



Programação Web Front-End

Aula 3 - JavaScript

Profa. Rosangela de Fátima Pereira Marquesone
romarquesone@utfpr.edu.br

Proposta: apresentar as funções para manipulação da API DOM via JavaScript.

Objetivos: espera-se que após essa aula, você tenha habilidade para compreender os seguintes tópicos:

1. [Aprender a utilizar o método `querySelector\(\)` e `querySelectorAll\(\)`](#)
2. [Aprender a utilizar o método `classList.add\(\)`](#)
3. [Aprender a utilizar o método `classList.remove\(\)`](#)
4. [Criar uma lista de tarefas com HTML, CSS e JavaScript](#)

Dicas de aprendizado:

- Execute todos os passos com atenção, compreendendo o que está sendo realizado;
- Procure não copiar código, para ter a prática de digitar o código desenvolvido;
- Pergunte quando tiver alguma dúvida;
- Mantenha um histórico dos códigos desenvolvidos, seja no github ou em algum outro meio de armazenamento (e-mails, google drive, etc.);
- Tenha curiosidade e explore os recursos apresentados.

Tópicos anteriores:

- Compreender o que é HTML
- Compreender o que são tags HTML básicas
- Criar um arquivo .html no Visual Studio (VS) Code
- Abrir o arquivo .html em um navegador
- Visualizar o código-fonte de uma página em um navegador
- Inspeccionar a página em um navegador
- Utilizar o Live Server no VS Code
- Aprender a utilizar tags semânticas
- Aprender a inserir links
- Aprender a inserir listas
- Aprender a criar uma página com seu Curriculum Vitae (CV) (atividade prática)
- Aprender a inserir figuras
- Aprender a utilizar a tag semântica `<figure>`
- Inserir figuras em seu Curriculum Vitae (CV) (atividade prática)
- Aprender a criar formulários
- Criar um formulário (atividade prática)
- Descobrir o que é CSS
- Aprender a sintaxe do CSS

- Aprender os tipos de seletores CSS
- Aprender as formas de inclusão de CSS
- Aprender a definir cores
- Aprender a alterar as propriedades de texto
- Aprender o conceito de modelo de caixa do CSS
- Aprender a trabalhar com a margem
- Aprender a trabalhar com a borda
- Aprender a trabalhar com o preenchimento (padding)
- Aprender a usar a propriedade display
- Aprender a utilizar a propriedade float
- Aprender a utilizar a propriedade overflow
- Estruturar páginas por meio do modelo de caixa (atividade prática)
- Aprender o conceito de flex-box
- Aprender as propriedades do elemento pai (flex container)
- Aprender as propriedades dos elementos filhos (flex items)
- Descobrir a história da linguagem JavaScript
- Compreender as formas de uso da linguagem JavaScript
- Conhecer as características da linguagem JavaScript
- Atividade prática com JavaScript
- Aprender a utilizar JavaScript com DOM HTML
- Atividade prática com JavaScript - Criação de um jogo

Passo 1 - Aprender a utilizar o método `querySelector()` e `querySelectorAll()`

Além dos métodos vistos no tutorial anterior para a manipulação de DOM, um dos métodos muito utilizado para acessar os elementos HTML via JavaScript, por meio da API DOM HTML, é o **`querySelector`**. Ele é considerado um método para selecionar elementos HTML com base em um determinado seletor CSS (atributo, classe, ID, etc.).

A partir da sua chamada, ele retorna o primeiro elemento que corresponde ao seletor especificado, ou null, caso não o encontre.

Veja alguns exemplos a seguir:

```
// Seleção por um seletor de atributos
var elemAtributo = document.querySelector('input[type="text"]')

// Seleção por um seletor de pseudo-classe
var elemPseudoClasse = document.querySelector("a:hover");

// Seleção por um seletor de ID
var elemID = document.querySelector("#meuElementoID");
```

Além do `querySelector`, também é possível utilizar o método **`querySelectorAll`**. Nesse caso, torna-se possível retornar mais de um elemento, conforme o exemplo a seguir.

```
// Seleção de todos os elementos com a classe "minhaClasse"
var elementos = document.querySelectorAll(".minhaClasse");
```

É importante identificar que o método `querySelectorAll` retorna uma **`NodeList`**, ou seja, uma coleção de elementos encontrados no documento. Dessa forma, após a captura desses elementos, é possível iterar sobre a `NodeList` por meio de laço de repetição, para realizar ações em cada elemento.

Veja outras formas de obter tais elementos por meio do `querySelectorAll`:

```
var elemento = document.querySelectorAll('p');
var elemento = document.querySelectorAll('p, h1');
var elemento = document.querySelectorAll('.warning');
var elemento = document.querySelectorAll('span.error');
```

Confira também um exemplo de laço de repetição para iterar sobre os elementos obtidos.

```
// Seleção dos elementos com a classe "minhaClasse"
var elementos = document.querySelectorAll(".minhaClasse");
```

```
// Iteração sobre elementos usando um laço for
for (var i = 0; i < elementos.length; i++) {
    var elemento = elementos[i];
    console.log(elemento.textContent);
}
```

Perceba que o atributo **elementos.length** pode ser utilizado para identificar o tamanho de elementos retornados pelo método. Após isso, cada elemento pode ser atribuído a uma nova variável, para ser utilizado conforme necessário.

PRATICANDO: Descompacte o arquivo **exemplos-aula3-js.zip** disponível no moodle e abra o arquivo exemplo1.html no Visual Studio Code. Após isso, visualize o conteúdo e abra o arquivo em um navegador, via Live Server.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Exemplo 1</title>
<style>
  .box {
    border: 2px solid black;
    margin: 20px;
  }
</style>

</head>
<body>

<h1>Elemento H1</h1>

<div>Elemento div</div>

<p>Primeiro elemento p.</p>

<p>Segundo elemento p .</p>

<div class="box">
  <p>Elemento p dentro de uma div.</p>
</div>

<p>Clique no botão para alterar a cor de todos os elementos p.</p>

<button onclick="alteraCor()">Alterar cor</button>

<script>
function alteraCor() {
  var elementos = document.querySelectorAll("p");
  var i;
  for (i = 0; i < elementos .length; i++) {
```

```
        elementos[i].style.backgroundColor = "yellow";
    }
}
</script>
</body>
</html>
```

Elemento H1

Elemento div

Primeiro elemento p.

Segundo elemento p .

Elemento p dentro de uma div.

Clique no botão para alterar a cor de todos os elementos p.

Alterar cor

Elemento H1

Elemento div

Primeiro elemento p.

Segundo elemento p .

Elemento p dentro de uma div.

Clique no botão para alterar a cor de todos os elementos p.

Alterar cor

Perceba que a partir do `querySelectorAll`, foi retornado uma `NodeList` contendo todos os elementos do tipo `<p>`. Após iterar sobre esses elementos, foi alterada a propriedade `backgroundColor` para amarelo.

Passo 2 - Aprender a utilizar o método `classList.add()`

Um outro método disponível na API DOM HTML é o **`classList.add()`**, utilizado para adicionar uma classe em múltiplos elementos.

Um dos cenários de sua utilização, pode ser da seguinte maneira:

- Obter uma lista de todos os elementos com um método como `document.querySelectorAll()`;
- Iterar a lista com um laço de repetição;
- Para cada elemento, chamar o método `elemento.classList.add(classe)` para adicionar a classe em a cada elemento.

PRATICANDO: abra o arquivo `exemplo2.html`, e visualize um exemplo no qual o `classList` é adicionado a um elemento, alterando a decoração do texto do elemento.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Exemplo 2</title>
</style>
.italico {
  font-style: italic;
}

.underline {
  text-decoration: underline;
}
</style>
</head>
<body>

<div id="div1">Front-end</div>

<div>Back-end</div>

<div id="div3">Full-stack</div>

<button id="destacar"> Destacar</button>

<script>
  var elementos = document.querySelectorAll('#div1, #div3');
  var btnDestacar = document.getElementById('destacar');
  btnDestacar.addEventListener('click', function() {
    elementos.forEach(function(elem) {
      elem.classList.add('italico', 'underline');
    });
  });
});
```

```
</script>  
</body>  
</html>
```

Front-end
Back-end
Full-stack
Destacar

Front-end
Back-end
Full-stack
Destacar

Veja que nesse exemplo foi utilizado o laço de repetição `elementos.forEach(function(elem) { ... })`. Esse tipo de laço itera por todos os elementos armazenados em uma lista, permitindo que você acesse cada elemento individualmente.

Passo 3 - Aprender a utilizar o método `classList.remove()`

Após utilizar o método `classList.Add()`, outro método que pode ser muito útil é o **`classList.remove()`**, que é usado para remover uma classe em múltiplos elementos, conforme o exemplo a seguir:

- Obter uma lista de todos os elementos com um método como `document.querySelectorAll()`;
- Iterar a lista com um laço de repetição e verifica se o elemento contém uma determinada classe;
- Caso tenha, para cada elemento, usar o método `elemento.classList.remove(classe)` para remover a classe em cada elemento.

PRATICANDO: abra o arquivo `exemplo3.html`. Caso tenha-se como objetivo remover a classe adicionada ao novo clicar do botão, pode-se utilizar o código destacado em vermelho.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Exemplo 3</title>
</style>
.italico {
  font-style: italic;
}

.underline {
  text-decoration: underline;
}
</style>
</head>
<body>

<div id="div1">Front-end</div>

<div>Back-end</div>

<div id="div3">Full-stack</div>

<button id="destacar"> Destacar</button>

<script>
  var elementos = document.querySelectorAll('#div1, #div3');
  var btnDestacar = document.getElementById('destacar');
  btnDestacar.addEventListener('click', function() {
    elementos.forEach(function(elem) {
      if (elem.classList.contains('italico', 'underline')) {
```



```
        elem.classList.remove('italico', 'underline');
    }else{
        elem.classList.add('italico', 'underline');
    }
    });
});
</script>
</body>
</html>
```

Front-end

Back-end

Full-stack

Destacar

Front-end

Back-end

Full-stack

Destacar

Nesse exemplo, foi adicionada uma linha de código que verifica se o elemento possui a classe `italico` ou `destacado`, e, caso a condição seja verdadeira, o método `classList.remove()` é acionado, passando como parentes os nomes das classes que se deseja remover do elemento.

Passo 4 - Criar uma lista de tarefas com HTML, CSS e JavaScript

Para praticar, utilizaremos o código da construção de uma “to-do list”, disponível no site da W3Schools. Esse exemplo permitirá realizar as seguintes atividades:

- Incluir uma nova tarefa à lista
- Visualizar as tarefas adicionadas
- Marcar as tarefas como realizadas
- Excluir tarefas

Para realizar essas funcionalidades, os seguintes métodos vistos nessa aula e nas aulas anteriores serão utilizados:

- getElementById
- getElementsByTagName
- getElementsByClassName
- createElement
- createTextNode
- appendChild
- addEventListener
- querySelector

Também utilizaremos as seguintes propriedades:

- length
- className
- parentElement
- style.display
- tagName
- classList.toggle
- value

Primeiramente, crie um arquivo chamado **index.html**. Nesse arquivo, iremos inserir o seguinte código HTML, contendo um cabeçalho, um campo para entrada de uma tarefa, um elemento span para a inclusão dessa tarefa, e 3 tarefas inicialmente já fixadas.

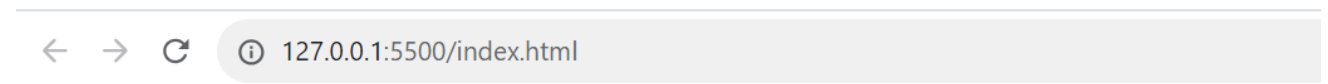
```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Lista de tarefas</title>
</head>
<body>

<div class="header">
  <h2>Lista de tarefas</h2>
  <input type="text" id="tarefa" placeholder="Informe a tarefa">
  <span onclick="addElemento()" class="addBtn">Incluir</span>
```

```
</div>

<ul id="itemLista">
  <li class="checked">Tomar café</li>
  <li>Fazer compras</li>
  <li>Entregar atividade no moodle</li>
</ul>
<script src="script.js"></script>
</body>
</html>
```

Ao visualizar o resultado no navegador, via Live Server, será obtido o conteúdo similar à figura a seguir:



Lista de tarefas

Incluir

- Tomar café
- Fazer compras
- Entregar atividade no moodle

Na sequência, iremos inserir o código CSS a seguir, para alterar as cores, tamanho e posicionamento dos elementos. Para isso, crie um arquivo chamado style.css e inclua o código a seguir. Lembre-se também de fazer a referência do arquivo css no HTML.

```
body {
  margin: 0;
  min-width: 250px;
}

* {
  box-sizing: border-box;
}

ul {
  margin: 0;
  padding: 0;
}

ul li {
  cursor: pointer;
  position: relative;
  padding: 12px 8px 12px 40px;
  background: #eee;
  font-size: 18px;
}
```

```
.close {
  position: absolute;
  right: 0;
  top: 0;
  padding: 12px 16px 12px 16px;
}
```

```
.header {
  background-color: blueviolet;
  padding: 30px 40px;
  color: white;
  text-align: center;
}
```

```
.header:after {
  content: "";
  display: table;
  clear: both;
}
```

```
input {
  margin: 0;
  border: none;
  border-radius: 0;
  width: 75%;
  padding: 10px;
  float: left;
  font-size: 16px;
}
```

```
.addBtn {
  padding: 10px;
  width: 25%;
  background: #d9d9d9;
  color: #555;
  float: left;
  text-align: center;
  font-size: 16px;
  cursor: pointer;
  transition: 0.3s;
  border-radius: 0;
}
```

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" type="text/css" href="style.css">
  <title>Lista de tarefas</title>
</head>
```

A partir desse código no arquivo style.css, podemos ver a mudança no estilo da página, conforme o exemplo a seguir:



Para oferecer uma melhor experiência e melhorar o design da página, iremos atuar com as pseudo-classes, adicionando recursos que alteram o design de acordo com algum determinado estado do elemento, como o passar do mouse ou um click.

Também adicionaremos um conjunto alternado de cores aos elementos da lista, para tornar mais legível. Para isso, inclua os códigos destacados em vermelho.

```
body {  
  margin: 0;  
  min-width: 250px;  
}  
  
* {  
  box-sizing: border-box;  
}  
  
ul {  
  margin: 0;  
  padding: 0;  
}  
  
ul li {  
  cursor: pointer;  
  position: relative;  
  padding: 12px 8px 12px 40px;  
  background: #eee;  
  font-size: 18px;  
}  
  
ul li:nth-child(odd) {  
  background: #f9f9f9;  
}
```

```
ul li:hover {
  background: #ddd;
}

ul li.checked {
  background: #888;
  color: #fff;
  text-decoration: line-through;
}

.close {
  position: absolute;
  right: 0;
  top: 0;
  padding: 12px 16px 12px 16px;
}

.close:hover {
  background-color: blueviolet;
  color: white;
}

.header {
  background-color: blueviolet;
  padding: 30px 40px;
  color: white;
  text-align: center;
}

.header:after {
  content: "";
  display: table;
  clear: both;
}

input {
  margin: 0;
  border: none;
  border-radius: 0;
  width: 75%;
  padding: 10px;
  float: left;
  font-size: 16px;
}

.addBtn {
  padding: 10px;
  width: 25%;
  background: #d9d9d9;
  color: #555;
  float: left;
  text-align: center;
}
```

```
font-size: 16px;
cursor: pointer;
transition: 0.3s;
border-radius: 0;
}

.addBtn:hover {
  background-color: #bbb;
}
```

A partir desse novo código, podemos visualizar a mudança no estilo da página ao passar o mouse nos elementos e ao listar os itens da lista, conforme a figura a seguir.



Por fim, iremos trabalhar com o código Javascript.

Inicialmente, crie um arquivo externo chamado script.js (veja que esse arquivo já está vinculado ao final do código html).

O código a ser inserido é utilizado para incluir um elemento span com o texto “x” a cada item da lista, para que seja adicionada uma função ao evento onclick, que altera o display do elemento para none.

Para isso, inclua o código a seguir.

```
let myNodelist = document.getElementsByTagName("li");

for (let i = 0; i < myNodelist.length; i++) {
  let span = document.createElement("span");
  let txt = document.createTextNode("\u00D7"); //caracter x
  span.className = "close";
  span.appendChild(txt);
  myNodelist[i].appendChild(span);
}

let close = document.getElementsByClassName("close");
```

```
for (let i = 0; i < close.length; i++) {  
  close[i].onclick = function() {  
    let div = this.parentElement;  
    div.style.display = "none";  
  }  
}
```

Após isso, salve o arquivo e visualize o resultado via Live Server.

Conforme é mostrado na figura a seguir, a opção de excluir o elemento é apresentada na lateral direita de cada item. Ao passar o mouse sobre o elemento, a cor também é alterada para a cor roxo.



Na sequência, iremos incluir o código responsável por alterar um item da lista ao ser clicado, podendo anotá-lo como *“checked”*.

Para isso, inclua o seguinte código:

```
let list = document.querySelector('ul');  
  
list.addEventListener('click', function(ev) {  
  if (ev.target.tagName === 'LI') {  
    ev.target.classList.toggle('checked');  
  }  
}, false);
```

O código destacado em vermelho refere-se ao método **toggle('checked')**, que faz parte da propriedade `classList`. Esse método tem como objetivo verificar se a classe "checked" está presente no elemento. Se a classe estiver presente, ela a remove; caso contrário, a adiciona.

Com esse código, torna-se possível habilitar a propriedade `checked` para os demais elementos da lista, conforme a figura a seguir.



Por fim, será adicionada uma função que adiciona uma nova tarefa à lista. Para isso, insira o código a seguir da função `addElemento()`.

```
function addElemento() {
  let li = document.createElement("li");
  let inputValue = document.getElementById("tarefa").value;
  let t = document.createTextNode(inputValue);
  li.appendChild(t);
  if (inputValue === "") {
    alert("Você precisa descrever a tarefa");
  } else {
    document.getElementById("itemLista").appendChild(li);
  }
  document.getElementById("tarefa").value = "";

  let span = document.createElement("SPAN");
  let txt = document.createTextNode("\u00D7");
  span.className = "close";
  span.appendChild(txt);
  li.appendChild(span);

  for (let i = 0; i < close.length; i++) {
    close[i].onclick = function() {
      let div = this.parentElement;
      div.style.display = "none";
    }
  }
}
```

Veja que esse código realiza diversas funcionalidades. Inicialmente, um novo elemento do tipo "li" é criado, para incluir a nova tarefa na lista. Em seguida, o valor do campo de entrada com o ID "tarefa" é recuperado. Este campo é onde o usuário inseriu a descrição da nova tarefa. Após isso, um novo nó de texto é criado com o valor do campo de entrada, representando a descrição da nova tarefa. Assim, o nó de texto criado é anexado ao elemento "li" criado anteriormente.

Caso o texto de entrada esteja vazio, é emitido um pop-up de alerta. Caso não esteja vazio, o novo

elemento "li" é anexado à lista de tarefas (identificada pelo elemento com o ID "itemLista").

Por fim, a última parte do código é responsável por criar e anexar um botão de fechamento ("x") a cada nova tarefa (elemento "li"). Esse botão permite ao usuário remover tarefas da lista quando desejado. Ele é anexado como um elemento "span" com a classe "close", e a função de remoção é associada a esses botões. A parte de estilo desse botão foi inserida anteriormente no código CSS, a partir do seletor ".close".

Uma vez que o código foi inserido, torna-se possível incluir novos elementos da lista, conforme a figura a seguir.



Para dar continuidade ao conhecimento visto neste tutorial, faça as seguintes alterações na lista de tarefas:

1. Inclua um parágrafo após o h1, apresentando ao usuário uma descrição da página web.
2. Altere a cor de fundo de novos itens inseridos, via javascript.
3. Altere a fonte de todos os itens da lista, via javascript.

Considerações finais

Caso tenha chegado até aqui, você conseguiu completar o conteúdo do terceiro tutorial sobre JavaScript. A partir desses recursos, você passa a compreender a base para o desenvolvimento de funcionalidades via JavaScript. Nas aulas seguintes veremos ainda mais funcionalidades para tornar as páginas ainda mais dinâmicas.

Bom estudo!