



Programação Web Front-End

Aula 2 - JavaScript

Profa. Rosângela de Fátima Pereira Marquesone

romarquesone@utfpr.edu.br

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

Objetivos: espera-se que após essa aula, você tenha habilidade para compreender o seguinte tópico:

1. [Aprender a utilizar JavaScript com DOM HTML](#)

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
- Inspecionar 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

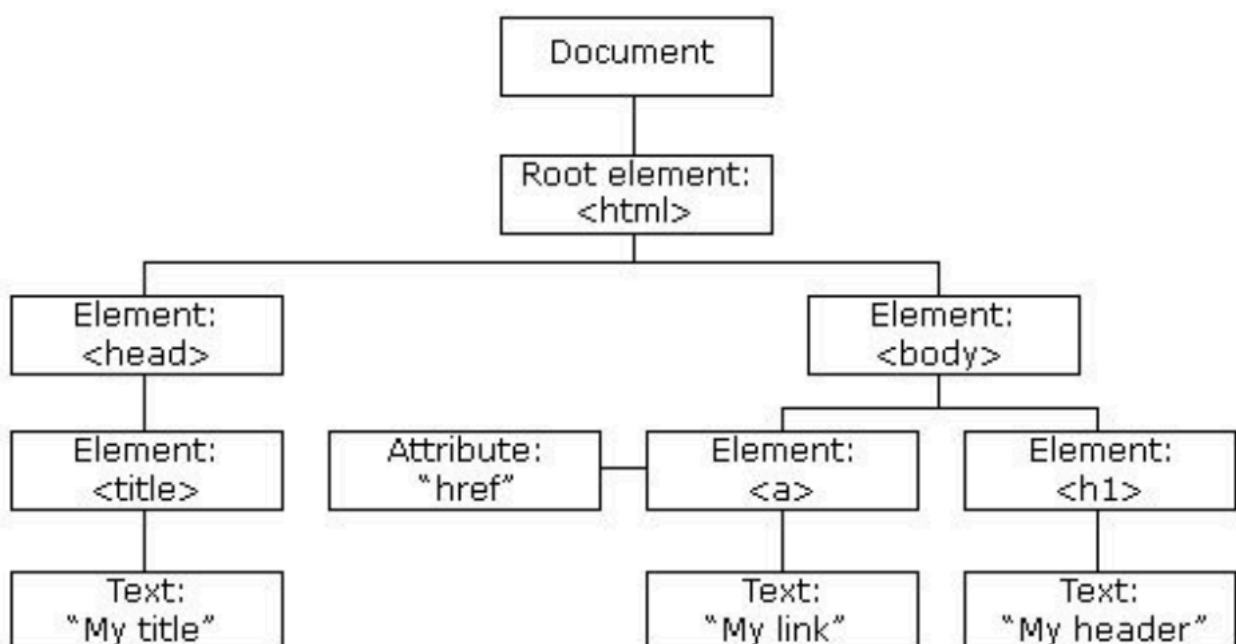
- Aprender a alterar as propriedades de texto
- Aprender o conceito de modelo de caixa do CSS
- Aprender a trabalhar com a margem
- Aprender a trabalhar com a borda
- Aprender a trabalhar com o preenchimento (padding)
- Aprender a usar a propriedade display
- Aprender a utilizar a propriedade float
- 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

Passo 1 - Aprender a utilizar JavaScript com DOM HTML

Como vimos na aula anterior, uma das funcionalidades providas pelo JavaScript é a manipulação do *Document Object Model* (DOM), ou Modelo de Documento por Objetos.

O DOM é uma interface de programação para os documentos HTML. Ele representa a página de forma que os programas possam alterar a estrutura do documento, modificando seu estilo e conteúdo.

Cada página carregada em um navegador possui um objeto chamado Document, conforme exemplificado na figura a seguir.



Dessa forma, a interface Document serve como um ponto de entrada para o conteúdo da página (Ex.: tags <body> e <p>).

Veja a seguir um resumo dos principais conceitos relacionados ao DOM HTML:

- **Árvore de elementos:** o DOM HTML organiza a estrutura de uma página web como uma árvore hierárquica, na qual o elemento <html> é a raiz e os elementos <head> e <body> são seus filhos. Outros elementos HTML, como <div>, <p>, <a>, etc., são aninhados dentro desses elementos pai, formando uma estrutura em árvore que representa a página.
- **Elementos HTML como objetos:** no DOM HTML, cada elemento HTML é representado como um objeto em JavaScript. Assim como é apresentado em disciplinas como Programação Orientada a Objetos (POO), tais objetos possuem propriedades e métodos que podem ser usados para acessar e manipular o conteúdo e os atributos dos elementos.

- **Acesso aos elementos:** possibilita acessar elementos HTML no DOM HTML usando seletores, como getElementById, getElementsByClassName, getElementsByTagName, etc. Esses métodos permitem que você selecione elementos específicos na estrutura DOM e faça a alteração do código HTML via JavaScript.
- **Manipulação do conteúdo:** possibilita alterar o conteúdo de elementos HTML, como texto e atributos, usando propriedades como innerHTML, value, setAttribute, getAttribute, entre outras.
- **Eventos:** o DOM HTML permite a vinculação de eventos a elementos HTML. Permite usar o método addEventListener para ouvir eventos, como cliques de mouse, envios de formulários e digitação, e responder a esses eventos com funções de retorno.
- **Navegação na árvore:** Você pode percorrer a árvore DOM para acessar elementos pai, filhos e irmãos. Isso é útil para navegar pela estrutura da página e acessar elementos relacionados.

A seguir, veremos os seguintes exemplos de APIs em scripting de páginas Web e XML usando a API DOM:

- document.getElementsByTagName(name)
- document.getElementById(id)
- element.getAttribute()
- document.createElement(name)
- parentNode.appendChild(node)
- document.createTextNode(data)
- element.innerHTML
- element.addEventListener()

document.getElementsByTagName(name)

Refere-se a parte da API DOM em JavaScript que permite selecionar elementos HTML em uma página da web com **base em seus nomes de tag** (e.g., p, h1, div). Essa função retorna uma coleção de todos os elementos com a tag especificada.

PRATICANDO: Para verificar o uso da API DOM, crie um arquivo chamado **index.html**, contendo a estrutura a seguir.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Exemplos - JavaScript</title>
</head>
<body>
<main>
  <section id="inicio">
    
  </section>
```

```

<section>
  <h1>Objetivos de Desenvolvimento Sustentável</h1>
  <p id="desc">Em 2015, a ONU propôs aos seus países membros uma nova agenda de
desenvolvimento sustentável para os próximos 15 anos, a Agenda 2030, composta pelos 17
Objetivos de Desenvolvimento Sustentável (ODS). </p>
  <h2>Conheça mais</h2>
  <p>Os Objetivos de Desenvolvimento Sustentável são um apelo global à ação para acabar
com a pobreza, proteger o meio ambiente e o clima e garantir que as pessoas, em todos os
lugares, possam desfrutar de paz e de prosperidade.</p>
</section>
<section>
  <button onclick="mudarCor('red');">Alterar cor</button>
</section>
</main>

<script type="text/javascript">
  //insira o código aqui

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

```

Por exemplo, se você deseja selecionar todos os elementos `<p>` (parágrafos) em uma página da Web, você pode inserir o seguinte código dentro da tag **<script>** em seu arquivo `index.html`:

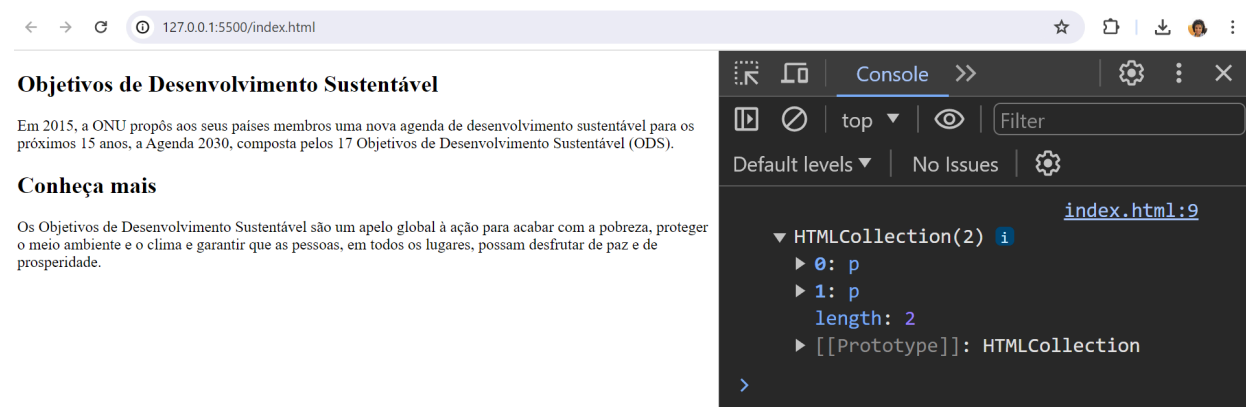
```

var textos = document.getElementsByTagName("p");
console.log(textos);

```

Esse comando percorrerá o DOM e retornará uma coleção de todos os elementos `<p>` na página, atribuindo-os para a variável de nome `textos`.

Ao visualizar o código via Live Server e acessar o console (via `ctrl + shift + j`), você verá que foi retornado via `console.log` os elementos `p` encontrados, conforme a imagem a seguir.



Como o conteúdo da página continha dois parágrafos, foram apresentados dois elementos do tipo `<p>` para o console. Você pode utilizar a seta de cada elemento para identificar as propriedades e os valores pertencentes a eles, conforme o exemplo a seguir.

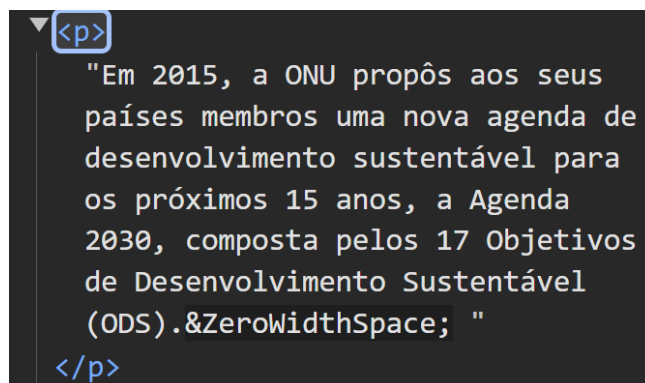


Embora muitas propriedades estejam com valor null, percorra as demais propriedades e veja que muitas delas estão com valores preenchidos, como é o caso da propriedade `innerHTML` e `textContent`, que retorna o texto apresentado no parágrafo.

Você pode acessar e manipular esses elementos por meio da coleção usando índices, assim como faria em um vetor ou matriz. Por exemplo, para acessar o primeiro parágrafo, pode-se utilizar o seguinte comando:

```
var primeiroParagrafo = textos[0];  
console.log(primeiroParagrafo);
```

O resultado será o conteúdo do primeiro parágrafo da página, conforme a figura a seguir.



Uma outra possibilidade para acessar um elemento HTML é você já definir diretamente na chamada do método `getElementsByTagName` qual é o nome da tag que deseja, bem como o índice e a propriedade, conforme exemplo a seguir.

```
console.log(document.getElementsByTagName("p")[1].innerHTML);  
console.log(document.getElementsByTagName("h2")[0]);
```

A primeira linha irá retornar o conteúdo da propriedade `innerHTML` do segundo parágrafo (índice 1) e a segunda linha irá retornar o texto do primeiro elemento `h2` (índice 0), conforme a figura a seguir.

► <p>...</p>

Os Objetivos de [index.html:23](#)
Desenvolvimento Sustentável são um
apelo global à ação para acabar com a
pobreza, proteger o meio ambiente e o
clima e garantir que as pessoas, em
todos os lugares, possam desfrutar de
paz e de prosperidade.

[index.html:24](#)
<h2>Conheça mais</h2>

document.getElementById(id) e element.getAttribute()

document.getElementById(id) é uma função para selecionar um elemento HTML com base no valor de seu atributo id. Cada elemento HTML pode ter um atributo id exclusivo que o identifica de maneira única na página.

Quando você invoca document.getElementById(id), é retornado o elemento que possui o atributo id correspondente ao id passado como argumento.

PRATICANDO: além da função document.getElementById(id) mencionada, há também a função element.getAttribute(attribute), que permite acessar o valor de qualquer atributo específico de um elemento HTML. Você passa o nome do atributo que deseja acessar como argumento, e a função retorna o valor desse atributo. Veja um exemplo a seguir:

```
var img_ods = document.getElementById("ods-img");  
var texto_alt = img_ods.getAttribute("alt");  
console.log(texto_alt);
```

Verifique no console se foi possível recuperar o texto alternativo (atributo “alt”) da imagem.

Além disso, você também pode acessar o atributo diretamente pelo elemento obtido, usando a função getElementsByTagName, conforme exemplo a seguir.

```
function mudarCor(novaCor) {  
    var elemento = document.getElementsByTagName("p")[1];  
    elemento.style.color = novaCor;  
}
```

Essa ação irá gerar a alteração da cor do segundo parágrafo ao clicar no botão “Alterar cor”, conforme a figura a seguir. Observação: lembre-se também que a chamada da função ocorre pelo fato que há a chamada no evento onclick do botão (“<button onclick=“mudarCor('red');”>Alterar cor</button>”)

Conheça mais

Os Objetivos de Desenvolvimento Sustentável são um apelo global à ação para acabar com a pobreza, proteger o meio ambiente e o clima e garantir que as pessoas, em todos os lugares, possam desfrutar de paz e de prosperidade.

Alterar cor

Para saber mais!

Assim como temos a função `document.getElementById`, para selecionar um elemento HTML com base no valor de seu atributo `id`, também há a função `getElementsByClassName`, que seleciona elementos HTML com base no valor de seu atributo `class`.

Confira mais informações sobre essa função nesse [link](#).

`document.createElement(name)`, `parentNode.appendChild(node)` e `document.createTextNode(data)`

`document.createElement(name)` é uma função para **criar um novo elemento** HTML com base no nome da tag especificado como argumento. Ela retorna um novo elemento vazio com a tag especificada, que pode ser posteriormente personalizado e manipulado antes de ser adicionado à página.

Exemplo da criação de uma nova `div`, a partir da função `createElement`:

```
var novoDiv = document.createElement("div");
```

`parentNode.appendChild(node)` é uma função que permite adicionar um nó (elemento, texto, etc.) como um filho de outro nó. O nó de destino é referenciado como `parentNode`, e o nó que você deseja adicionar é referenciado como `node`. Isso é normalmente usado para adicionar o elemento criado com `createElement` (ou qualquer outro nó) a um elemento existente na página.

Veja um exemplo de como adicionar o novo `<div>` criado anteriormente a um elemento pai (por exemplo, o `<body>`):

```
var body = document.body;  
body.appendChild(novoDiv);
```

Assim, ao combinar essas duas funções, você pode criar um novo elemento com `createElement` e, em seguida, adicioná-lo como um filho a um elemento existente com `appendChild`.

`document.createTextNode(data)` é uma função para criar e inserir nós de texto em elementos HTML. Um nó de texto é uma sequência de texto que pode ser inserida em um elemento HTML para exibir texto na página.

Veja um exemplo utilizando as 3 funções mencionadas:


```
#cria um novo elemento p
var novoParagrafo = document.createElement("p");
#cria um texto para o elemento p
var textoNovo = document.createTextNode("Inserindo novo conteúdo sobre ODS!");
#faz um append do texto ao elemento p
novoParagrafo.appendChild(textoNovo);
#busca uma section a partir do ID "inicio"
var conteudo = document.getElementById("inicio");
#faz um append do parágrafo à section buscada
conteudo.appendChild(novoParagrafo);
```

element.innerHTML

`element.innerHTML` é uma propriedade da API DOM para acessar ou modificar o conteúdo HTML interno de um elemento HTML específico. Essa propriedade é usada para ler ou definir o conteúdo HTML de um elemento, permitindo a manipulação do conteúdo de forma dinâmica.

Para acessar o conteúdo HTML de um elemento usando `element.innerHTML`, você pode simplesmente acessar a propriedade, conforme exemplo a seguir:

```
var meuElemento = document.getElementById("desc");
var conteudoHTML = meuElemento.innerHTML;
console.log(conteudoHTML);
```

No exemplo acima, `conteudoHTML` terá o conteúdo HTML interno do elemento parágrafo que possui o id "desc".

Você também pode usar `element.innerHTML` para definir o conteúdo HTML de um elemento. Isso permite que você atualize dinamicamente o conteúdo do elemento. Veja um exemplo a seguir:

```
var meuElemento = document.getElementById("desc");
meuElemento.innerHTML = "Alterando o conteúdo";
```

Perceba que ao usar `element.innerHTML` para definir o conteúdo, ele substituirá completamente o conteúdo existente do elemento.

element.addEventListener()

`element.addEventListener()` é um método da API DOM para vincular funções de retorno de chamada (*event handlers*) a elementos HTML para lidar com eventos específicos. Eventos podem ser compreendidos como ações que ocorrem no navegador, como cliques de mouse, pressionamentos de tecla, carregamento de páginas, etc.

Confira a seguir uma lista com alguns exemplos de eventos que podem ser detectados:

- **blur**: quando um elemento perde o foco.
- **copy**: quando o conteúdo é copiado para a área de transferência.
- **cut**: quando o conteúdo é cortado para a área de transferência.
- **paste**: quando o conteúdo é colado da área de transferência.
- **click**: quando um elemento é clicado.
- **dblclick**: quando um elemento é clicado duas vezes rapidamente.
- **input**: quando o valor de um elemento é alterado em tempo real.
- **mouseover**: quando o ponteiro do mouse é movido sobre um elemento.
- **mouseout**: quando o ponteiro do mouse deixa a área de um elemento.
- **mousemove**: quando o ponteiro do mouse é movido dentro de um elemento.
- **mousedown**: quando um botão do mouse é pressionado sobre um elemento.
- **mouseup**: quando um botão do mouse é liberado após ter sido pressionado.
- **keydown**: quando uma tecla do teclado é pressionada.
- **keyup**: quando uma tecla do teclado é liberada após ter sido pressionada.
- **keypress**: quando uma tecla do teclado é pressionada e liberada.
- **focus**: quando um elemento recebe o foco.
- **load**: quando um recurso, como uma imagem ou uma página, é completamente carregado.
- **resize**: quando a janela do navegador é redimensionada.
- **scroll**: quando o conteúdo de um elemento que possui uma barra de rolagem é rolado.
- **submit**: quando um formulário é enviado.

O método `addEventListener()` é amplamente usado para criar interatividade em páginas da web, pois permite responder a ações do usuário e realizar ações com base nesses eventos.

PRATICANDO: Crie os arquivos a seguir, onde há os códigos de exemplo de uso do `addEventListener()`. Após isso, verifique o resultado via Live Server.

exemplo1.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Exemplo 1</title>
</head>
<body>
  <h1>HTML DOM Events</h1>
  <p>Informe uma frase:</p>
  <input type="text" id="entrada_dados">
  <p id="demo"></p>
  <script>
    const entrada = document.getElementById("entrada_dados");
    const demo = document.getElementById("demo");

    entrada.addEventListener('input', () => {
      let texto = entrada.value;
      demo.textContent = "Texto digitado: " + texto;
    });
  </script>
</body>
</html>
```

HTML DOM Events

Informe uma frase:

Texto digitado: novo texto

exemplo2.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Exemplo 2</title>
</head>
<body>
  <h1>HTML DOM Events</h1>
  <p>Mova o mouse para ver a posição:</p>
  <p id="pos_mouse"></p>
  <script>
    const posicaoMouse = document.getElementById("pos_mouse");

    document.addEventListener('mousemove', (event) => {
      const mouseX = event.clientX;
      const mouseY = event.clientY;
      posicaoMouse.textContent = `Posição X,Y do mouse: (${mouseX}, ${mouseY})`;
    });
  </script>
</body>
</html>
```

HTML DOM Events

Mova o mouse para ver a posição:

Posição X,Y do mouse: (172, 194)

exemplo3.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Exemplo 3</title>
</head>
<body>
  <h1>HTML DOM Events</h1>
  <button id="clickBtn">Clique para Plantar uma Árvore</button>
  <p id="contagem">Árvores plantadas: <span id="valorContador">0</span></p>

  <script>
    let contador = 0;

    const clickBtn = document.getElementById("clickBtn");
    const valorContador = document.getElementById("valorContador");

    clickBtn.addEventListener('click', () => {
      contador++;
      valorContador.textContent = contador;
      alert('Você plantou uma árvore!');
    });
  </script>
</body>
</html>
```

HTML DOM Events

Clique para Plantar uma Árvore

Árvores plantadas: 2

Considerações finais

Caso tenha chegado até aqui, você conseguiu completar o conteúdo do segundo 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!