares, representado pelo fragmento de código abaixo:

var expressao = new RegExp(this.value,"i");

if (!expressao.test(nome)) {

paciente.classList.add("invisivel");

} else{

paciente.classList.remove("invisivel");

}

Mas há um modo de fazer essa comparação sem a necessidade de utilizar expressões regulares! Podemos utilizar a função substr, que recebe dois parâmetros, fazendo com que ela devolva parte da string, com o tamanho definido nos parâmetros. O primeiro parâmetro é o início, começando do 0 (que representa o primeiro caractere). O segundo parâmetro define o fim (exclusivo, mostramos até o penúltimo caractere). Por exemplo:

var string = "Alura";

var resultado = string.substr(1, 4);

Extraímos uma string que começa no segundo caractere (número 1) e termina no quarto caractere (número 3, índice anterior ao número 4). O valor da variável resultado é:

lura

Conhecendo essa função, pense em um modo de comparar o valor digitado com parte do nome, sem utilizar expressões regulares. Veja em seguida a resposta do instrutor.

VER OPINIÃO DO INSTRUTOR

Opinião do Instrutor

Configurações

Como o primeiro parâmetro é o inicio, e queremos comparar desde o início da string nome, vamos utilizar como início o valor 0, ou seja, sempre a partir do primeiro caractere. Mas qual é o fim? O fim é justamente o tamanho do valor digitado:

nome.substr(0, this.value.length);

Podemos guardar essa string em uma variável, e compará-la com o que está sendo digitado. Caso seja falso, adicionamos a classe invisivel, se não for, removemos-a:

var comparavel = nome.substr(0, this.value.length);

if (!(this.value == comparavel)) {

paciente.classList.add("invisivel");

} else{

paciente.classList.remove("invisivel");

}

Mas e a distinção entre letras maiúsculas e minúsculas? Nesse caso não temos distinção entre letras maiúsculas e minúsculas, mas para contornar isso, antes de compará-las, podemos colocar as duas strings em letras minúsculas, para efetuar a comparação entre elas em seguida:

var comparavel = nome.substr(0, this.value.length);

var comparavelMinusculo = comparavel.toLowerCase();

var valorDigitadoMinusculo = this.value.toLowerCase();

if (!(valorDigitadoMinusculo == comparavelMinusculo)) {

paciente.classList.add("invisivel");

} else{

paciente.classList.remove("invisivel");

}

Esta é uma alternativa de implementar a mesma funcionalidade sem expressão regular, porém temos que escrever mais e nos preocupar com mais detalhes! Fica ai esta opção para você guardar nos seus conhecimentos.

# Trazendo dados externos

Para começar, devemos selecionar o botão e escutar quando ele for clicadfo

var botaoAdicionar = document.querySelector("#buscar-pacientes");

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

//O XMLHttpRequest é um objeto do JS responsável por fazer requisições HTTP.

var xhr = new XMLHttpRequest();

//configurando para que ele use o método get e abra o endereço que queremos

xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");

//Escutando para saber se a página já foi carregada

xhr.addEventListener("load", function() {

//pegando o conteúdo da página

var resposta = xhr.responseText;

//fazendo o parce de json que é o conteúdo da página para um array

var pacientes = JSON.parse(resposta);

//fazendo um foreach

pacientes.forEach(function(paciente) {

//chamando função adicionapaciente na tabela que foi criada e também modificada função monta tr que está preso no botão de adicionar

adicionaPacienteNaTabela(paciente);

//enviando a requsição

xhr.send();

});

});

xhr.send();

});

xhr.send();

});

Podemos ainda usar

console.log(typeof pacientes);

para saber qual o tipo do objeto que está sendo retornado exemplo:

var botaoAdicionar = document.querySelector("#buscar-pacientes");

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

console.log("Buscando pacientes...");

var xhr = new XMLHttpRequest();

xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");

xhr.addEventListener("load", function(){

var resposta = xhr.responseText;

console.log(resposta);

console.log(typeof resposta);

var pacientes = JSON.parse(resposta);

console.log(pacientes);

console.log(typeof pacientes);

});

xhr.send();

});

No console, veremos que já temos um array, sendo que cada item do array é um objeto que representa o paciente, basta iterar por eles e adicionar cada um na tabela.