

Apostila para Introdução a Programação com JavaScript

Prof. Andrey A. Masiero

Table of Contents

COMEÇANDO COM JAVASCRIPT	3
VARIÁVEIS E OPERAÇÕES.....	6
LAÇOS DE REPETIÇÃO E FORMATAÇÃO DE ESTILO	13
EVENTOS, FORMULÁRIOS E CRIANDO ELEMENTOS.....	21
BOAS PRÁTICAS	31
VALIDAÇÃO DOS DADOS DO FORMULÁRIO	42
REMOÇÃO, DELEGAÇÃO E ANIMAÇÃO.....	52
FILTRANDO PACIENTES NA TABELA.....	57
AJAX	61

Começando com JavaScript

Nesse capítulo serão apresentados os primeiros passos para aprender a trabalhar com o JavaScript.

Por que aprender javascript?

Linguagem nativa da web. Utilizado para dar dinamismo e interatividade nas páginas da web. E por mais que podemos utilizar o html e o css para fazer algumas animações e validações, existem outras que não são possíveis sem o javascript.

Mas sua grande popularidade veio pelo node.js que trouxe a programação com javascript para o Back-end, permitindo que um desenvolvedor full-stack precisa de apenas uma linguagem para programar. Além disso, outros frameworks foram desenvolvidos como o Electron para o Desktop, o Johnny-Five para IoT e Robótica e o MongoDB para banco de dados.

Exibindo informações ao usuário (Olá Mundo)

Para que um javascript seja entendido pelo browser é necessário que seja envolvido por uma tag chamada de script. Existem duas maneiras de utilizar essa tag:

A primeira maneira é diretamente no body do html (página)

```
<script>
  // Seu código aqui
</script>
```

E a segunda é através de um arquivo externo e relacionando ele com a tag:

```
<script src="caminho/arquivo.js"></script>
```

Para exibir a mensagem para o usuário o javascript conta com duas funções, a primeira o alert:

```
alert("Escreva sua mensagem aqui!");
```

Resultado:

ducao-javascript/index.html

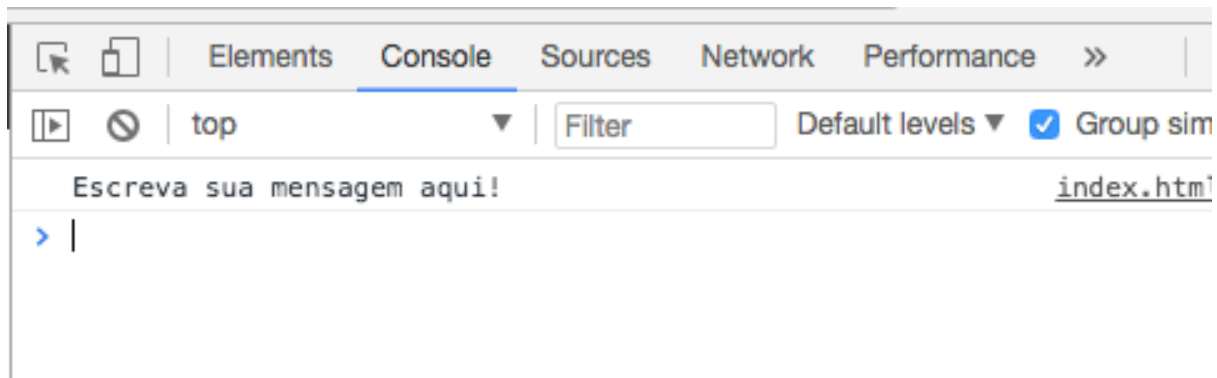
This page says

Escreva sua mensagem aqui!

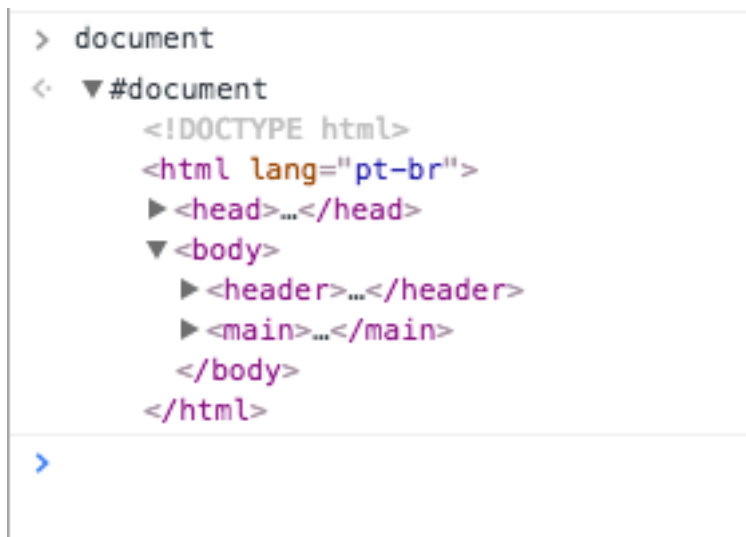
OK

Ou através do console do browser, com a função console.log:

```
console.log("Escreva sua mensagem aqui!");
```



Contudo, o intuito principal do javascript é manipular a página html, precisa-se então de um elemento que represente o html como objeto js. Esse elemento é conhecido como DOM (Document Object Model). No código a palavra chave para acessar a todo o conteúdo do javascript e o document.



O document é a ponte entre o HTML e o JavaScript. Tudo que for alterado nesse objeto, será alterado na exibição do HTML também.

Para pesquisar alguma informação no document, o JavaScript disponibiliza ao desenvolvedor uma função chamada `querySelector` (`query` => buscar, `selector` => seletor). E como ela funciona, simples passamos como parâmetro o nome da tag (seletor) entre aspas. Por exemplo, se eu quero selecionar o elemento `h1` da página, faço o seguinte código:

```
var elemento = document.querySelector("h1");  
> document.querySelector("h1");  
< <h1>Aparecida Nutrição</h1>
```

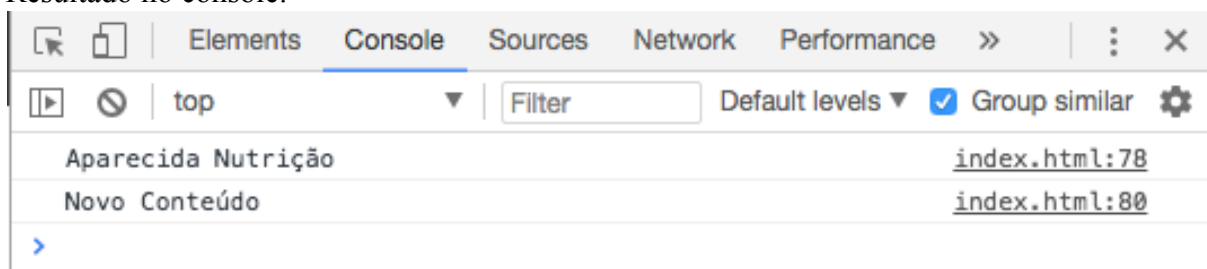
Essa função retorna todo o html do elemento `h1` contido na página.

Boa prática: Quando carregarmos o script da página, é importante que coloquemos ele como a última linha antes de fechar a tag body, pois assim garantirmos que o documento html está totalmente carregado.

Um DOM tem várias propriedades que auxiliam na manipulação do HTML. Uma delas é o `textContent` (Conteúdo de Texto). Com ela é possível manipular o conteúdo exibido para o usuário, o conteúdo que vai entre as tags do html.

```
var elemento = document.querySelector("h1");
console.log(elemento.textContent);
elemento.textContent = "Novo Conteúdo";
console.log(elemento.textContent);
```

Resultado no console:



No HTML:



Boa prática: Através da busca do `querySelector` a um nome de tag, podemos gerar um problema, pois se outra pessoa trocar a tag para melhorar o layout ou algo do tipo, a busca irá retornar como nula. Então é melhor utilizar outras informações que a função `querySelector` possibilita para nós, desenvolvedores. Como ela faz a busca pelo seletor, pode-se utilizar seletores como no CSS para buscar elementos como **.classe** e a **#identificação**. Use sempre algum seletor que irá representar melhor a informação que você deseja buscar no objeto HTML.

Boa prática: É importante separar os conteúdos das ferramentas em seus devidos e apropriados locais. Isso quer dizer que o código HTML fica no arquivo HTML, o código CSS no arquivo CSS e o código JavaScript no arquivo JavaScript. Depois de criar o arquivo, basta vincular ele à página HTML, como exemplo:

```
<script type="text/javascript" src="js/principal.js"></script>
```

Variáveis e Operações

O capítulo a seguir apresentará formas de armazenar conteúdos em variáveis e como realizar operações com conteúdos numéricos extraídos da página em exercício.

Agora que quero selecionar um elemento da tabela, e sabendo que existe a função `querySelector` que me retorna um elemento HTML a partir de algum seletor, seja o id, classe ou nome da tag, é necessário reconhecer no código algo que identifique de maneira exclusiva o paciente da primeira linha.

Para seguir com o exemplo, o objetivo é selecionar os valores de peso e altura, e calcular de maneira automática o valor do IMC de cada paciente.

O código da tabela atual é:

```
<table>
  <thead>
    <tr>
      <th>Nome</th>
      <th>Peso (kg) </th>
      <th>Altura (m) </th>
      <th>Gordura Corporal (%) </th>
      <th>IMC</th>
    </tr>
  </thead>
  <tbody id="tabela-pacientes">
    <tr class="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>

    <tr class="paciente" >
      <td class="info-nome">João</td>
      <td class="info-peso">80</td>
      <td class="info-altura">1.72</td>
      <td class="info-gordura">40</td>
      <td class="info-imc">0</td>
    </tr>

    <tr class="paciente" >
      <td class="info-nome">Erica</td>
      <td class="info-peso">54</td>
      <td class="info-altura">1.64</td>
      <td class="info-gordura">14</td>
      <td class="info-imc">0</td>
    </tr>
```

```

<tr class="paciente">
  <td class="info-nome">Douglas</td>
  <td class="info-peso">85</td>
  <td class="info-altura">1.73</td>
  <td class="info-gordura">24</td>
  <td class="info-imc">0</td>
</tr>
<tr class="paciente" >
  <td class="info-nome">Tatiana</td>
  <td class="info-peso">46</td>
  <td class="info-altura">1.55</td>
  <td class="info-gordura">19</td>
  <td class="info-imc">0</td>
</tr>
</tbody>
</table>

```

Existe apenas o seletor da classe, mas ele se repete nas demais linhas com paciente. Dessa maneira, é interessante determinar um id para a primeira linha da tabela de maneira que seja um identificador único para ela. Sendo assim, o código fica:

```

<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>

```

Então utilizando agora o meu javascript para buscar pelo id e solicitarmos para exibir no console, devemos ter exatamente o código html acima, sendo exibindo:

JavaScript:

```

var paciente = document.querySelector("#primeiro-paciente");
console.log(paciente);

```

Resultado no console:



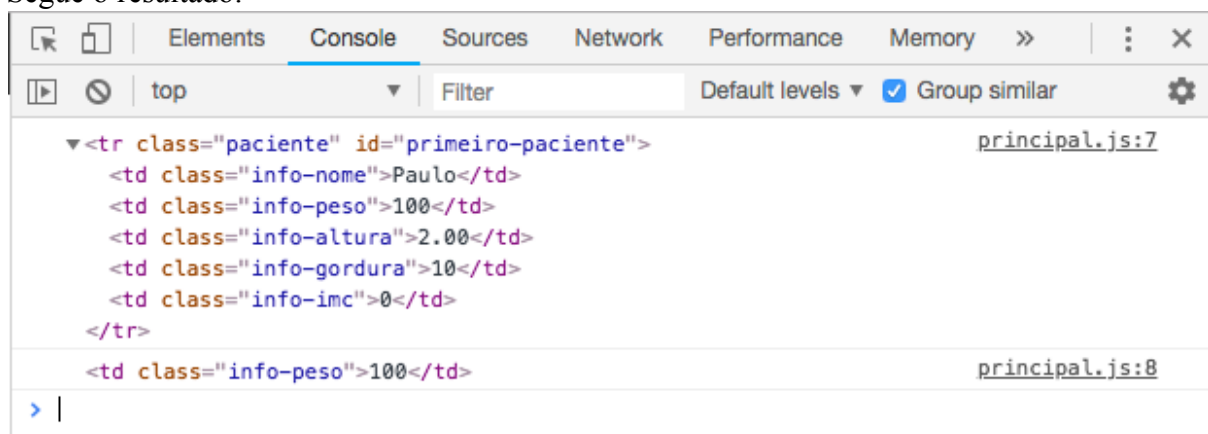
Maravilha, o paciente retorna com todo o conteúdo da tr que representa o primeiro paciente. Sendo o document a página html, e o paciente um trecho da página html, será que é possível utilizar o `querySelector` na variável paciente, que representa o elemento tr do html?

O td que tem o peso será utilizado como exemplo para este teste. O código JavaScript fica assim então:

```
var paciente = document.querySelector("#primeiro-paciente");  
var tdPeso = paciente.querySelector".info-peso"); // Busca  
pela classe que representa o td do Peso
```

```
console.log(paciente); // Exibe o tr do primeiro paciente  
console.log(tdPeso); // Exibe o td do peso para o primeiro  
paciente
```

Segue o resultado:



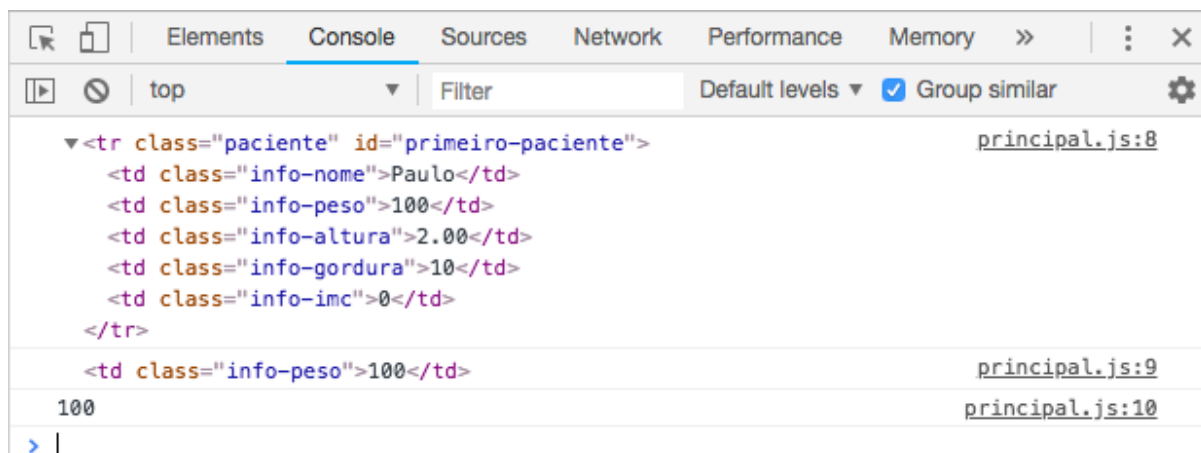
Porém o elemento td não resolve o problema para calcular o IMC. É necessário que apenas o valor do conteúdo em td seja selecionado e armazenado em uma variável.

O código então fica:

```
var paciente = document.querySelector("#primeiro-paciente");  
var tdPeso = paciente.querySelector".info-peso"); // Busca  
pela classe que representa o td do Peso  
var peso = tdPeso.textContent; // Conteúdo do td peso
```

```
console.log(paciente); // Exibe o tr do primeiro paciente  
console.log(tdPeso); // Exibe o td do peso para o primeiro  
paciente  
console.log(peso); // Exibe 100, o valor do peso do primeiro  
paciente
```

O resultado é:



Agora basta fazer o mesmo procedimento para a altura. E depois realizar o cálculo do IMC que é:

$$IMC = \frac{peso}{altura^2}$$

O meu código fica então fica:

```
var paciente = document.querySelector("#primeiro-paciente");

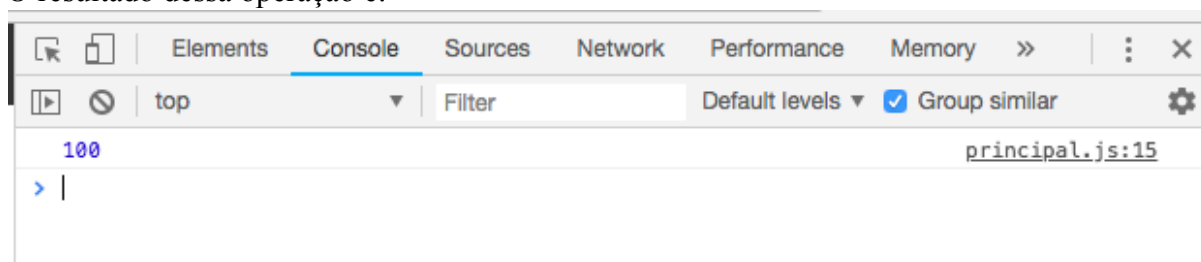
var tdPeso = paciente.querySelector(".info-peso"); // Busca
pela classe que representa o td do Peso
var peso = tdPeso.textContent; // Conteúdo do td peso

var tdAltura = paciente.querySelector(".info-altura"); //
Busca pela classe que representa o td da altura
var altura = tdAltura.textContent; // Conteúdo da td altura

// Calculo do IMC
var imc = peso / altura * altura;

console.log(imc);
```

O resultado dessa operação é:



Se observar a tabela de dados, pode-se notar que existe algo de estranho nessa conta:

Aparecida Nutricionista

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	0
João	80	1.72	40	0
Erica	54	1.64	14	0
Douglas	85	1.73	24	0
Tatiana	46	1.55	19	0

O IMC do Paulo era para ser 25 e não 100. Por que isso? O peso dele é 100 e a altura 2.0 então substituindo na equação ficaria: $imc = 100 / 2.0 * 2.0 \Rightarrow 100 / 4.0 \Rightarrow 25$. Mas o que aconteceu?

O problema é que foi esquecido da precedência matemática na hora de montar a equação no código. Dessa forma, o cálculo ficou: $imc = 100 / 2.0 * 2.0 \Rightarrow 50 * 2.0 \Rightarrow 100$. Para resolver isso, basta utilizar os parênteses separando as operações que devem ser realizadas em primeiro lugar. Sendo assim, o cálculo do imc fica:

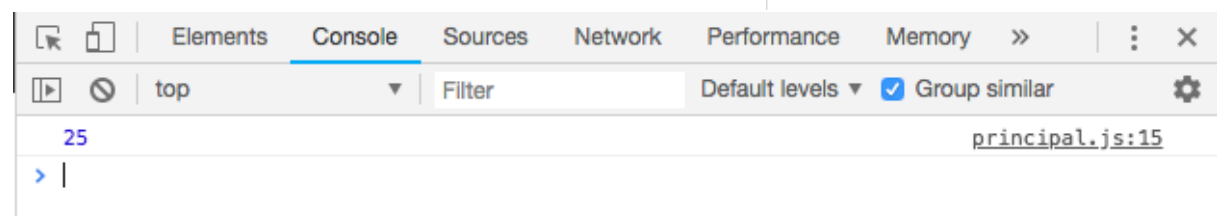
```
// Calculo do IMC
var imc = peso / (altura * altura);
```

Agora sim, com o resultado adequado:

Aparecida Nutricionista

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	0
João	80	1.72	40	0
Erica	54	1.64	14	0
Douglas	85	1.73	24	0
Tatiana	46	1.55	19	0



O próximo passo é inserir o valor do imc calculado na célula da tabela correspondente ao IMC do primeiro paciente. Neste momento esse conteúdo é igual a zero (0). Para que isso seja possível, é necessário selecionar o td referente do IMC e atribuir o valor do imc calculado ao conteúdo de texto da célula. O código fica assim:

```
var tdImc = paciente.querySelector(".info-imc");

var imc = peso / (altura * altura);
```

```
tdImc.textContent = imc;
```

Se tudo estiver correto, o resultado é:

Aparecida Nutricionista

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25
João	80	1.72	40	0
Erica	54	1.64	14	0
Douglas	85	1.73	24	0
Tatiana	46	1.55	19	0

Perceba que o resultado 25 foi atribuído à célula IMC do primeiro paciente de nome Paulo.

Um outro ponto de atenção é a validação dos valores de peso e altura antes de realizar os cálculos do IMC. Por exemplo, um peso negativo não deve ser aceito, nem um peso muito alto como 500kg. Para realizar essas validações utiliza-se os condicionais já existentes nas linguagens de programação como Java e C++. Como fica a validação para o peso?

```
var tdImc = paciente.querySelector(".info-imc");

// Definir variáveis de controle e inicializo com verdadeiro
var pesoEhValido = true;
var alturaEhValida = true;

// Verificar se o peso é negativo ou muito alto
if (peso < 0 || peso > 500) {
    console.log("Peso inválido!");
    pesoEhValido = false; // Dizer que o peso agora é inválido
}

// Verificar se o altura é negativa ou muito alta
if (altura < 0 || altura > 3) {
    console.log("Altura inválida!");
    alturaEhValida = false; // Dizer que a altura agora é inválida
}

// Se o peso e a altura forem válidos, ai pode-se calcular e
exibir o IMC
if (pesoEhValido && alturaEhValida) {
    var imc = peso / (altura * altura);
```

```

        tdImc.textContent = imc;
    }

```

Só que nesse momento, têm-se apenas no console a mensagem de erro. O ideal é que essa mensagem também aparecesse para o usuário, deixando o erro mais visual e de fácil correção. Para isso, utiliza-se a célula de IMC para dizer o que está errado em cada candidato. Assim, basta alterar o conteúdo da célula para a mensagem de erro. O código das validações fica da seguinte maneira:

```

// Verificar se o peso é negativo ou muito alto
if (peso < 0 || peso > 500) {
    console.log("Peso inválido!");
    pesoEhValido = false; // Dizer que o peso agora é inválido
    tdImc.textContent = "Peso inválido!";
}

// Verificar se o altura é negativa ou muito alta
if (altura < 0 || altura > 3) {
    console.log("Altura inválida!");
    alturaEhValida = false; // Dizer que a altura agora é inválida
    tdImc.textContent = "Altura inválida!";
}

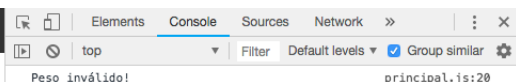
```

O resultado, caso haja um valor inválido:

Aparecida Nutricionista

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	1000	2.00	10	Peso inválido!
João	80	1.72	40	0
Erica	54	1.64	14	0
Douglas	85	1.73	24	0
Tatiana	46	1.55	19	0



Laços de Repetição e Formatação de Estilo

Na atual situação do código, obteve-se o cálculo e validação do IMC para o primeiro paciente. Esse paciente consiste na primeira linha da tabela. Só que existe a necessidade de replicar aos demais pacientes armazenados na tabela. A primeira opção é utilizar do artifício do id exclusivo para cada linha da tabela. Por exemplo, as duas primeiras linhas da tabela são:

```
<tr class="paciente" id="primeiro-paciente" >
  <td class="info-nome">Paulo</td>
  <td class="info-peso">1000</td>
  <td class="info-altura">2.00</td>
  <td class="info-gordura">10</td>
  <td class="info-imc">0</td>
</tr>

<tr class="paciente" >
  <td class="info-nome">João</td>
  <td class="info-peso">80</td>
  <td class="info-altura">1.72</td>
  <td class="info-gordura">40</td>
  <td class="info-imc">0</td>
</tr>
```

Agora basta atribuir um id também para a segunda linha, ficando:

```
<tr class="paciente" id="primeiro-paciente" >
  <td class="info-nome">Paulo</td>
  <td class="info-peso">1000</td>
  <td class="info-altura">2.00</td>
  <td class="info-gordura">10</td>
  <td class="info-imc">0</td>
</tr>

<tr class="paciente" id="segundo-paciente" >
  <td class="info-nome">João</td>
  <td class="info-peso">80</td>
  <td class="info-altura">1.72</td>
  <td class="info-gordura">40</td>
  <td class="info-imc">0</td>
</tr>
```

Contudo, esse artifício é ruim e não remete a uma boa prática uma vez que teremos que criar um paciente2, paciente3 e assim por diante, repetindo várias vezes o mesmo código. A repetição do código gera uma mão de obra desnecessária ao realizar uma manutenção e aumenta a possibilidade de cometer o erro entre pacientes ou registros. E outro ponto crítico é que sempre deverá ser criado mais um trecho de código a cada inclusão de um novo paciente. Isso é inviável para a criação de um sistema.

A boa prática de programação nesse caso é utilizar de um loop (laço de repetição) para repetir o número de iterações necessárias o trecho de cálculo e validação do IMC. O número de iterações nesse caso é determinado pelo número de pacientes. Olhando no trecho de código acima, qual é o seletor que se repete em todas as linhas e representa o paciente como um todo? O seletor mais indicado é a classe paciente, pois ela se repete em todas as linhas da tabela e representam o registro como um todo.

Até agora para selecionar um objeto do HTML utilizou-se a função `querySelector`. Essa função retorna apenas a primeira ocorrência no HTML do seletor informado. Agora a necessidade diz que é necessário retornar todas as ocorrências de um determinado seletor. Para isso, utiliza-se uma nova função a `querySelectorAll`. Ela armazena todas as ocorrências do seletor em uma lista e devolve em um objeto único para ser armazenado em uma variável. Veja o exemplo com a classe paciente:

```
var pacientes = document.querySelectorAll(".paciente");
console.log(pacientes);
```

O resultado é:



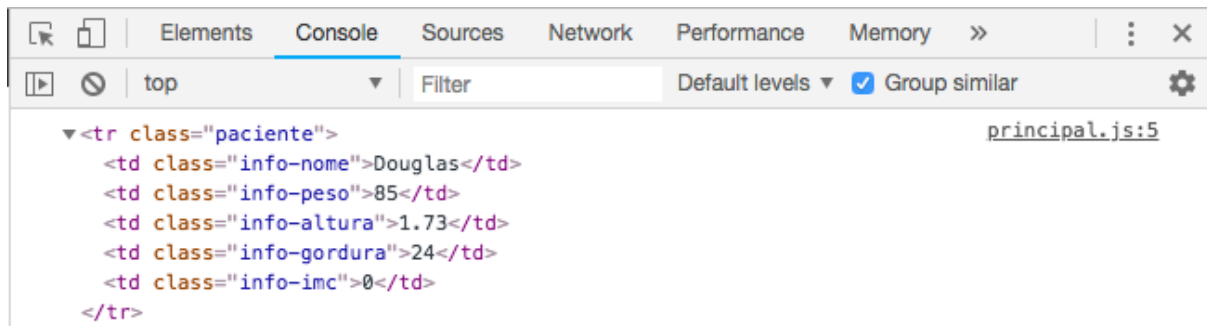
Esse NodeList armazena cada elemento tr com a classe paciente em uma posição. As posições vão de 0 até n-1, no exemplo n-1 = 4, pois existem 5 pacientes na tabela. O NodeList nos proporciona algumas informações úteis para sua manipulação. A principal é a propriedade `length` que informa o tamanho da lista gerada pela função `querySelectorAll`.

Boa prática: Toda vez que se trabalha com listas, o nome da variável que representa a lista é determinado no plural. Por exemplo: a lista de pacientes tem a variável com o nome *pacientes*.

Agora para manipular cada elemento da lista, basta utilizar um dos loops vistos em linguagens de programação como Java e C++. Pode-se utilizar o `for` ou o `while`. E para determinar o elemento da lista que gostaria de visualizar ou manipular, basta informar o seu índice (posição de armazenamento) entre colchetes. Veja a exibição do elemento tr armazenado na posição 3:

```
console.log(pacientes[3]);
```

O resultado é:



Sendo assim, é necessário nesse momento, ajustar o código de cálculo e validação do IMC para que seja realizado em todos os elementos da tabela. O código fica da seguinte maneira:

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

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

    var paciente = pacientes[registro];

    var tdPeso = paciente.querySelector(".info-peso");
    var peso = tdPeso.textContent;

    var tdAltura = paciente.querySelector(".info-altura");
    var altura = tdAltura.textContent;

    var tdImc = paciente.querySelector(".info-imc");

    var pesoEhValido = true;
    var alturaEhValida = true;

    if (peso < 0 || peso > 500) {
        console.log("Peso inválido!");
        pesoEhValido = false;
        tdImc.textContent = "Peso inválido!";
    }

    if (altura < 0 || altura > 3) {
        console.log("Altura inválida!");
        alturaEhValida = false;
        tdImc.textContent = "Altura inválida!";
    }

    if (pesoEhValido && alturaEhValida) {
        var imc = peso / (altura * altura);
        tdImc.textContent = imc;
    }

}
```

Agora têm o cálculo para todos os IMC de pacientes existentes na tabela. Confira:

Aparecida Nutricionista

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25
João	80	1.72	40	27.041644131963228
Erica	54	1.64	14	20.077334919690664
Douglas	85	1.73	24	28.40054796351365
Tatiana	46	1.55	19	19.14672216441207

Existem algumas melhorias que ainda podem ser realizadas nesse momento. Uma delas é a quantidade de casas decimais que são utilizadas na exibição do IMC. Para resolver esse problema de visualização, existe uma função chamada `toFixed`, que limita a quantidade de casas decimais utilizadas na exibição de um número real. Altera-se então o trecho de código que está atribuindo o valor de IMC para o HTML. Esse trecho é:

```
if (pesoEhValido && alturaEhValida) {  
    var imc = peso / (altura * altura);  
    tdImc.textContent = imc;  
}
```

Ele fica:

```
if (pesoEhValido && alturaEhValida) {  
    var imc = peso / (altura * altura);  
    tdImc.textContent = imc.toFixed(2); // Como parâmetro  
    deve-se informar a quantidade de casas decimais desejada  
}
```

O resultado é:

Aparecida Nutricionista

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

O código assim fica mais padronizado, e com uma melhor visualização.

Olhando pela visualização do sistema, existe um outro ponto que deve ser melhorado. Se existir algum erro nos dados que foram inseridos, é exibido apenas uma mensagem no lugar do IMC. Veja o exemplo:

Aparecida Nutricionista

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	800	1.72	40	Peso inválido!
Erica	54	1.64	14	20.08
Douglas	85	10.73	24	Altura inválida!
Tatiana	46	1.55	19	19.15

Para uma tabela pequena, esse tipo de tratamento resolve e atende os objetivos. Porém, quando existir um grande aumento dos registros, essa abordagem torna-se inviável para o usuário. Sendo assim, a melhor maneira de trabalhar com um visual de erros no sistema é com o uso de cores que chamem a atenção para o erro. Uma opção é deixar a linha toda com a cor vermelha para o texto.

O primeiro passo é identificar qual variável armazena o elemento tr da tabela. No último código que foi construído essa variável é o paciente. Sabendo disso, basta agora utilizar a propriedade style em conjunto com a color para atribuir a cor desejada. O código fica assim:

```
if (peso < 0 || peso > 500) {  
    console.log("Peso inválido!");  
    pesoEhValido = false;  
    tdImc.textContent = "Peso inválido!";  
    paciente.style.color = "red";  
}  
  
if (altura < 0 || altura > 3) {  
    console.log("Altura inválida!");  
    alturaEhValida = false;  
    tdImc.textContent = "Altura inválida!";  
    paciente.style.color = "red";  
}
```

O resultado obtido é:

Aparecida Nutricionista

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	800	1.72	40	Peso inválido!
Erica	54	1.64	14	20.08
Douglas	85	10.73	24	Altura inválida!
Tatiana	46	1.55	19	19.15

Já auxilia na visualização, mas ainda não é adequado. Se pensar no cenário de scroll para verificar a tabela, apenas a fonte pode passar despercebido pelo usuário. Uma outra opção é utilizar a formatação do estilo do background da linha. Então o código fica:

```
if (peso < 0 || peso > 500) {  
    console.log("Peso inválido!");  
    pesoEhValido = false;  
    tdImc.textContent = "Peso inválido!";  
    paciente.style.backgroundColor = "red";  
}  
  
if (altura < 0 || altura > 3) {  
    console.log("Altura inválida!");  
    alturaEhValida = false;  
    tdImc.textContent = "Altura inválida!";  
    paciente.style.backgroundColor = "red";  
}
```

Seguido pelo seguinte resultado:

Aparecida Nutricionista

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	800	1.72	40	Peso inválido!
Erica	54	1.64	14	20.08
Douglas	85	10.73	24	Altura inválida!
Tatiana	46	1.55	19	19.15

Agora a visualização do erro é mais clara e fica mais fácil de encontrar os registros com problemas. Pode-se melhorar um pouco mais, alterando a cor vermelha para uma mais fraca, possibilitando assim um melhor contraste na visualização das informações para o usuário. Uma cor que pode ser utilizada nesse caso é a lightcoral. Confira o resultado:

Aparecida Nutricionista

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	800	1.72	40	Peso inválido!
Erica	54	1.64	14	20.08
Douglas	85	10.73	24	Altura inválida!
Tatiana	46	1.55	19	19.15

O problema de visualização no sistema para o usuário foi resolvido. Porém, não é uma boa prática manipular estilos de formatação de maneira tão específica direto no código do JavaScript. Toda a parte de formatação de estilo deve ser realizada no arquivo css. Sendo assim, criei uma classe para representar esse paciente inválido:

```
.paciente-invalido {  
    background-color: lightcoral;  
}
```

Com a classe criada no arquivo css, basta agora adicionar ela ao elemento inválido após a validação do JavaScript. Essa operação é realizada através da propriedade `classList`, que dá acesso a lista de classes aplicada a um determinado elemento HTML. O código fica da seguinte maneira:

```
if (peso < 0 || peso > 500) {  
    console.log("Peso inválido!");  
    pesoEhValido = false;  
    tdImc.textContent = "Peso inválido!";  
    paciente.classList.add("paciente-invalido");  
}  
  
if (altura < 0 || altura > 3) {  
    console.log("Altura inválida!");  
    alturaEhValida = false;  
    tdImc.textContent = "Altura inválida!";  
    paciente.classList.add("paciente-invalido");  
}
```

O resultado permanece o mesmo, confira:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	800	1.72	40	Peso inválido!
Erica	54	1.64	14	20.08
Douglas	85	10.73	24	Altura inválida!
Tatiana	46	1.55	19	19.15

Pronto, todas as responsabilidades do código foram atribuídas de maneira adequada a cada uma das partes do sistema e ainda o usuário tem uma boa apresentação do erro. Nesse momento, todas as linhas que são inseridas na tabela recebem a tratativa de validação dos dados do paciente e também os respectivos cálculos de IMC de todos os pacientes.

Eventos, Formulários e Criando Elementos

Com a tabela de exibição dos dados totalmente funcional, é importante que seja inserido um formulário para que o usuário insira pacientes através de uma interface e não diretamente no código HTML.

O primeiro passo é adicionar o código HTML que define a estrutura do formulário, isso é feito logo após a tag main:

```
<section class="container">
  <h2 id="titulo-form">Adicionar novo paciente</h2>
  <form id="form-adiciona">
    <div class="">
      <label for="nome">Nome:</label>
      <input id="nome" name="nome" type="text"
placeholder="digite o nome do seu paciente" class="campo">
    </div>
    <div class="grupo">
      <label for="peso">Peso:</label>
      <input id="peso" name="peso" type="text"
placeholder="digite o peso do seu paciente" class="campo
campo-medio">
    </div>
    <div class="grupo">
      <label for="altura">Altura:</label>
      <input id="altura" name="altura" type="text"
placeholder="digite a altura do seu paciente" class="campo
campo-medio">
    </div>
    <div class="grupo">
      <label for="gordura">% de Gordura:</label>
      <input id="gordura" name="gordura" type="text"
placeholder="digite a porcentagem de gordura do seu paciente"
class="campo campo-medio">
    </div>

    <button id="adicionar-paciente" class="botao boto-
principal">Adicionar</button>
  </form>
</section>
```

O resultado deve ser o seguinte:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Adicionar novo paciente

Nome:

Peso:

Altura:

% de Gordura:

Adicionar

O comportamento esperado ao preencher esse formulário é que uma nova linha seja inserida na tabela. Ao inserir o sistema deve realizar todo o cálculo de IMC e validação dos demais pacientes, também no novo paciente. Para que a ação ocorra com sucesso, é necessário primeiro identificar quando o usuário clica no botão Adicionar. O clique do usuário em qualquer parte da tela do sistema é informado através de eventos. Os eventos são ações que ocorrem no navegador através da interação do usuário. É possível escutar ou detectar os eventos através do código JavaScript para aí tomar-se uma decisão no sistema.

Escutar um evento, em primeiro lugar, é determinar qual o elemento que será observado. Por exemplo, se quisermos trocar a cor do título ao clicar sobre ele. Deve-se primeiro separá-lo em uma variável. E depois utilizar a função `addEventListener` para escutar o evento. Veja o exemplo a seguir:

```
var titulo = document.querySelector(".titulo");
titulo.textContent = "Aparecida Nutricionista";
```

```
titulo.addEventListener("click", trocarCor);
```

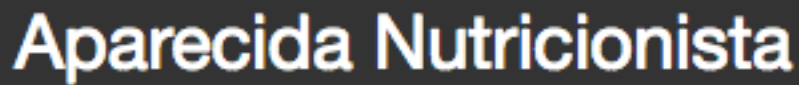
```
// declaração da função que será executada ao escutar o evento.
```

```
function trocarCor() {
    titulo.style.color = "blue";
}
```

A função `addEventListener` recebe dois parâmetros, o primeiro é o evento que deseja escutar, o clique no exemplo. O segundo é a função que deseja executar ao escutar o evento, nesse caso trocar o a cor do título.

O resultado do antes e depois do clique:

Antes:



Depois:



Ao observar o código é possível perceber que a quantidade de eventos em elementos da tela é imenso e trabalhar com essas ações pode se tornar inviável devido ao número de funções que devem ser necessárias para tornar a interação entre o usuário e o sistema mais dinâmica. Além de ser inviável, também fugiria das boas práticas de programação.

Uma solução para isso é o uso da função anônima, um recurso que o JavaScript disponibiliza aos desenvolvedores. A função anônima é a declaração da função direto no parâmetro de uma outra função. Assim não é preciso criar uma função separada. Veja como fica o código:

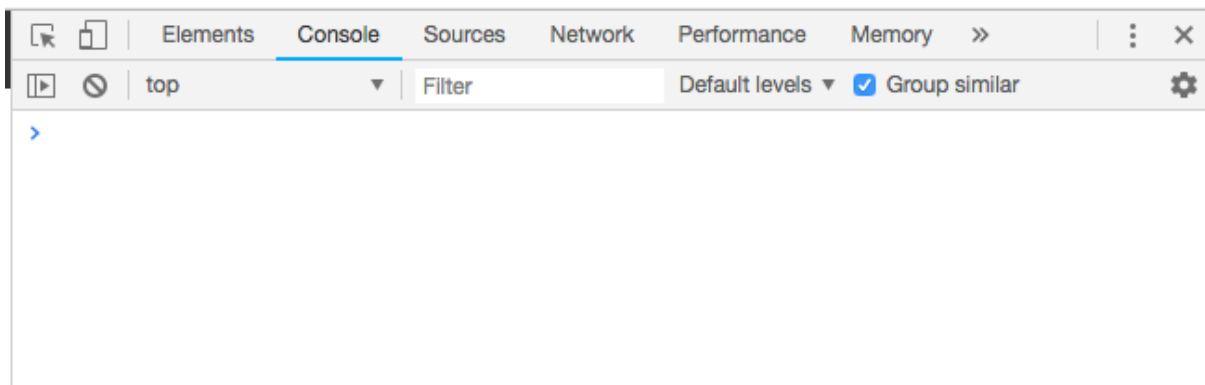
```
titulo.addEventListener("click", function () {  
    this.style.color = "blue";  
});
```

Um outro ponto interessante é que a função é declarada dentro do objeto observado, assim ao manipula-lo não é necessário chamar a variável que representa o elemento. Nesse caso, basta utilizar a palavra `this` que representa o próprio objeto que contém o código feito. O resultado é o mesmo, porém o código fica mais enxuto e de fácil compreensão.

Agora é o momento de utilizar essa técnica para escutar o evento do clique no botão. Veja o código:

```
var adicionar = document.querySelector("#adicionar-paciente");  
  
adicionar.addEventListener("click", function () {  
    console.log("Botão clicado!");  
});
```

O resultado é:



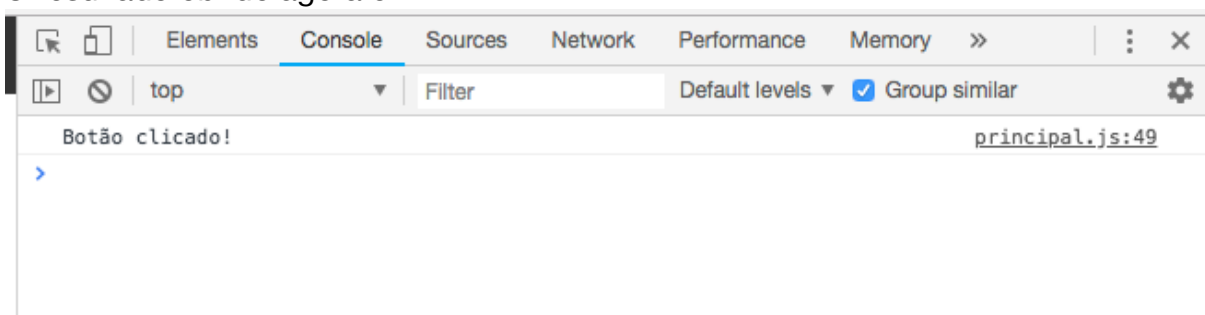
Mas por que a mensagem não apareceu no console como planejado? O problema é o comportamento padrão que o elemento do botão tem. Ele sempre realiza a submissão do formulário na página para processamento no servidor. Isso faz com que a página seja atualizada e limpe o console ao ser exibida novamente no navegador. A solução para esse problema é trabalhar com a prevenção desse comportamento padrão.

Dessa maneira, é necessário informar para a função anônima que deseja-se receber o evento como parâmetro. Com o evento recebido, basta utilizar a função `preventDefault` que vai inibir o comportamento padrão do elemento ao escutar aquele evento. O código fica assim:

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

adicionar.addEventListener("click", function(evento) {
    evento.preventDefault();
    console.log("Botão clicado!");
});
```

O resultado obtido agora é:



Agora sim, é possível trabalhar com as informações do formulário para inclui-las na tabela de pacientes. O primeiro passo é capturar as informações do formulário. Assim, deve-se armazená-lo em uma variável para que seja possível manipulá-lo. Dentro da função anônima do clique do botão é executado um `querySelector` para obter o elemento `form`.

Quando se manipula o elemento `form`, é possível acessar os `inputs` dentro dele através da propriedade `name` de cada um deles. Por exemplo, o peso ou a altura, veja:

```
adicionar.addEventListener("click", function(evento) {
    evento.preventDefault();
```



```

var form = document.querySelector("#form-adiciona");
console.log(form.peso);
console.log(form.altura);
});

```

O resultado é:



Para acessar as informações de um input, não deve-se utilizar a propriedade `textContent`. Nesse caso específico o elemento input possui uma outra propriedade que contém o valor digitado. Essa propriedade é o `value`. O código fica assim:

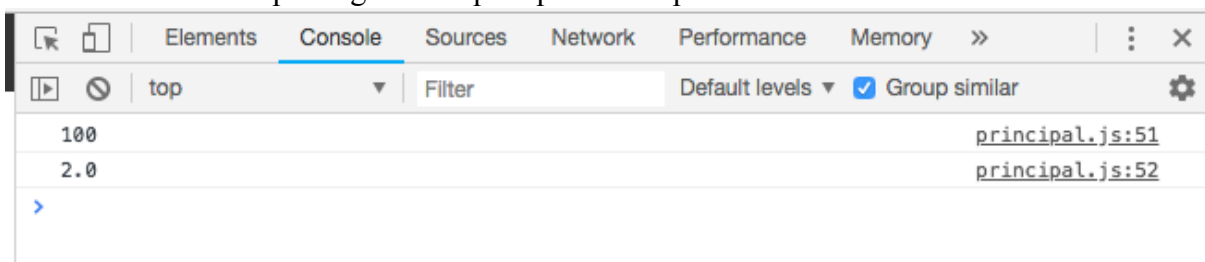
```

adicionar.addEventListener("click", function(evento) {
    evento.preventDefault();

    var form = document.querySelector("#form-adiciona");
    console.log(form.peso.value);
    console.log(form.altura.value);
});

```

Confira o resultado após digitar 100 para peso e 2.0 para altura:



Nesse momento, é possível armazenar os valores do paciente em variáveis, para começar a manipulação da tabela onde serão inseridas as informações. Então, o paciente deve ter os dados digitados separados em variáveis, como no código a seguir:

```

adicionar.addEventListener("click", function(evento) {
    evento.preventDefault();

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

    var nome = form.nome.value;
    var peso = form.peso.value;
    var altura = form.altura.value;
    var gordura = form.gordura.value;

```

```
});
```

Com os dados armazenados no JavaScript, utiliza-se a função `createElement` para começar a criação dos elementos HTML que armazenam e exibem o paciente na tela. Cada paciente é um elemento `tr` e cada dado do paciente é um elemento `td`. Dessa maneira, deve-se criar uma `tr` e cinco `td`'s para cada paciente que será exibido. O código fica assim:

```
adicionar.addEventListener("click", function(evento) {
    evento.preventDefault();

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

    var nome = form.nome.value;
    var peso = form.peso.value;
    var altura = form.altura.value;
    var gordura = form.gordura.value;

    // Criando a tr para o paciente
    var trPaciente = document.createElement("tr");

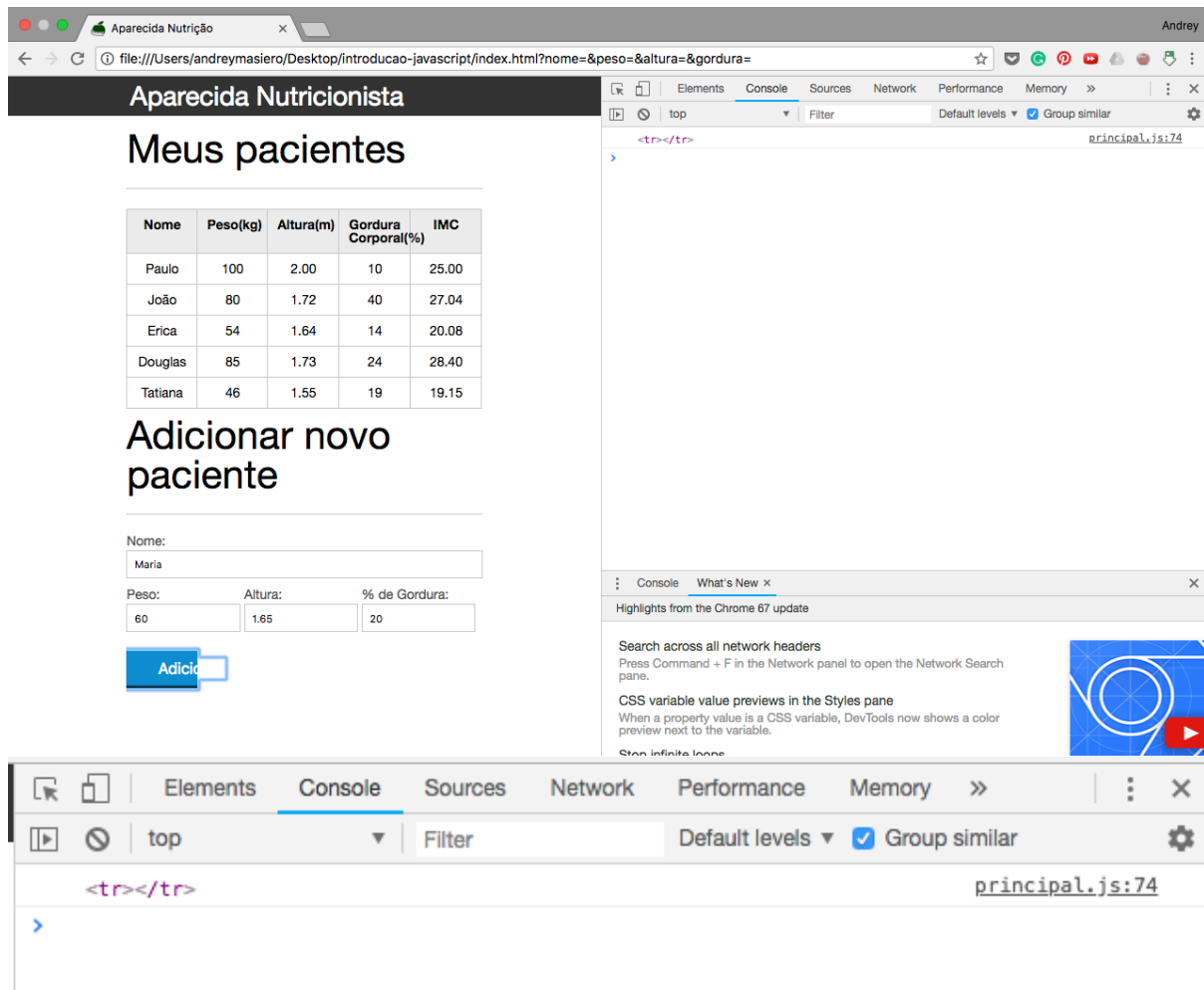
    // Criando as td's que armazenam os dados do paciente
    var tdNome = document.createElement("td");
    var tdPeso = document.createElement("td");
    var tdAltura = document.createElement("td");
    var tdGordura = document.createElement("td");
    var tdImc = document.createElement("td");

    // Atribuindo os valores na propriedade textContent
    tdNome.textContent = nome;
    tdPeso.textContent = peso;
    tdAltura.textContent = altura;
    tdGordura.textContent = gordura;
    tdImc.textContent = (peso / (altura * altura)).toFixed(2);

    console.log(trPaciente);

});
```

O resultado obtido é:



Por que saiu apenas a tr em branco? Os elementos foram criados, mas em nenhum momento adicionou-se os td's à tr. Essa operação é realizada através da função `appendChild`. Confira:

```
adicionar.addEventListener("click", function(evento) {
    evento.preventDefault();

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

    var nome = form.nome.value;
    var peso = form.peso.value;
    var altura = form.altura.value;
    var gordura = form.gordura.value;

    // Criando a tr para o paciente
    var trPaciente = document.createElement("tr");

    // Criando as td's que armazenam os dados do paciente
    var tdNome = document.createElement("td");
    var tdPeso = document.createElement("td");
    var tdAltura = document.createElement("td");
    var tdGordura = document.createElement("td");
    var tdImc = document.createElement("td");
```

```

// Atribuindo os valores na propriedade textContent
tdNome.textContent = nome;
tdPeso.textContent = peso;
tdAltura.textContent = altura;
tdGordura.textContent = gordura;
tdImc.textContent = (peso / (altura * altura)).toFixed(2);

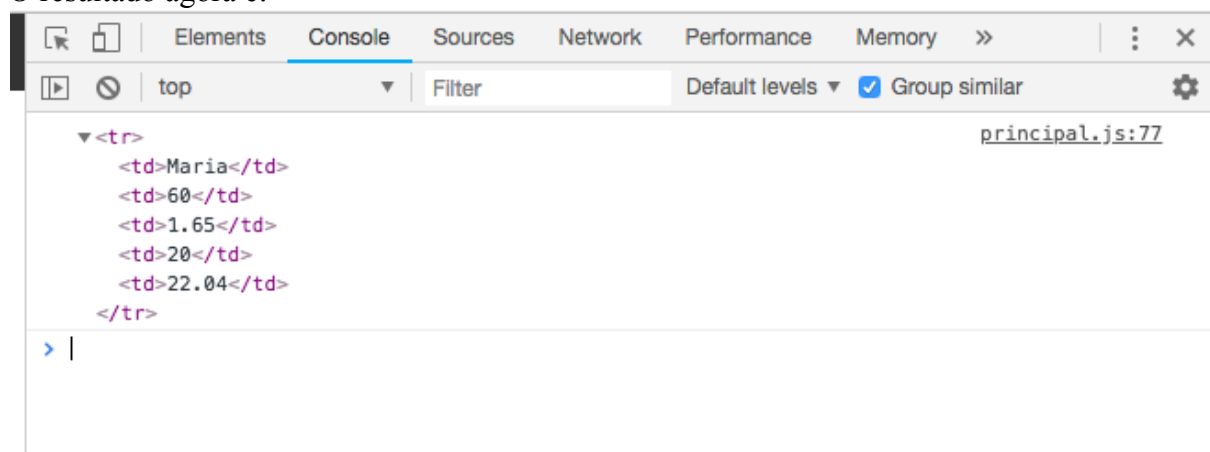
// Inserindo os td's na tr
trPaciente.appendChild(tdNome);
trPaciente.appendChild(tdPeso);
trPaciente.appendChild(tdAltura);
trPaciente.appendChild(tdGordura);
trPaciente.appendChild(tdImc);

console.log(trPaciente);

});

```

O resultado agora é:



Pronto, utilizando o mesmo princípio, adiciona-se a tr na tabela de pacientes. Primeiro selecionando o elemento da tabela e na sequência aplica-se a função appendChild para incluir a tr. O código final fica:

```

adicionar.addEventListener("click", function(evento) {
    evento.preventDefault();

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

    var nome = form.nome.value;
    var peso = form.peso.value;
    var altura = form.altura.value;
    var gordura = form.gordura.value;

    // Criando a tr para o paciente
    var trPaciente = document.createElement("tr");

```

```

// Criando as td's que armazenam os dados do paciente
var tdNome = document.createElement("td");
var tdPeso = document.createElement("td");
var tdAltura = document.createElement("td");
var tdGordura = document.createElement("td");
var tdImc = document.createElement("td");

// Atribuindo os valores na propriedade textContent
tdNome.textContent = nome;
tdPeso.textContent = peso;
tdAltura.textContent = altura;
tdGordura.textContent = gordura;
tdImc.textContent = (peso / (altura * altura)).toFixed(2);

// Inserindo os td's na tr
trPaciente.appendChild(tdNome);
trPaciente.appendChild(tdPeso);
trPaciente.appendChild(tdAltura);
trPaciente.appendChild(tdGordura);
trPaciente.appendChild(tdImc);

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

});

```

E esse é o resultado:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15
Maria	60	1.65	20	22.04

Adicionar novo paciente

Nome:

Maria

Peso:

60

Altura:

1.65

% de Gordura:

20

Adicionar

Boas práticas

Antes de continuar o sistema é necessário realizar uma refatoração do código. Isso é, reestruturar o código de maneira que ele fique mais organizado. O arquivo principal.js possui o seguinte conteúdo:

```
var titulo = document.querySelector(".titulo");
titulo.textContent = "Aparecida Nutricionista";

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

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

    var paciente = pacientes[registro];

    var tdPeso = paciente.querySelector(".info-peso");
    var peso = tdPeso.textContent;

    var tdAltura = paciente.querySelector(".info-altura");
    var altura = tdAltura.textContent;

    var tdImc = paciente.querySelector(".info-imc");

    var pesoEhValido = true;
    var alturaEhValida = true;

    if (peso < 0 || peso > 500) {
        console.log("Peso inválido!");
        pesoEhValido = false; // Dizer que o peso agora é
inválido
        tdImc.textContent = "Peso inválido!";
        paciente.classList.add("paciente-invalido");
    }

    if (altura < 0 || altura > 3) {
        console.log("Altura inválida!");
        alturaEhValida = false; // Dizer que a altura agora é
inválida
        tdImc.textContent = "Altura inválida!";
        paciente.classList.add("paciente-invalido");
    }

    if (pesoEhValido && alturaEhValida) {
        var imc = peso / (altura * altura);
        tdImc.textContent = imc.toFixed(2);
    }
}

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

```

adicionar.addEventListener("click", function(evento) {
    evento.preventDefault();

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

    var nome = form.nome.value;
    var peso = form.peso.value;
    var altura = form.altura.value;
    var gordura = form.gordura.value;

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

    var tdNome = document.createElement("td");
    var tdPeso = document.createElement("td");
    var tdAltura = document.createElement("td");
    var tdGordura = document.createElement("td");
    var tdImc = document.createElement("td");

    tdNome.textContent = nome;
    tdPeso.textContent = peso;
    tdAltura.textContent = altura;
    tdGordura.textContent = gordura;
    tdImc.textContent = (peso / (altura * altura)).toFixed(2);

    trPaciente.appendChild(tdNome);
    trPaciente.appendChild(tdPeso);
    trPaciente.appendChild(tdAltura);
    trPaciente.appendChild(tdGordura);
    trPaciente.appendChild(tdImc);

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

});

```

A partir do código acima, será realizado um processo de refatoração para diminuir a quantidade de códigos repetidos e também separar cada objetivo do código em arquivos exclusivos.

No código acima, à primeira vista, pode ser separado em duas funcionalidades ou objetivos. O primeiro é o cálculo do IMC:

```

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

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

    var paciente = pacientes[registro];

    var tdPeso = paciente.querySelector(".info-peso");

```



```

var peso = tdPeso.textContent;

var tdAltura = paciente.querySelector(".info-altura");
var altura = tdAltura.textContent;

var tdImc = paciente.querySelector(".info-imc");

var pesoEhValido = true;
var alturaEhValida = true;

if (peso < 0 || peso > 500) {
    console.log("Peso inválido!");
    pesoEhValido = false; // Dizer que o peso agora é
inválido
    tdImc.textContent = "Peso inválido!";
    paciente.classList.add("paciente-invalido");
}

if (altura < 0 || altura > 3) {
    console.log("Altura inválida!");
    alturaEhValida = false; // Dizer que a altura agora é
inválida
    tdImc.textContent = "Altura inválida!";
    paciente.classList.add("paciente-invalido");
}

if (pesoEhValido && alturaEhValida) {
    var imc = peso / (altura * altura);
    tdImc.textContent = imc.toFixed(2);
}
}

```

O segundo é a adição do paciente no formulário:

```

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

adicionar.addEventListener("click", function(evento) {
    evento.preventDefault();

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

    var nome = form.nome.value;
    var peso = form.peso.value;
    var altura = form.altura.value;
    var gordura = form.gordura.value;

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

    var tdNome = document.createElement("td");
    var tdPeso = document.createElement("td");
    var tdAltura = document.createElement("td");
    var tdGordura = document.createElement("td");

```

```

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

tdNome.textContent = nome;
tdPeso.textContent = peso;
tdAltura.textContent = altura;
tdGordura.textContent = gordura;
tdImc.textContent = (peso / (altura * altura)).toFixed(2);

trPaciente.appendChild(tdNome);
trPaciente.appendChild(tdPeso);
trPaciente.appendChild(tdAltura);
trPaciente.appendChild(tdGordura);
trPaciente.appendChild(tdImc);

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

});

```

Então o que deve ser feito. Cria-se um arquivo form.js que irá conter todo o conteúdo de código responsável por qualquer interação ou ação do formulário existente. Nele será inserido toda a segunda parte do código acima. Depois dessa ação, deve-se inserir o novo arquivo no HTML, ficando da seguinte maneira:

```

<script type="text/javascript" src="js/principal.js"></script>
<script type="text/javascript" src="js/form.js"></script>

```

Agora, a organização dos arquivos já está melhor. Existe um arquivo js para cada objetivo, ou conjunto de funcionalidades no sistema. Dessa maneira, é mais fácil realizar uma manutenção e inclusão de novas funcionalidades para requisitos do sistema.

Observando agora o código, é possível notar que existem alguns códigos repetidos entre os dois arquivos, como o cálculo do IMC. E se observar apenas o arquivo principal.js existem pelo menos três diferentes funcionalidades em cada trecho do código, por exemplo:

Pegar os dados do paciente:

```

var paciente = pacientes[registro];

var tdPeso = paciente.querySelector(".info-peso");
var peso = tdPeso.textContent;

var tdAltura = paciente.querySelector(".info-altura");
var altura = tdAltura.textContent;

var tdImc = paciente.querySelector(".info-imc");

```

Validar os dados do paciente:

```

var pesoEhValido = true;

```

```

var alturaEhValida = true;

if (peso < 0 || peso > 500) {
    console.log("Peso inválido!");
    pesoEhValido = false; // Dizer que o peso agora é inválido
    tdImc.textContent = "Peso inválido!";
    paciente.classList.add("paciente-invalido");
}

if (altura < 0 || altura > 3) {
    console.log("Altura inválida!");
    alturaEhValida = false; // Dizer que a altura agora é
    inválida
    tdImc.textContent = "Altura inválida!";
    paciente.classList.add("paciente-invalido");
}

```

Calcular o IMC do paciente:

```

if (pesoEhValido && alturaEhValida) {
    var imc = peso / (altura * altura);
    tdImc.textContent = imc.toFixed(2);
}

```

Iniciando a fatora  o pelo c  culo do IMC, basta criar uma fun   o que recebe o peso e a altura como par  metros e retorna o resultado do c  culo do IMC. Por que iniciar pelo c  culo? Ele ser   utilizado nos dois arquivos e ter   uma melhor manuten   o caso mude a regra. A fun   o fica:

```

function calculaImc(peso, altura) {
    var imc = 0;
    imc = peso / (altura * altura);
    return imc.toFixed(2);
}

```

E ela ser   usada no if depois do peso valido e altura valida:

```

if (pesoEhValido && alturaEhValida) {
    var imc = calculaImc(peso ,altura);
    tdImc.textContent = imc;
}

```

E tamb  m na cria   o de um novo paciente, no form.js:

```

tdImc.textContent = calculaImc(peso, altura);

```

Assim, tudo fica padronizado e obt  m-se o mesmo resultado no comportamento do sistema. Aproveitando que foi alterado o c  digo no form.js, deve-se realizar a an  lise de refatora   o nele tamb  m. Veja as poss  veis divis   es das responsabilidades desse c  digo:

Recupera os dados do formul  rio:

```

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

var nome = form.nome.value;

```

```
var peso = form.peso.value;
var altura = form.altura.value;
var gordura = form.gordura.value;
```

Cria o tr e os tds:

```
var trPaciente = document.createElement("tr");

var tdNome = document.createElement("td");
var tdPeso = document.createElement("td");
var tdAltura = document.createElement("td");
var tdGordura = document.createElement("td");
var tdImc = document.createElement("td");
```

Adiciona os valores de cada td:

```
tdNome.textContent = nome;
tdPeso.textContent = peso;
tdAltura.textContent = altura;
tdGordura.textContent = gordura;
tdImc.textContent = calculaImc(peso, altura);
```

Adiciona os tds no tr e o tr na tabela:

```
trPaciente.appendChild(tdNome);
trPaciente.appendChild(tdPeso);
trPaciente.appendChild(tdAltura);
trPaciente.appendChild(tdGordura);
trPaciente.appendChild(tdImc);
```

```
var tabelaPacientes = document.querySelector("#tabela-
pacientes");
tabelaPacientes.appendChild(trPaciente);
```

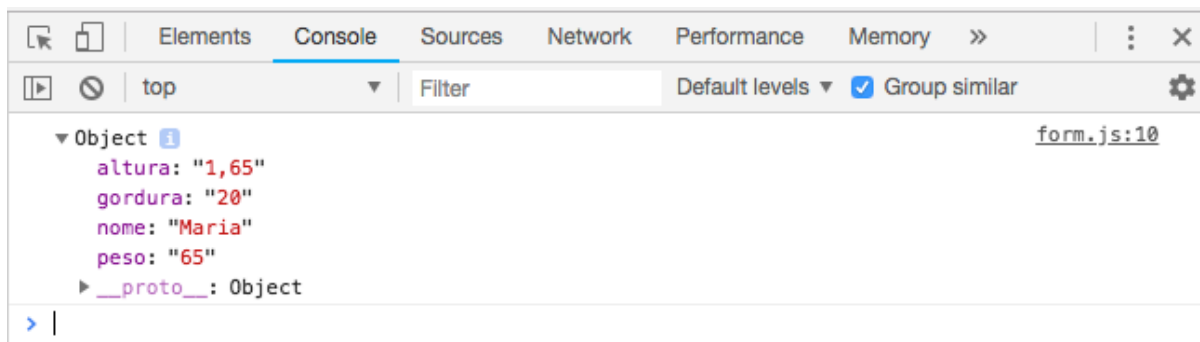
Ao observar apenas a recuperação dos dados do formulário, pode-se criar uma função que recebe como parâmetro o formulário e retorna um paciente para trabalhar no sistema. Para representar o paciente no sistema, a melhor prática é criar um objeto no JavaScript. Para isso utiliza-se chaves e a notação de chave-valor para cada característica ou propriedade desse objeto. Observe o código da função obterPacienteDoFormulario:

```
function obterPacienteDoFormulario(form) {

    var paciente = {
        nome : form.nome.value,
        peso : form.peso.value,
        altura : form.altura.value,
        gordura : form.gordura.value
    }

    return paciente;
}
```

Mas o que efetivamente é esse objeto paciente? Para isso, veja o resultado do console.log(paciente):



Ele é um objeto com as quatro propriedades (características) definidas na criação dentro da função obterPacienteDoFormulario. Esse objeto carrega com ele todas as informações necessária para manipular um paciente no sistema. Se for necessário acessar qualquer uma dessas propriedades de maneira individual basta utilizar o ponto seguido da propriedade, por exemplo, paciente.altura ou paciente.gordura.

Já que o objeto contém todas as informações do paciente, pode-se já definir a propriedade do imc dentro da criação do objeto paciente. Veja o resultado:

```
function obterPacienteDoFormulario(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;  
}
```

Basta agora, alterar a função do evento clique para utilizar a função obterPacienteDoFormulario. Ela fica assim:

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

Isso já elimina muito código e também torna mais legível a função do clique.

Uma outra função que pode ser criada é para montar a tr da tabela referente ao paciente incluído. Então, mãos a obra. Ela recebe o objeto paciente como parâmetro e retorna a tr. Veja:

```
function montarTr(paciente) {  
    var trPaciente = document.createElement("tr");  
  
    var tdNome = document.createElement("td");  
    var tdPeso = document.createElement("td");  
    var tdAltura = document.createElement("td");  
    var tdGordura = document.createElement("td");  
    var tdImc = document.createElement("td");
```

```

    tdNome.textContent = paciente.nome;
    tdPeso.textContent = paciente.peso;
    tdAltura.textContent = paciente.altura;
    tdGordura.textContent = paciente.gordura;
    tdImc.textContent = paciente.imc;

    trPaciente.appendChild(tdNome);
    trPaciente.appendChild(tdPeso);
    trPaciente.appendChild(tdAltura);
    trPaciente.appendChild(tdGordura);
    trPaciente.appendChild(tdImc);

    return trPaciente;
}

```

Ao fazer essa função já torna o código mais legível, porém ainda pode melhorar. Primeiro ponto a melhorar é que nos dados inseridos previamente na tabela, todos possuem uma classe, tanto tr quanto as tds. E isso não ocorre nos novos registros. Para que isso ocorra de maneira automática, é necessário utilizar a função `classList` e o método `add` inserindo as classes na tags corretas. Sendo assim, a nova função `montarTr` fica:

```

function montarTr(paciente) {
    var trPaciente = document.createElement("tr");
    trPaciente.classList.add("paciente");

    var tdNome = document.createElement("td");
    tdNome.classList.add("info-nome");
    tdNome.textContent = paciente.nome;

    var tdPeso = document.createElement("td");
    tdPeso.classList.add("info-peso");
    tdPeso.textContent = paciente.peso;

    var tdAltura = document.createElement("td");
    tdAltura.classList.add("info-altura");
    tdAltura.textContent = paciente.altura;

    var tdGordura = document.createElement("td");
    tdGordura.classList.add("info-gordura");
    tdGordura.textContent = paciente.gordura;

    var tdImc = document.createElement("td");
    tdImc.classList.add("info-imc");
    tdImc.textContent = paciente.imc;

    trPaciente.appendChild(tdNome);
    trPaciente.appendChild(tdPeso);
    trPaciente.appendChild(tdAltura);
    trPaciente.appendChild(tdGordura);
    trPaciente.appendChild(tdImc);
}

```

```
    return trPaciente;
}
```

Perceba que o código de montar a td se repete muitas vezes na função. Uma boa prática é criar mais uma função que monte os tds, onde ela irá receber uma classe e o valor que será atribuído ao conteúdo da td. A função montaTd é exibida a seguir:

```
function montaTd(conteudo, classe) {
    var td = document.createElement("td");
    td.classList.add(classe);
    td.textContent = conteudo;

    return td;
}
```

Com a nova função para montar a td, a função monta tr fica:

```
function montarTr(paciente) {
    var trPaciente = document.createElement("tr");
    trPaciente.classList.add("paciente");

    var tdNome = montaTd(paciente.nome, "info-nome");
    var tdPeso = montaTd(paciente.peso, "info-peso");
    var tdAltura = montaTd(paciente.altura, "info-altura");
    var tdGordura = montaTd(paciente.gordura, "info-gordura");
    var tdImc = montaTd(paciente.imc, "info-imc");

    trPaciente.appendChild(tdNome);
    trPaciente.appendChild(tdPeso);
    trPaciente.appendChild(tdAltura);
    trPaciente.appendChild(tdGordura);
    trPaciente.appendChild(tdImc);

    return trPaciente;
}
```

Perceba que as funções diminuíram e ficaram mais legíveis. Além disso, cada parte possui uma responsabilidade clara do comportamento junto ao sistema. Veja o resultado ao inserir um paciente:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15
Maria	65	1.65	20	23.88

Adicionar novo paciente

Nome:

Maria

Peso:

65

Altura:

1.65

% de Gordura:

20

Adicionar

E o código HTML também ficou adequado aos demais pacientes da tabela, confira:

```

Elements Console Sources Network Performance Memory >>
<tr class="paciente">...</tr>
<tr class="paciente">
  <td class="info-nome">Maria</td> == $0
  <td class="info-peso">65</td>
  <td class="info-altura">1.65</td>
  <td class="info-gordura">20</td>
  <td class="info-imc">23.88</td>
</tr>
</tbody>

```

Perceba que quando a Maria foi adicionada, as informações do formulário se mantiveram. É uma boa prática limpar o formulário após a inserção com sucesso das informações do paciente. O objeto `form` já possui uma função que auxilia nesse processo é a função `reset`. Veja, por fim, como ficou a função do evento do clique do botão:

```

adicionar.addEventListener("click", function(evento) {
    evento.preventDefault();

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

    var paciente = obterPacienteDoFormulario(form);

```



```
    var trPaciente = montarTr(paciente);

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

    form.reset();
});
```

Agora que todo o código foi refatorado de maneira mínima para continuar, pode-se expandir as suas funcionalidades.

Validação dos dados do Formulário

Se analisar o código de modo mais detalhado, percebe-se que as validações realizadas na altura e no peso só se aplicam aos pacientes já existentes na tabela anteriormente. Os que são inseridos pelo formulário não possuem essa validação. Veja:

Aparecida Nutricionista

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15
Maria	65	1.65	20	23.88
José	-100	2.0	15	-25.00
Jessica	70	4.5	20	3.46

Adicionar novo paciente

Nome:

Peso:

Altura:

% de Gordura:

Adicionar

Os paciente José e Jessica tiveram informações de altura e peso inseridas fora da regra de validação. Dessa maneira, uma boa prática é realizar a mesma validação também no formulário, após pressionar o botão adicionar. Uma maneira de solucionar esse problema é criar uma função de validação do paciente e chama-la antes de adicionar esse paciente na tabela.

A função de validação deve receber como parâmetro o objeto paciente, assim fica mais coerente para validar todas as propriedades necessárias. Contudo, toda essa validação já foi feita em outra parte do código, e pelas boas práticas quando existe uma repetição de código é recomendado que seja gerado uma função. Então, para que também seja utilizada na função de validação do paciente em criação, faça a extração das validações de altura e peso criadas anteriormente. Elas se encontram no arquivo principal.js.

As duas funções ficaram da seguinte maneira:

```
function validaPeso(peso) {  
    if (peso > 0 && peso < 500) {  
        return true;  
    } else {  
        return false;  
    }  
}  
  
function validaAltura(altura) {  
    if(altura > 0 && altura < 3) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

Como estão verificando se peso e altura são válidos, é necessário inverter as condições feitas anteriormente no código. Veja:

```
var pesoEhValido = validaPeso(peso);  
var alturaEhValida = validaAltura(altura);  
  
if (!pesoEhValido) {  
    console.log("Peso inválido!");  
    pesoEhValido = false;  
    tdImc.textContent = "Peso inválido!";  
    paciente.classList.add("paciente-invalido");  
}  
  
if (!alturaEhValida) {  
    console.log("Altura inválida!");  
    alturaEhValida = false;  
    tdImc.textContent = "Altura inválida!";  
    paciente.classList.add("paciente-invalido");  
}  
  
if (pesoEhValido && alturaEhValida) {  
    var imc = calculaImc(peso, altura);  
    tdImc.textContent = imc;  
}
```

O ponto de exclamação na frente das variáveis booleanas (verdadeira ou falsa) invertem a condição. É a negação lógica das condições.

Para verificar se está funcionando a validação do paciente também na inserção, é necessário fazer um pequeno teste. O teste será feito com o peso do paciente. A função validaPaciente (temporária) fica da seguinte maneira:

```
function validarPaciente(paciente) {  
    if(validaPeso(paciente.peso)) {
```

```

        return true;
    } else {
        return false;
    }
}

```

E a função do evento do clique para inserir o paciente é:

```

adicionar.addEventListener("click", function(evento) {
    evento.preventDefault();

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

    var paciente = obterPacienteDoFormulario(form);

    if(!validarPaciente(paciente)) {
        console.log("Paciente inválido");
        return;
    }

    var trPaciente = montarTr(paciente);

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

    form.reset();
});

```

Note que caso ela seja inválida, ela exibirá uma mensagem no console. Porém, o paciente não pode ser inserido na tabela. Para evitar essa ação, basta inserir um return vazio. Ele faz com que o código pare de executar e saia da função. Veja o resultado ao tentar inserir um paciente com peso negativo:

Aparecida Nutricionista

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Adicionar novo paciente

Nome:

Maria

Peso:

-10

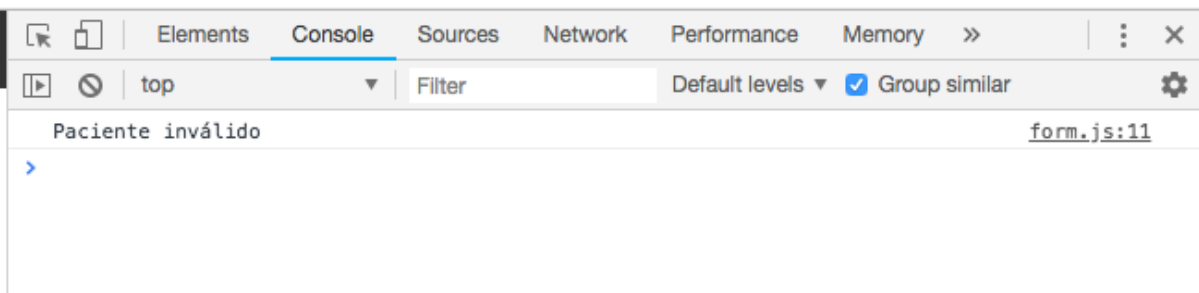
Altura:

1.65

% de Gordura:

20

Adicionar



Porém, é necessário acrescentar a validação da altura e a mensagem de erro deve ser apresentada para o usuário e não no console. Uma maneira de fazer isso é ao invés de retornar verdadeiro ou falso na validação do paciente é retornar uma mensagem de erro:

```
function validarPaciente(paciente) {  
  if (validaPeso(paciente.peso)) {  
    return "";  
  } else {  
    return "Peso inválido";  
  }  
}
```

Se estiver tudo ok retorna uma string vazia, caso contrário retorna a mensagem de erro. Só que no local onde é chamado a validação também precisa de um ajuste:

```

var erro = validarPaciente(paciente);
if(erro.length > 0) {
    // Faz o que?
    return;
}

```

O retorno da função validarPaciente é uma string, então basta verificar se ela é maior que 0 para identificar se houve ou não um erro. Porém onde será exibida essa mensagem? Deve-se criar um elemento span dentro do formulário para que seja exibida na tela do usuário. No HTML faça:

```

...
<h2 id="titulo-form">Adicionar novo paciente</h2>
<span id="mensagem-erro"></span>
<form id="form-adiciona">
...

```

Com o HTML preparado para receber a mensagem de erro, basta inserir na validação o código para exibir o erro:

```

if(erro.length > 0) {
    var msgErro = document.querySelector("#mensagem-erro");
    msgErro.textContent = erro;
    return;
}

```

Veja o resultado obtido com o novo código:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Adicionar novo paciente

Peso inválido

Nome:

Maria

Peso:

-65

Altura:

1.65

% de Gordura:

20

Adicionar

Ainda pode ser melhorado a usabilidade dessa exibição, deixando-a na cor vermelha. Para isso, basta inserir o seguinte código no arquivo css:

```
#mensagem-erro {  
    color: red;  
}
```

Veja o resultado dessa atualização:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Adicionar novo paciente

Peso inválido

Nome:

Maria

Peso:

-65

Altura:

1.65

% de Gordura:

20

Adicionar

Contudo, ainda só é realizado a validação do peso. Agora deve ser realizado o tratamento para os demais campos. Uma função ela só pode retornar um objeto por vez. Não é possível fazer com que ela tenha mais do que um retorno em um mesmo momento. Um recurso fácil para solucionar essa barreira é utilizar uma lista ou array. Assim, a cada erro que existir, será armazenado na lista de erros uma nova mensagem de erro. Veja o código da função validarPaciente para peso e altura:

```
function validarPaciente(paciente) {

    var erros = [];

    if(!validaPeso(paciente.peso)) {
        erros.push("Peso inválido");
    }

    if(!validaAltura(paciente.altura)) {
        erros.push("Altura inválida");
    }

    return erros;
}
```


Pronto, a lista é inicializada através dos colchetes vazios. Depois, foi invertida as condições, pois só existirá uma ação caso a validação seja negativa. Utiliza-se o método push para adicionar um objeto na lista, nesse caso a string com a mensagem de erro. Por fim, retorna a lista de erros para ser exibida.

O retorno da lista de erros impossibilita o trabalho com um único elemento span. Porém, criar vários spans não é a melhor solução para exibir uma lista no HTML. Existe um elemento mais apropriado para uma lista, que é o ul. Nele pode ser inserido diversos item, os li's. Assim, o sistema fica mais adequado a necessidade das mensagens e podem existir mais validações que não haverá necessidade de manutenção do código.

Como o código do JavaScript indica que será um bloco muito grande e pode ser utilizado em outras partes do código, recomenda-se a criação de uma função para isso. Para percorrer a lista de erros é necessário utilizar um laço de repetição como o for ou o while. Porém, o objeto lista do JavaScript permite através de um método um outro tipo de laço. Esse laço é o forEach. O código da função é exibido a seguir:

```
function exibeMensagensDeErro(erros) {  
  
    var msgErro = document.querySelector("#mensagens-erro");  
  
    erros.forEach( function(erro) {  
        var li = document.createElement("li");  
        li.textContent = erro;  
        msgErro.appendChild(li);  
    });  
  
}
```

O forEach é um for especial que irá passar por todos os elementos da lista, sem a necessidade de especificar a quantidade de interação. Ele recebe como parâmetro uma função anônima que representa a ação que ele irá realizar com cada elemento. Para que tenha acesso ao elemento da lista, é necessário declarar um parâmetro que será recebido pela função anônima. Como boa prática, o nome da lista é no plural e o parâmetro da função anônima é o singular, porém pode ser adotado qualquer nome que seja conveniente ao desenvolvedor. O código da função anônima é a criação de um elemento li, a adição do conteúdo e a inclusão dele na ul de erros. Confira o resultado na figura abaixo:

Adicionar novo paciente

Peso inválido
Altura inválida
Nome:

Maria

Peso:

-65

Altura:

3.65

% de Gordura:

20

Adicionar

Com o código bem estruturado, fica mais fácil a adição de novas validações no formulários, como por exemplo, a obrigatoriedade do nome e da gordura do paciente.

```
function validarPaciente(paciente) {  
  
    var erros = [];  
  
    if(paciente.nome.length == 0) {  
        erros.push("Nome do paciente obrigatório");  
    }  
  
    if(!validaPeso(paciente.peso)) {  
        erros.push("Peso inválido");  
    }  
  
    if(!validaAltura(paciente.altura)) {  
        erros.push("Altura inválida");  
    }  
  
    if(paciente.gordura.length == 0) {  
        erros.push("Percentual de gordura obrigatório");  
    }  
  
    return erros;  
}
```

Confira o resultado ao não preencher nenhum campo do formulário:

Adicionar novo paciente

Nome do paciente obrigatório
Peso inválido
Altura inválida
Percentual de gordura obrigatório
Nome:

Peso:

Altura:

% de Gordura:

Adicionar

Porém, existe um problema. Se preencher apenas o nome do paciente agora e clicar em adicionar, o sistema irá acrescentar mais erros. Ele não limpa os erros antigos, veja:

Adicionar novo paciente

Nome do paciente obrigatório
Peso inválido
Altura inválida
Percentual de gordura obrigatório
Peso inválido
Altura inválida
Percentual de gordura obrigatório
Nome:

Maria

Peso:

digite o peso do seu paciente

Altura:

digite a altura do seu paciente

% de Gordura:

digite a porcentagem de gordura do seu pac

Adicionar

Isso pode gerar um grande problema na usabilidade do sistema. Para corrigir esse problema basta utilizar a propriedade `innerHTML`. A propriedade `textContent` permite controlar o conteúdo de texto de um elemento. Já a `innerHTML`, permite controlar todo o HTML dentro de um elemento. A manipulação com o `innerHTML` é mais rústica, direto no código de maneira pura. Então, se adicionar uma string vazia a propriedade `innerHTML` do elemento `ul` da lista de erros, irá remover todos os elementos li inserido nele. A nova função de exibir erros fica:

```
function exibeMensagensDeErro(erros) {  
  
    var msgsErro = document.querySelector("#mensagens-erro");  
    msgsErro.innerHTML = "";  
  
    erros.forEach( function(erro) {  
        var li = document.createElement("li");  
        li.textContent = erro;  
        msgsErro.appendChild(li);  
    });  
  
}
```

As duas linhas acima do `forEach` também podem ser inseridas após o reset do formulário, para limpar as mensagens de erro quando o paciente for adicionado com sucesso após a correção do erro. E assim, todo o formulário é validado com sucesso.

Remoção, Delegação e Animação

Nesse momento, todas as validações estão ocorrendo. Pode-se inserir um novo paciente a qualquer momento do ciclo de vida do sistema. Porém, caso o usuário insira um paciente que atende as regras do negócio, mas ainda sim é errado ou mesmo que ele não deseje mais tê-lo no sistema, o usuário não consegue remove-lo. Então o que deve ser realizado no sistema agora é trabalhar com a remoção de pacientes que estão já inseridos na tabela.

Para adicionar a ação de exclusão do paciente da tabela, pode-se adotar diversas estratégias. Uma delas é criar uma nova coluna com a imagem de X ou da lixeira para quando clicar excluir a linha. Outra opção é trabalhar com o evento de duplo clique, onde ao efetuar o duplo clique a linha que recebeu o evento é excluída da tabela. Para isso, o primeiro passo será introduzir o “escutador” de eventos na linha da tabela. Como já foi realizado o trabalho de refatoração e agora sabe-se como organizar o código, é recomendado que faça essa função em outro arquivo js. Uma sugestão de nome para o arquivo é `remover-paciente.js`, e não deve esquecer de adicioná-lo no HTML.

O evento será colocado em cada linha da tabela. Então, o que deve ser feito é listar todos os pacientes através do método `querySelectorAll` e adicionar para cada um deles o `addEventListener` para escutar o duplo clique. O código fica assim:

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

pacientes.forEach( function(paciente) {
    paciente.addEventListener("dblclick", function() {
        this.remove();
    });
});
```

Note que utilizou a função `forEach` para percorrer a lista. Para cada paciente foi adicionado o escutar do evento de duplo clique (`dblclick`). Dentro da função anônima do evento, chamou-se o `this.remove()`. `Remove` é uma função existente no DOM, que remove o elemento do HTML existente. Já o `this`, é uma palavra reservada para estabelecer o responsável pelo evento. O dono do evento, quem está atrelado ao evento, quem acionou o evento nesse caso é o paciente, que no HTML é uma das `tr`'s da tabela de pacientes. Então, ao efetuar o duplo clique em uma linha ela simplesmente desaparece.

Antes:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Depois:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

A paciente Erica foi removida da tabela, a partir do duplo clique. Porém, existe um pequeno bug, erro nessa implementação. Caso seja, incluído um novo paciente, esse evento não será escutado por ele. Isso ocorre, pois o script é carregado quando foi aberto o navegador. E nesse momento, o novo paciente não existia na página. Se incluir o código de escutar o duplo clique toda vez que criar um paciente, não será uma boa opção, pois duplicará códigos e isso já foi visto que não é uma boa prática. Mas como contornar essa situação?

No JavaScript existe um efeito bolha nos eventos. Isso funciona da seguinte maneira, o evento do duplo clique no paciente, ocorre não só no elemento que está em escuta, mas também em todos os elementos pais. No caso do paciente que é uma tr, o evento também ocorre na tbody, table, section e assim por diante. Ele vai até chegar na tag body. Ou seja, o evento vai do elemento mais específico até o mais geral. Então, em vez de colocar um evento em um elemento mais específico, que fica mais difícil o controle nesse caso, é melhor colocar o evento direto em um elemento mais geral. No caso dos pacientes o mais geral é a tabela-pacientes. O código então fica:

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

tabela.addEventListener("dblclick", function() {
    this.remove();
});
```

Veja o resultado, antes:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Depois:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
------	----------	-----------	---------------------	-----

Ele removeu toda a tabela. O this nesse caso não resolve a questão da remoção. Precisa identificar quem foi o elemento mais específico que sofreu o duplo clique, e depois remover a linha que ele pertence. Solucionando esse problema:

```
tabela.addEventListener("dblclick", function(event) {  
    event.target.remove();  
});
```

É necessário utilizar o parâmetro event para identificar quem foi o elemento que recebeu o clique. Esse elemento é identificado pela propriedade target. A diferença entre o this e o target é que o this é quem escutou o evento e o target é quem realmente sofreu o evento. Veja o resultado.

Antes:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Depois:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
80	1.72	40	27.04	20.08
Erica	54	1.64	14	
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Agora apenas a célula com o nome foi removida. A remoção foi no elemento td, o objetivo é remover o elemento tr. Esse tr é o pai do td. Existe uma outra propriedade que informa quem é o nó pai do elemento no DOM. Essa propriedade é o parentNode. O código fica:

```
tabela.addEventListener("dblclick", function(event) {  
    event.target.parentNode.remove();  
});
```

Veja o resultado. Antes:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Depois:

Meus pacientes

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

O João agora foi removido por completo.

Para finalizar, adiciona-se uma animação da linha durante a exclusão para melhorar o visual e a usabilidade do sistema. É necessário que haja modificações no arquivo CSS e também na função de remover o paciente do JavaScript.

No CSS cria-se uma classe com o nome de `fadeOut`, que é uma animação de esmaecer até que o elemento desapareça por completo. Ela joga a opacidade do elemento para 0 durante a transição de meio segundo. Veja o código da classe `fadeOut`:

```
.fadeOut {  
    opacity: 0;  
    transition: 0.5s;  
}
```

Na parte do JavaScript, será adicionado a classe ao elemento `tr` selecionado. Porém, é necessário que ele espere 0.5 segundos para remover de fato o elemento da tabela. Para que isso ocorra, o JavaScript possui uma função chamada `setTimeout`. Ela possui dois parâmetros, uma função anônima que é o que ele deve executar após o tempo de espera. E o segundo parâmetro é o tempo de espera que deve ser informado em milissegundos. O código final da função de remoção com a animação é:

```
tabela.addEventListener("dblclick", function(event) {  
    event.target.parentNode.classList.add("fadeOut");  
    setTimeout(function() {  
        event.target.parentNode.remove();  
    }, 500);  
});
```

Pronto, a funcionalidade de exclusão de pacientes da tabela está completa.

Filtrando pacientes na tabela

Nesse momento o usuário pode adicionar e excluir qualquer paciente da tabela. Porém, quando existirem muitos pacientes, procurar por um específico torna-se uma tarefa cansativa e nada prática. Uma boa prática é criar um formulário para filtrar os pacientes da tabela. O primeiro passo é criar o campo de filtro no HTML.

```
...
<h2>Meus pacientes</h2>
<label for="filtrar-tabela">Filtro:</label>
<input type="text" name="filtro" id="filtrar-tabela"
placeholder="Digite o nome do paciente">
<table>
...
```

O segundo passo é acrescentar um estilo para o input. Isso é realizado no arquivo CSS:

```
#filtrar-tabela {
    width: 200px;
    height: 35px;
    margin-bottom: 10px;
}
```

O resultado deve ser esse:

Meus pacientes

Filtro:

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Com o HTML e CSS preparados, o próximo passo é trabalhar no JavaScript. Separando em um arquivo exclusivo para a funcionalidade de filtro (filtra.js).

O filtro deve funcionar a partir da digitação do usuário. Toda vez que ele digitar algo a tabela vai removendo os pacientes que não possuem a sequência de caracteres digitados em seu nome. O evento que trabalha a entrada de dados em um campo é o input. É a partir dele que será realizada a filtragem dos elementos.

O primeiro passo é seleccionar o campo de filtro e na sequência adicionar o evento de input para o escutador dele. E depois percorrer todos os pacientes existentes na tabela. Para então conseguir fazer a comparação entre o que foi digitado no filtro e o nome dos pacientes. Esse primeiro passo é representado através do código a seguir:

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

filtro.addEventListener("input", function() {
    var filtroDigitado = this.value;
    var pacientes = document.querySelectorAll(".paciente");

    pacientes.forEach(function(paciente) {
        var nome = paciente.querySelector(".info-
nome").textContent;
    });
});
```

O próximo passo é efetuar a comparação entre o filtroDigitado e o nome do paciente. Se forem iguais, então esse paciente deve ser exibido, caso diferentes, deverão ser ocultos. Deve-se verificar e aplicar o filtro apenas se existe algo digitado, caso contrário, todos os pacientes devem ser mostrados. E se o nome do paciente for igual ao campo de filtro exibir o paciente filtrado e ocultar os demais. O código é:

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

filtro.addEventListener("input", function() {
    var filtroDigitado = this.value;
    var pacientes = document.querySelectorAll(".paciente");

    if (filtroDigitado.length > 0) {
        pacientes.forEach(function(paciente) {
            var nome = paciente.querySelector(".info-
nome").textContent;
            if (nome != filtroDigitado) {
                paciente.classList.add("invisivel");
            } else {
                paciente.classList.remove("invisivel");
            }
        });
    } else {
        pacientes.forEach(function(paciente) {
            paciente.classList.remove("invisivel");
        });
    }
});
```

O conteúdo da classe invisível é:

```
.invisivel {
    display: none;
}
```

Veja alguns prints de exemplo:

Meus pacientes

Filtro:

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
------	----------	-----------	---------------------	-----

Filtro:

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Erica	54	1.64	14	20.08

Filtro:

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Porém, isso não é muito usável. O ideal seria que fossem comparando caracter a caracter e realizando o filtro por partes do nome. Para isso, é necessário o uso de uma técnica chamada de expressão regular. O código de comparação com a expressão regular é:

```
pacientes.forEach(function(paciente) {  
    var nome = paciente.querySelector(".info-nome").textContent;  
    var expressao = new RegExp(filtroDigitado, "i");  
    if (expressao.test(nome)) {  
        paciente.classList.add("invisivel");  
    } else {  
        paciente.classList.remove("invisivel");  
    }  
});
```

Veja que cria-se um objeto do tipo RegExp onde é passado a variável de busca e um segundo parâmetro que é o tipo de sensibilidade de caixa entre as letras. O “i” significa que ele irá ignorar a diferença entre maiúsculas e minúsculas. Se quisesse que essa diferença não fosse

ignorada, opta-se pelo valor de “g”. Mais informações no endereço https://www.w3schools.com/jsref/jsref_obj_regexp.asp. E por fim, na condição do if, utiliza-se a função test passando como parâmetro a outra variável de comparação. Veja o resultado:

Meus pacientes

Filtro:

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Contudo, o nome com p foi escondido. Isso ocorreu, pois a lógica foi invertida no if. Basta nega-la e tudo volta ao normal.

```
pacientes.forEach(function(paciente) {  
    var nome = paciente.querySelector(".info-nome").textContent;  
    var expressao = new RegExp(filtroDigitado, "i");  
    if (!expressao.test(nome)) {  
        paciente.classList.add("invisivel");  
    } else {  
        paciente.classList.remove("invisivel");  
    }  
});
```

Resultado:

Meus pacientes

Filtro:

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00

Ajax

Nesse momento ocorreu uma transferência de pacientes que estavam cadastrados em outro sistema. Então, existe a necessidade de realizar uma integração entre os dois sistemas. Eles estão disponíveis no seguinte endereço: api-pacientes.herokuapp.com/pacientes. Veja o que aparece ao acessar esse endereço:

```
[
  {
    "nome": "Jéssica",
    "peso": 47,
    "altura": 1.54,
    "gordura": 17,
    "imc": 19.82
  },
  {
    "nome": "Flavio",
    "peso": 70,
    "altura": 1.7,
    "gordura": 17,
    "imc": 24.22
  },
  {
    "nome": "Teresa",
    "peso": 60,
    "altura": 1.7,
    "gordura": 13,
    "imc": 20.76
  },
  {
    "nome": "Marina",
    "peso": 75,
    "altura": 1.7,
    "gordura": 26,
    "imc": 25.95
  },
  {
    "nome": "Lucas",
    "peso": 23,
    "altura": 1.25,
    "gordura": 10,
    "imc": 14.72
  },
  {
    "nome": "Stevie",
    "peso": 73,
    "altura": 1.75,
    "gordura": 10,
    "imc": 23.84
  },
  {
    "nome": "Daniel",
    "peso": 78,
    "altura": 1.85,
    "gordura": 19.
  }
]
```

É um array, lista, com diversos pacientes cadastrados no outro sistema. A intenção agora é acessar esse endereço através de uma funcionalidade do sistema e então disponibilizar esses pacientes para o usuário. Para que isso ocorra, será adicionado um botão com a ação de

carregar esses objetos pacientes no tabela. O código HTML que deve ser adicionado após a tabela de pacientes, antes de fechar a tag section:

```
<button id="buscar-pacientes" class="botao boto-principal">Buscar Pacientes</button>
```

Confira o resultado:

Meus pacientes

Filtro:

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Buscar
Pacientes

Para o botão Buscar Pacientes, deve-se criar uma nova funcionalidade. Para manter o código organizado, recomenda-se criar um novo arquivo. Ele se chama buscar-pacientes.js . Não deve esquecer de referencia-lo no final da página.

Primeiro passo, é selecionar o botão com o JavaScript e adicionar um escutador de eventos nele:

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

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

});
```

Na sequência, deve-se realizar o processo de requisição feito no navegador para exibir a lista, porém dentro dessa função. Então como deve ser informado ao JavaScript para acessar o endereço <https://api-pacientes.herokuapp.com/pacientes> e trazer os pacientes gerados por essa api? Utiliza-se um objeto chamado XMLHttpRequest. Esse objeto retorna um arquivo XML com as informações de uma requisição ao servidor utilizando o protocolo HTTP, que é o mesmo protocolo utilizado pelos navegadores. Confira como fica o código:

```
botaoBuscar.addEventListener("click", function() {

    var xhr = new XMLHttpRequest();
```

```

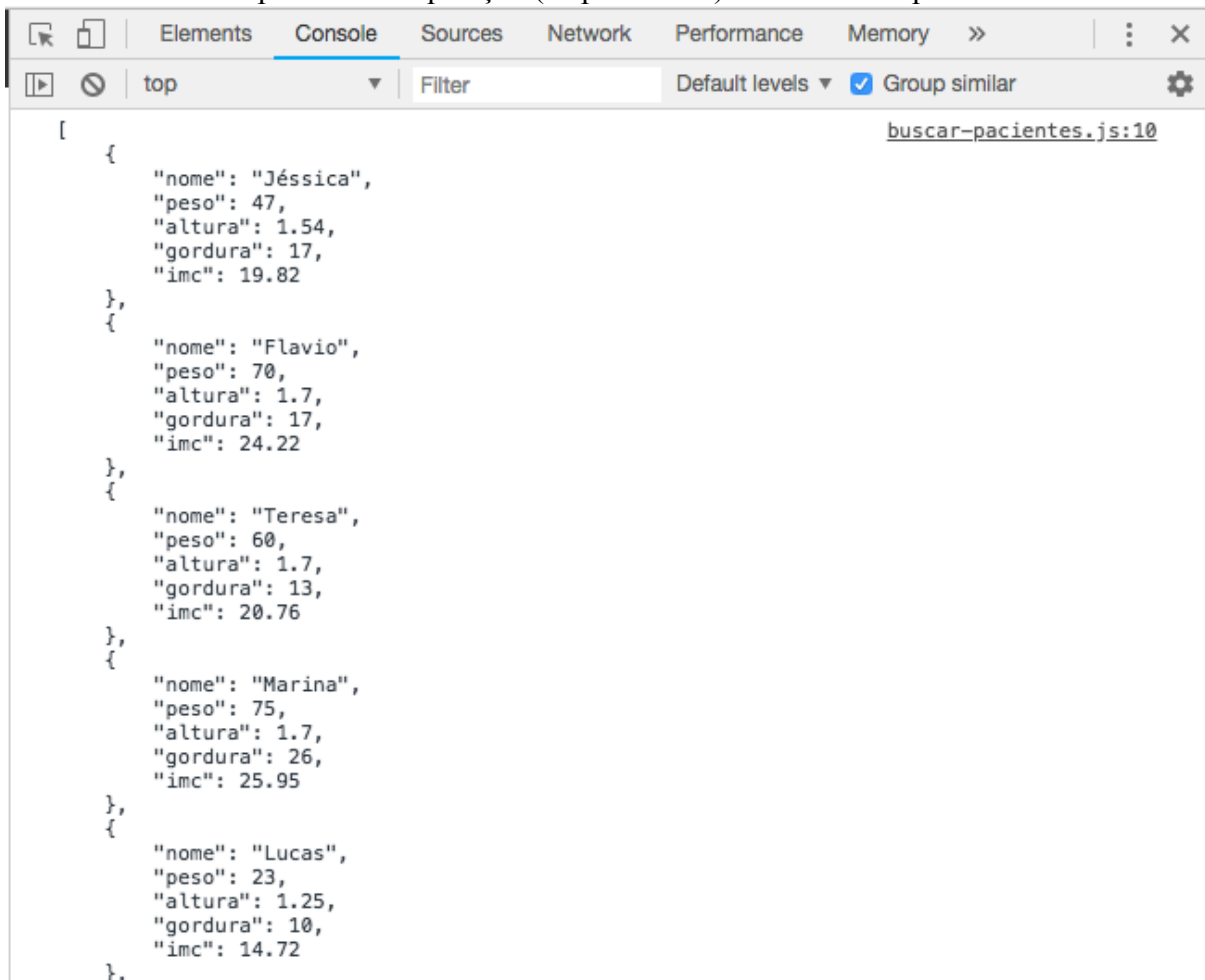
xhr.open("GET", "https://api-
pacientes.herokuapp.com/pacientes");

xhr.addEventListener("load", function() {
    console.log(xhr.responseText);
});

xhr.send();
});

```

No objeto criado xhr, existem diversos métodos que auxiliaram na tarefa de realizar a requisição ao navegador. A primeira é a função open, que funciona como o usuário abrindo o navegador e digitando o endereço que deseja acessar. Ela possui dois parâmetros, o primeiro é o método que será utilizado na requisição. Como essa requisição é simples, utiliza-se o método GET que é responsável por buscar informações. Mais sobre os tipos de métodos de requisições em: <https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>. O segundo parâmetro é o endereço que deseja acessar, no caso a api que lista os pacientes cadastrados no segundo sistema. A função send representa o clicar do botão enter após digitar o endereço no navegador. Ele quem envia a requisição configurada na função open. Porém, antes de enviar a requisição, é necessário adicionar um escutador de eventos ao objeto xhr. Esse escutador de evento realizará uma tarefa após a requisição retornar para que seja possível carregar a página. O evento escutado é o load. No exemplo acima, será realizado apenas a exibição no console do texto respondido à requisição (responseText). O resultado apresentado é:



```

[
  {
    "nome": "Jéssica",
    "peso": 47,
    "altura": 1.54,
    "gordura": 17,
    "imc": 19.82
  },
  {
    "nome": "Flavio",
    "peso": 70,
    "altura": 1.7,
    "gordura": 17,
    "imc": 24.22
  },
  {
    "nome": "Teresa",
    "peso": 60,
    "altura": 1.7,
    "gordura": 13,
    "imc": 20.76
  },
  {
    "nome": "Marina",
    "peso": 75,
    "altura": 1.7,
    "gordura": 26,
    "imc": 25.95
  },
  {
    "nome": "Lucas",
    "peso": 23,
    "altura": 1.25,
    "gordura": 10,
    "imc": 14.72
  }
]

```


O próximo passo é transformar essa resposta em uma lista de pacientes para que sejam adicionados na tabela. O tipo de retorno na propriedade `responseText` é `string`. Um simples texto, sem nenhuma característica especial. Porém, se observar detalhadamente, o formato da `string` é o mesmo de um arquivo do tipo `JSON`. Esse formato de arquivo é o mesmo utilizado para declaração de objetos em `JavaScript`, assim como o paciente criado anteriormente.

Para realizar essa tradução de `string` para uma lista de objetos `JavaScript` existe uma função no objeto `JSON`. Essa função é a `parse`. Veja o código a seguir e confira como fica a resposta da requisição:

```
botaoBuscar.addEventListener("click", function() {

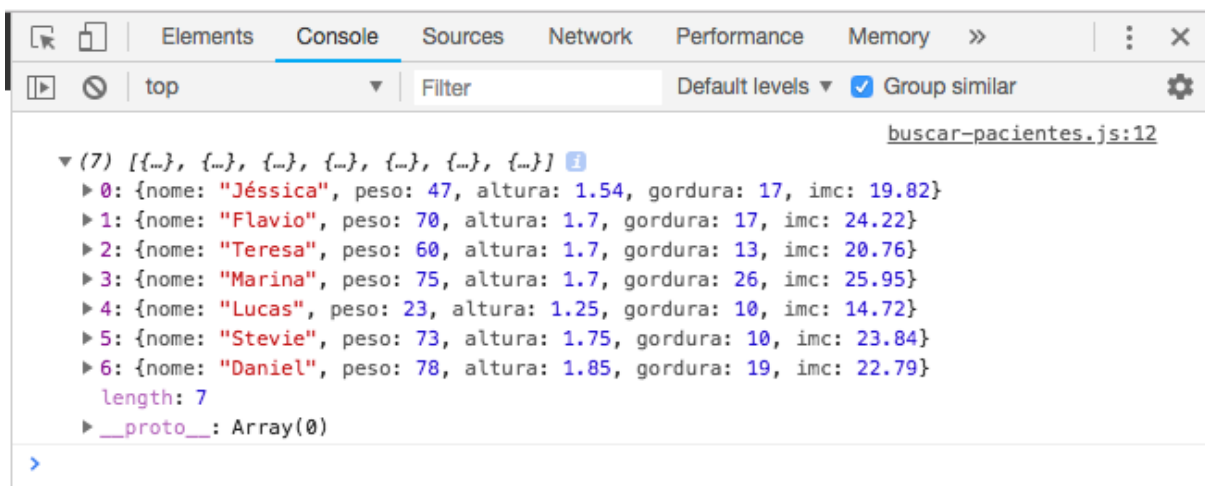
    var xhr = new XMLHttpRequest();

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

    xhr.addEventListener("load", function() {
        var resposta = xhr.responseText;
        var pacientes = JSON.parse(resposta);
        console.log(pacientes);
    });

    xhr.send();
});
```

O resultado obtido é:



Perceba que o resultado obtido é o mesmo da lista de pacientes que obtêm-se ao ler os pacientes da tabela, após os tratamentos inseridos. Com a lista em mãos, basta adicionar o paciente na tabela. No arquivo `form` já existe essa funcionalidade, porém ela não está em uma função exclusiva para isso. Então, deve-se refatorar o código para que exista a função que possa ser reaproveitada também com a resposta da requisição. O código atual é:

```
adicionar.addEventListener("click", function(evento) {
    evento.preventDefault();
```

```

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

var paciente = obterPacienteDoFormulario(form);

var erros = validarPaciente(paciente);

if(erros.length > 0) {
    exibeMensagensDeErro(erros);
    return;
}

var trPaciente = montarTr(paciente);

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

form.reset();
var msgErro = document.querySelector("#mensagens-erro");
msgErro.innerHTML = "";
});

```

Depois de refatorado:

```

adicionar.addEventListener("click", function(evento) {
    evento.preventDefault();

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

    var paciente = obterPacienteDoFormulario(form);

    var erros = validarPaciente(paciente);

    if(erros.length > 0) {
        exibeMensagensDeErro(erros);
        return;
    }

    adicionarPacienteNaTabela(paciente);

    form.reset();
    var msgErro = document.querySelector("#mensagens-erro");
    msgErro.innerHTML = "";
});

function adicionarPacienteNaTabela(paciente) {
    var trPaciente = montarTr(paciente);
    var tabelaPacientes = document.querySelector("#tabela-
pacientes");
    tabelaPacientes.appendChild(trPaciente);
}

```

```
}
```

Pronto, agora é só incluir na função de buscar os pacientes em outro sistema.

```
botaoBuscar.addEventListener("click", function() {
```

```
    var xhr = new XMLHttpRequest();
```

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

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

```
        var resposta = xhr.responseText;
```

```
        var pacientes = JSON.parse(resposta);
```

```
        pacientes.forEach(function(paciente) {
```

```
            adicionarPacienteNaTabela(paciente);
```

```
        });
```

```
    });
```

```
    xhr.send();
```

```
});
```

Quando clicar no botão de buscar pacientes, eles são retornados da requisição e adicionados na tabela de pacientes. Confira o resultado:

Meus pacientes

Filtro:

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15
Jéssica	47	1.54	17	19.82
Flavio	70	1.7	17	24.22
Teresa	60	1.7	13	20.76
Marina	75	1.7	26	25.95
Lucas	23	1.25	10	14.72
Stevie	73	1.75	10	23.84
Daniel	78	1.85	19	22.79

Buscar
Pacientes

A funcionalidade está de acordo com o requisito desejado. Contudo, existe a possibilidade de existir um erro no processo de requisição e resposta. Para validar se houve ou não erro, existe uma propriedade de status no objeto xhr. Quando a resposta está correta, sem erros, esse status é o 200. Então, pode-se testar para só realizar a adição dos pacientes na tabela se o status for 200. Caso contrário, ele apresentará o status e o responseText no console para que o desenvolvedor possa resolver o problema. O código fica:

```
botaoBuscar.addEventListener("click", function() {  
  
    var xhr = new XMLHttpRequest();  
  
    xhr.open("GET", "https://api-pacientes.herokuapp.com/pacientes");  
  
    xhr.addEventListener("load", function() {  
        if (xhr.status == 200) {  
            var resposta = xhr.responseText;  
            var pacientes = JSON.parse(resposta);  
  
            pacientes.forEach(function(paciente) {  
                adicionarPacienteNaTabela(paciente);  
            });  
        }  
    });  
});
```

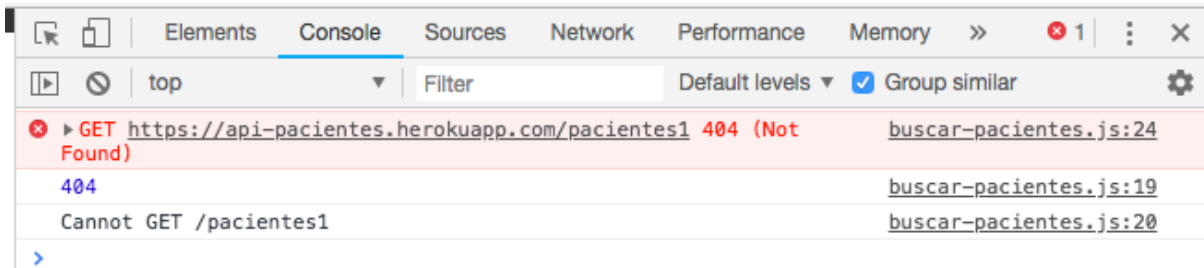
```

    } else {
        console.log(xhr.status);
        console.log(xhr.responseText);
    }
});

xhr.send();
});

```

Simulando o erro de url não encontra, observe o retorno no console:



Porém, apenas isso não basta. É necessário que o usuário também saiba que ocorreu um erro, para não ficar esperando o sistema responder. Para isso, pode-se colocar acima do botão, uma tag span com a mensagem de erro oculta. Ela será exibida sempre que ocorrer um erro. Como fica o código então:

HTML:

```
<span id="erro-request" class="invisivel">Erro ao buscar os pacientes</span>
```

JavaScript:

```

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

    var xhr = new XMLHttpRequest();

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

    xhr.addEventListener("load", function() {
        if (xhr.status == 200) {
            var resposta = xhr.responseText;
            var pacientes = JSON.parse(resposta);

            pacientes.forEach(function(paciente) {
                adicionarPacienteNaTabela(paciente);
            });
        } else {
            console.log(xhr.status);
            console.log(xhr.responseText);
            var erro = document.querySelector("#erro-request");
            erro.classList.remove("invisivel");
        }
    });
});

```

```
xhr.send();  
});
```

Resultado na tela:

Meus pacientes

Filtro:

Nome	Peso(kg)	Altura(m)	Gordura Corporal(%)	IMC
Paulo	100	2.00	10	25.00
João	80	1.72	40	27.04
Erica	54	1.64	14	20.08
Douglas	85	1.73	24	28.40
Tatiana	46	1.55	19	19.15

Erro ao buscar os pacientes

Buscar
Pacientes

Pronto. O sistema está completo agora.