

JavaScript básico





<title>

AGENDA DO </> </> </> MÓDULO

</title>



Agenda do módulo



Anotações



{O que é o DOM?}

{Coletando e exibindo valores na tela}

{Trabalhando com campos de texto e botões}

{Trabalhando dinamicamente com HTML}

{Customizando estilos}

{Trabalhando com formulários}



<title>

MISSÃO DO MÓDULO

</title>

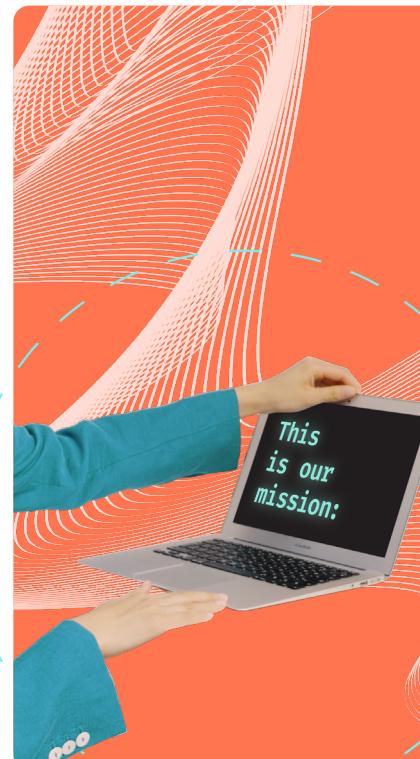
Manipular HTML e CSS usando **DOM** de maneira eficiente.

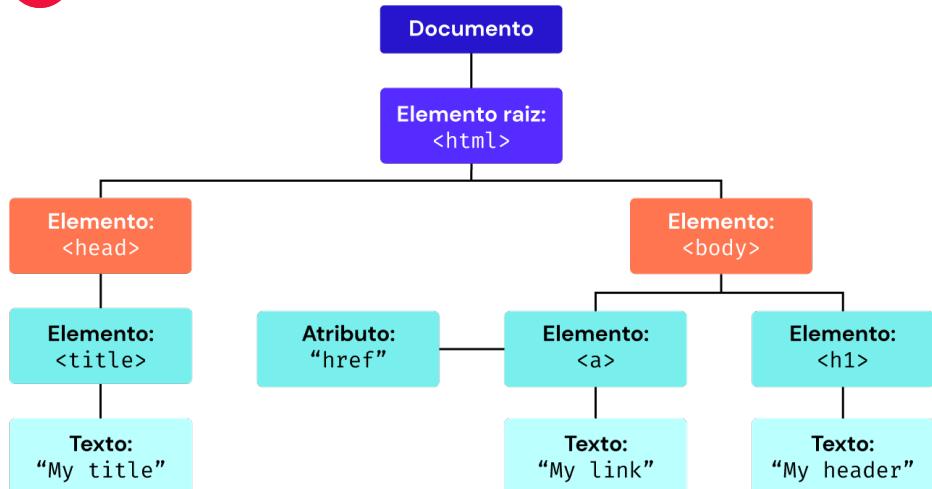
{O1.} O que é o DOM?

DOM quer dizer “Modelo de Documento por Objetos”.

É uma representação de como o navegador lê e interage com a página HTML. Essa interface permite a manipulação dos elementos HTML e CSS por meio de scripts.

W3C DOM e **WHATWG DOM** são os principais padrões de interpretação e manipulação utilizados nos navegadores atuais.





Como integrar o DOM com JS?

Normalmente usamos a tag `<script>` inserida no corpo do arquivo HTML com o endereço de onde está o arquivo, usando `src="nome_do_arquivo_script.js"`.

Aqui, é possível observar a tag `<script>`, como falamos.

```
Console +
```

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <title>Aula 1 - DOM</title>
5  </head>
6  <body>
7      <p class="prg1">Este é um exemplo de um parágrafo</p>
8      <p class="prg2">Este é outro exemplo de parágrafo</p>
9      <script src="exemplos.js"></script>
10 </body>
11 </html>
```



Ela está apontando para um arquivo externo com o código do JS.

```
Console +
```

```
1 var paragrafos = document.getElementsByTagName("p");
2
3 console.log(paragrafos[0]);
```

Caso seja uma implementação mais simples de JS, também podemos inserir o código dentro da própria tag <script> dentro do HTML.

Usando o mesmo exemplo, veja como fica se usado dentro do HTML.

```
Console +
```

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title>Aula 1 - DOM</title>
5   </head>
6   <body>
7     <p class="prg1">Este é um exemplo de um parágrafo</p>
8     <p class="prg2">Este é outro exemplo de parágrafo</p>
9   <!--<script src="exemplos.js"></script>-->
10    <script>
11      var paragrafos = document.getElementsByTagName("p");
12      console.log(paragrafos);
13    </script>
14  </body>
15 </html>
```

O código JavaScript **se manteve o mesmo**, ele apenas foi inserido entre as tags <script> e </script>, em vez de usarmos o src.



<title>

PULO DO GATO

</title>

- Crie o script em um documento separado.



**DOMine
seu HTML
com JS.**



De forma geral, os data types existentes são:

- Document
- Node
- Element
- NodeList
- Attribute
- NamedNodeMap

```
<title>
```

MÃO + NA MASSA

```
</title>
```



Dado o seguinte HTML:

```
<head>
  <title>Tipos de Dados</title>
</head>

<body>
  <div class="parent">
    <div id="children">
      <span>Texto 1</span>
      <span>Texto 2</span>
      <span>Texto 3</span>
    </div>
    <div>
      <input type="text" id="username" value="john doe" />
    </div>
  </div>
</body>

</html>
```

- Atribua o seletor correto à constante '**Element**' para obter o retorno "**div**" no console.
- Atribua o seletor correto à constante '**Element**' para obter o retorno "**Título**" no console.
- Atribua o seletor correto à constante '**Element**' para obter o retorno "**Subtítulo**" no console.



```
<title>
```

MÃO + + NA MASSA

```
</title>
```

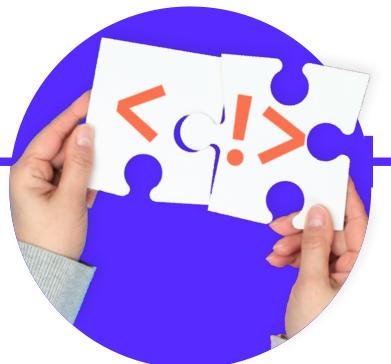
Dado o seguinte HTML:

```
<!DOCTYPE html>
<html>

<head>
    <title>Acessando o DOM</title>
</head>

<body>
    <div class="box">
        <h1 id="title">Titulo</h1>
        <h2>Subtitulo</h2>
    </div>
</body>

</html>
```



- Escreva um script que retorne a quantidade de elementos com a tag “**span**” no console.
- Escreva um script que selecione o segundo elemento com a tag “**span**” e retorne no console “**Texto 2**”, que é seu conteúdo de texto.
- Escreva um script que selecione o elemento com a tag “**input**” e retorne no console seu valor (“**john doe**”).



+

+

+

+

+

+

+



A arte de programar consiste
em organizar e dominar
a complexidade.

Edsger W. Dijkstra

{02.}

Coletando e exibindo valores na tela



A primeira tarefa que vamos aprender é como coletar elements e nodes do HTML para dentro do JavaScript.

Para fazermos isso, teremos ajuda principalmente dos **métodos do objeto document**.

Exemplo de estrutura HTML para facilitar o entendimento dos métodos get do objeto document:

```
Console + ●

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Aula 2 - Coletando e exibindo valores</title>
8  </head>
9  <body>
10     <h1 id="titulo">Minha página</h1>
11
12     <section class="textos">
13         <h2>Sobre mim</h2>
14         <p>Texto sobre a pessoa aqui.</p>
15     </section>
16
17     <section class="textos">
18         <h2>Meus projetos</h2>
19         <ul>
20             <li class="projeto">Projeto 1</li>
21             <li class="projeto">Projeto 2</li>
22             <li class="projeto">Projeto 3</li>
23             <li class="obs">Observação</li>
24         </ul>
25     </section>
26
27     <script src="exemplos.js"></script>
28
29 </body>
30 </html>
```

GetElementById()

O **getElementById()** é um método que busca o elemento no DOM, que possui o id especificado como parâmetro.



GetElementsByTagName()

O método `getElementsByTagName()` retorna todos os elementos que possuam a mesma tagName do que a especificada como parâmetro.

Diferentemente do `getElementById()`, que retorna apenas um elemento, nesse caso podemos ter múltiplos elementos retornados ou apenas um. Caso sejam retornados mais de um elemento, eles ficam dispostos na variável em forma de array.

GetElementsByClassName()

O método `getElementsByClassName()` retorna os elementos pertencentes a uma mesma classe, especificada como parâmetro do método.

Normalmente o resultado é uma array.

QuerySelector()

O `querySelector()` retorna o primeiro elemento que possua a especificação passada como parâmetro. Ele aceita tanto tagName quanto class. Mas como traz um só resultado, ele não cria um array.

QuerySelectorAll()

O `querySelectorAll()` é parecido com `querySelector()`, mas ele retorna todos os elementos que atendam às especificações passadas como parâmetro, sendo retornados em forma de array.

É uma alternativa ao `getElementsByClassName()` e ao `getElementsByTagName()` e permite combinar tags e classes.



Para entender como podemos modificar a exibição dos danos no HTML, vamos conhecer duas propriedades dos nodes: **textContent** e **innerText**.

textContent

Focando inicialmente no **textContent**, ele representa o conteúdo de texto de um node e seus descendentes. Dessa forma, ele tem como valor de retorno uma string ou null.

innerText

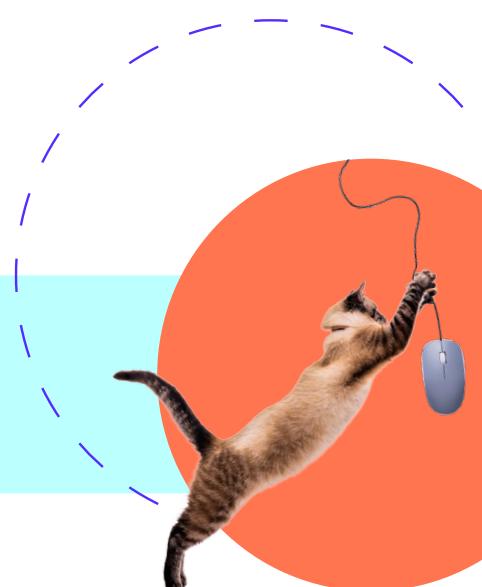
Tem um efeito parecido com o do **textContent**, porém a finalidade é modificar a exibição dos dados no HTML.

```
<title>
```

PULO DO GATO

```
</title>
```

- **textContent** pode ser utilizado em qualquer tipo de node.
- **innerText** é utilizada apenas com elementos HTML.





```
<title>
```

MÃO + + NA MASSA

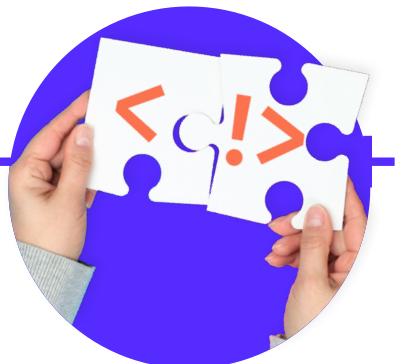
```
</title>
```

Com base no HTML a seguir:

```
<!DOCTYPE html>
<html>

<head>
    <title>Coletando Dados</title>
</head>

<body>
    <div class="userdata">
        <label for="username">Usuário:</label>
        <input type="text" id="username" value="Antonio Lucio
Vivaldi" />
        <label for="userpassword">Senha:</label>
        <input type="password" id="userpassword"
value="LeQuattroStagioni" />
    </div>
</body>
</html>
```



Escreva um script que retorne as informações do usuário (username e password) em uma única linha, como no exemplo. Exemplos:

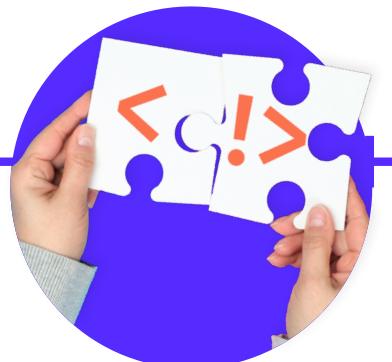
- “Usuário: John Doe, Senha: minhasenha”.
- “A senha do usuário John Doe é minhasenha”.



```
<title>
```

MÃO + + NA MASSA

```
</title>
```



Com base no HTML a seguir:

```
<!DOCTYPE html>
<html>

<head>
    <title>Dados</title>
</head>

<body>
    <fieldset>
        <label for="zipCode">Código Postal:</label>
        <input type="text" id="zipCode" />
        <label for="address">Logradouro:</label>
        <input type="text" id="address" />
        <label for="number">Número:</label>
        <input type="text" id="number" />
        <label for="complement">Complemento:</label>
        <textarea id="complement"></textarea>
    </fieldset>
</body>

</html>
```

Selecione os elementos de entrada de dados (input) e preencha com os dados de endereço.



{03.}

Trabalhando com campos de texto e botões

Vamos conhecer outros
recursos?

O **Input Text Object** no HTML DOM
é usado para representar o elemento
HTML <input> do tipo “text”.

Vamos relembrar rapidamente os
tipos de <input> que encontramos
no HTML?



Types	Significados
text	texto - valor padrão
color	cor
date	data
email	email
image	imagem
number	número
password	senha
submit	enviar
url	url

Property	Description
disabled	Define ou retorna se o campo de texto está desativado ou não.
size	Define ou retorna o valor padrão do campo de texto.
defaultValue	Define ou retorna o valor size (tamanho) do campo de texto.



Além das propriedades, ele também possui

3 métodos principais:

Property	Description
blur()	Remove o foco do campo de texto (retira o cursor do campo).
focus()	Dá o foco ao campo de texto (insere o cursor para escrita no campo de texto).
select()	Seleciona o conteúdo (.value) do campo de texto.

Vamos ver um exemplo e para isso vamos usar um HTML com um campo de `<input>`.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>
5          Aula 3 - Input e Button
6      </title>
7  </head>
8  <body>
9      <h1>Conquer</h1>
10     <h2>DOM Input Text Object</h2>
11     <input type="text" id="text_id" value="Hello Conquer!">
12
13     <p id="output"></p>
14
15     <script src="exemplos.js"></script>
16 </body>
17
18 </html>
```



No nosso arquivo de JS, vamos inserir a função `myInput()`.

```
Console +  
1 function myInput() {  
2     var txt = document.getElementById("text_id").value;  
3 }
```

Console +

Conquer

DOM Input Text Object

Assim, o campo funciona corretamente, permitindo escrever um texto.

Por meio do JS também atribuímos esse texto inserido à variável txt.

Para podermos fazer o trigger de executar a função `myInput()` e retornar esse valor de volta para outro campo do HTML, utilizamos um recurso conhecido como **button**.

O objeto Button representa o elemento HTML `<button>`, é possível criar um objeto button com o JavaScript por meio da seguinte sintaxe:

```
Console +  
1 var btn = document.createElement("BUTTON");
```



Assim como o objeto Input, ele possui algumas propriedades.
Vejamos elas:

Property	Description
disabled	Define ou retorna se o botão está desabilitado ou não.
value	Define ou retorna o valor do atributo type do botão.
form	Retorna a referência do form que contém o button.

```
<title>
```

PULO DO GATO

```
</title>
```

- › É comum utilizar a propriedade **onClick** na tag <button> do HTML para adicionar funcionalidade ao botão.

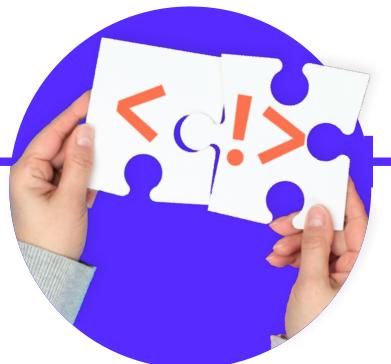




```
<title>
```

MÃO + + NA MASSA

```
</title>
```



Dado o HTML, obtenha as seguintes requisições.

```
<html>
  <head>
    <title>Input</title>
  </head>

  <body>
    <fieldset>
      <label for="volume">Volume: (%)</label>
      <input type="range" id="volume" value="0" />
      <hr />
      <label for="music">Música favorita:</label>
      <input type="text" id="music" />
    </fieldset>
  </body>
</html>
```

Perceba que o código HTML não tem coerência, sabendo disso, por meio de um script:

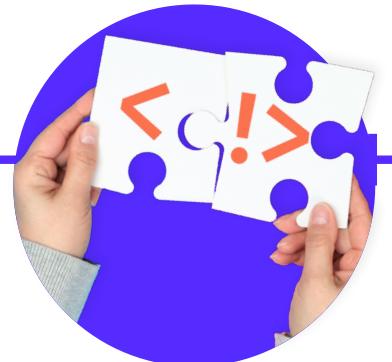
- Insira um novo valor para o “**volume**”.
- Exiba o valor atual do “**volume**” no elemento “**span**” (Ex: 50%).
- Insira o nome de uma música no campo correspondente.



```
<title>
```

MÃO + + NA MASSA

```
</title>
```



Dado o HTML, obtenha as seguintes requisições.

```
<html>
  <head>
    <title>Input</title>
  </head>

  <body>
    <fieldset>
      <label for="volume">Volume: (%)</label>
      <input type="range" id="volume" value="0" />
      <hr />
      <label for="music">Música favorita:</label>
      <input type="text" id="music" />
    </fieldset>
  </body>
</html>
```

Escreva uma função que realize a soma dos valores dos dois primeiros campos e retorne o resultado no terceiro.

Altere o HTML para que o elemento “button” execute a função ao ser clicado.

Remova a alteração anterior e faça o botão executar a função somente através do script.



{04.}

Trabalhando dinamicamente com HTML

Através do DOM, podemos criar novos elementos dentro do HTML.

Pode ser um elemento **dentro do body** ou um elemento “filho” de alguma tag que já **esteja presente no corpo do HTML**.

O método **createElement** auxilia muito nessa criação de elementos, podendo criar uma página inteira em HTML apenas com JavaScript.



Existem vários métodos que parecem muito similares, mas possuem certas particularidades.

Aqui vamos ver mais resumidamente. Para se aprofundar, é importante ver e rever os exemplos passados na videoaula.



1 innerText

O método **innerText** adiciona um texto para o elemento que for selecionado, porém será apenas o texto, sem qualquer tipo de formatação ou elementos HTML.



2 textContent

O método **textContent** retorna o texto com as formatações e sem o HTML.

3 innerHTML

O método **innerHTML** é um muito similar aos os métodos **innerText** e **textContent**. Porém, além de retornar o texto, o DOM retorna também com as formatações e com os elementos HTML.



4 outerHTML

O método **outerHTML** é similar ao **innerHTML**, porém em vez de retornar um texto com as formatações e propriedades, retorna ou “seta” o elemento HTML, incluindo seus atributos, além das tags iniciais e finais.



5 `createTextNode`

Um método muito similar ao `outerHTML` é o `createTextNode`, possuindo apenas uma forma de aplicação diferente.

6 `insertAdjacentText`

O método `insertAdjacentText` além de adicionar o texto, pede a posição na qual você deseja adicionar o texto.

Existem posições válidas que você pode utilizar neste método. Como por exemplo:

“afterbegin”

Esta posição indica que começa antes do início do primeiro elemento (primeiro filho).



“beforebegin”

Esta posição indica que é um elemento anterior ao elemento.

“beforeend”

Esta posição indica que é um elemento posterior ao último elemento (último filho).



7 `insertBefore`

O método `insertBefore` insere um elemento filho, antes de um elemento filho já existente.



8 document.write

O método **document.write** pode ser utilizado de três formas diferentes:

- a) A primeira forma é **escrevendo no HTML**, independentemente da formatação, imprimindo aquele texto de forma direta.
- b) A segunda forma é **adicionando os elementos HTML**, sendo assim, passando uma informação, porém com as tags de início e fim.
- c) A terceira forma é **imprimindo um texto, porém excluindo todo o HTML** presente anteriormente e retornando apenas o que está escrito neste método.

Existem dois métodos muito utilizados quando pensamos no DOM. E são conhecidos como **append** e **appendChild**.

Esses dois métodos abordados possuem algo em comum. Todos servem para elementos “filhos” e o programador consegue utilizar cada um de acordo com a sua necessidade.

Mas como fazemos para remover elementos? Temos três possibilidades:

1 removeChild

Com o **removeChild**, você basicamente remove um elemento que esteja presente dentro daquela tag e que seja um elemento “filho”.



2 replaceChild

Com o método **replaceChild**, ao contrário do método anterior, o elemento não será removido e sim, alterado para outro elemento e/ou outro texto.

3 replaceWith

Já o **replaceWith** é um método em que o programador consegue substituir mais de um elemento filho, caso seja necessário.

```
<title>
```

PULO DO GATO

```
</title>
```

- Use **replaceWith**. Isso agiliza o processo do método **replaceChild**, caso você queira alterar de maneira mais rápida e eficiente.





+

+

+

+

+

+

+

“



Não há nada tão inútil quanto fazer eficientemente o que não deveria ser feito.

Peter Drucker

”

<title>

PULO DO GATO

</title>

- Os atributos em HTML servem para **colocar informações adicionais** para os elementos HTML.

No caso de um elemento img,
podemos ter os seguintes atributos:

- alt
- width
- src
- height





Já no caso do método **setAttribute**, o programador vai conseguir incluir um atributo para aquele elemento, além de passar o valor que deseja.

Já ao **adicionar um atributo classe**, ele pode adicionar o nome dessa classe.

```
<title>
```

MÃO + NA MASSA

```
</title>
```



Com base no código HTML a seguir:

```
<html>

    <head>
        <title>Input</title>
    </head>

    <body>
        <fieldset>
            <label for="volume">Volume: (%)</label>
            <input type="range" id="volume" value="0" />
            <hr />
            <label for="music">Música favorita:</label>
            <input type="text" id="music" />
        </fieldset>
    </body>

</html>
```



Insira alguns elementos dinamicamente.

- Campo inserção de texto (tipo: text).
- Três botões.
- Elemento de texto.
- Elemento de caixa de seleção (tipo: checkbox).
- O primeiro botão deverá adicionar um elemento à lista com o texto presente no campo de inserção de texto somente se o mesmo não estiver vazio.
- Após a adição do elemento, o campo de inserção de texto deve ficar vazio.
- O segundo botão deverá contar os itens da lista e exibir o resultado no elemento de texto.
- A caixa de seleção servirá para indicar se o elemento deve ser adicionado ao topo ou ao final da lista (se estiver marcada, adicionar ao topo).
- O terceiro botão deve excluir o último elemento da lista.



{05.}

Customizando estilos

Assim como manipulação do HTML, é possível criar, alterar e remover o CSS através do DOM.

Podemos criar o elemento style para ter a tag no html, como também acessá-lo através do DOM, além de conseguir setar diretamente algum elemento.

Os elementos possuem por padrão o elemento nó style como filho.
É com este elemento que conseguimos manipular o CSS da página.



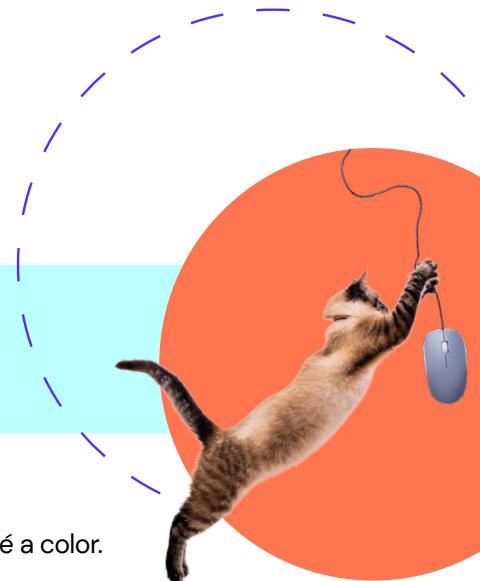
<title>

PULO DO GATO

</title>

- O **style**, através do DOM, precisa ser passado em string.

Uma das propriedades que pode ser alterada é a color.



Console

```
1 function styleButton() {  
2   const button = document.querySelector('.btn-login');  
3   button.style.color = 'blue';  
4   button.style.margin = '10px';  
5   button.style.width = '150px';  
6   button.style.borderRadius = '10px';  
7   button.style.backgroundColor = 'yellow';  
8 }
```



Console +

```
1 const telaLogin = document.getElementById('tela-login');
2 telaLogin.style.backgroundColor = '#bbffcc';
```

Quando se tem alguma propriedade que possui hífen, o programador deve usar **camelCase**.

Seguindo nas propriedades de CSS, temos o **display** que também podemos alterar por meio do DOM.

Console +

```
1 const telaLogin = document.getElementById('tela-login');
2 telaLogin.style.display = 'flex';
```

Assim como no CSS, com **border** você adiciona uma borda. Por meio do DOM, o programador coloca todas as propriedades em uma única string, como no exemplo abaixo:

Console +

```
1 const telaLogin = document.getElementById('tela-login');
2 telaLogin.style.border = '5px solid yellow';
```

No DOM, assim como no CSS, há uma forma de alterar os tamanhos da **altura e da largura**. Por ser no JavaScript, também deve ser passado como string.

Console +

```
1 const telaLogin = document.getElementById('tela-login');
2 telaLogin.style.height = '500px';
3 telaLogin.style.border = '500px';
```



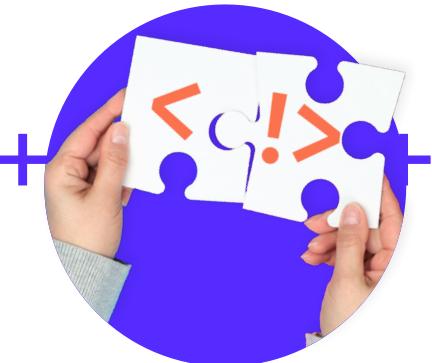
```
<title>
```

MÃO + + NA MASSA

```
</title>
```

Com base no código HTML a seguir:

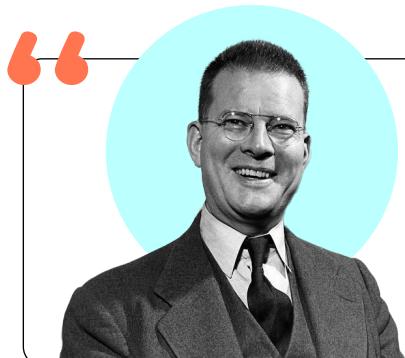
```
<!DOCTYPE html>
<html>
  <head>
    <title>Customizando Estilos</title>
  </head>
  <body>
  </body>
</html>
```



Crie um script para obter o resultado mostrado nas imagens a seguir.

Dicas (Cores, medidas em "rem" etc.):

- #264653 • #E76F51 • 1rem
- #2A9D8F • sans-serif • 2rem
- #FCFCFC • 10rem • 5rem



+

+

+

+

+

+

Melhorando a qualidade, automaticamente você estará melhorando a produtividade.

W. Edwards Deming



{06.}

Trabalhando com formulários

Com a ajuda do método **createElement**, vamos criar a tag form no HTML.

A partir disso, vamos criar os **inputs** que serão os filhos do elemento tag. Usando o **setAttribute**, vamos setar **placeholders**, **types** e **outros atributos** que achamos necessários para o formulário.

Existem duas maneiras de acessar os dados inseridos no formulário e verificar se eles estão sendo passados de forma correta, antes de serem enviados para alguma API.

1 **console.log**

Uma dessas maneiras é utilizando o **console.log** para acessar via console da aplicação, seja no VSCode ou na própria aplicação rodando, como na imagem.



```
▼ Object [i]
  email: "joao.mendes@email.com"
  nome: "João"
  senha: "123456789"
  sobrenome: "Mendes"
  whatsapp: "(21)956324853"
▶ [[Prototype]]: Object
```

[index.js:154](#)

2 alert

A segunda é utilizando o **alert**:

The screenshot shows a browser's developer tools console window. The title bar says 'Console' and has a '+' button. The code in the console is:

```
1 alert(JSON.stringify({
2   nome: form.name.value,
3   sobrenome: form.lastname.value,
4   email: form.email.value,
5   whatsapp: form.whatsapp.value,
6   senha: form.password.value
7 }))
```



```
<title>
```

PULO DO GATO

```
</title>
```

- Prefira usar a segunda opção: o alerta com **stringify**.
- Quando quiser mudar a forma para `console.log`, você muda somente em um lugar: dentro do método **log()**.



No HTML, aprendemos a utilizar o atributo **required** para requisitar que aquele input seja preenchido para que o formulário seja enviado. Vamos ver como isso funciona no JavaScript?

No JavaScript podemos fazer a mesma validação, mas sem usar o atributo. Utilizamos, então, uma **condicional** para verificar se aquele campo está preenchido de maneira correta ou não.

Quando há um **envio de informações**, os dados são enviados para uma **API**.

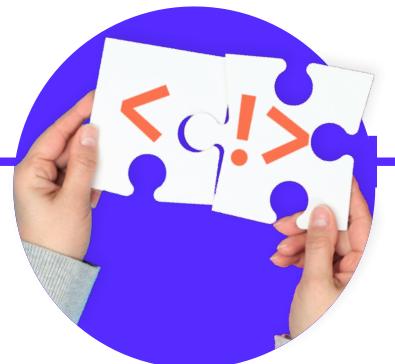
Porém, como não há consumo de API neste formulário básico, podemos conferir a confirmação das informações enviadas por meio do `console.log`, que auxilia caso tenhamos que alterar alguma coisa antes de realmente enviar para alguma API.



```
<title>
```

MÃO + + NA MASSA

```
</title>
```



1) Crie um formulário a partir do DOM, com os seguintes inputs (caso queira incluir mais inputs, fique à vontade):

- a) nome
- b) sobrenome
- c) e-mail
- d) número
- e) gênero
- f) checkbox para salvar as informações
- g) checkbox com os Termos de Uso
- h) botão para envio

- 2) Para conseguir enviar as informações, valide todos os inputs necessários, inclusive os checkboxes.
- 3) Imprima os valores dos itens “a” ao “e”, caso haja mais inputs que tenha adicionado, imprima-os também.
- 4) Crie um alert para os itens criados.

```
<!DOCTYPE html>
<html>

    <head>
        <title>Formulário em JavaScript</title>
    </head>

    <body>
    </body>

</html>
```



```
<title>
```

DESAFIO CONQUER

```
</title>
```

```
<body>
```

Crie um clone do Twitter!

- Não deve criar tags HTML na mão.
- Não deve conter estilo CSS importado ou inline.
- Toda tag HTML deverá ser criada via JavaScript.
- O desafio deverá conter:
 - Formulário.
 - Campo de texto.
 - Botão submit.
 - Lista de tweets publicados.
 - Mensagem para mais de 280 caracteres.



```
if (challenge)  
// start
```

```
</body>
```



<title>

DESAFIO CONQUER

</title>



<body>

A ordem em que os tweets deverão aparecer precisa respeitar: tweets mais recentes primeiramente e tweets mais antigos por último.

O padrão do tweet que aparecerá na listagem deverá ser:

- Círculo de avatar com fundo de qualquer cor que possa escolher e letras iniciais do seu nome.
- Seu nome à frente do avatar.
- Uma divisória separando o cabeçalho do tweet do corpo do tweet.
- Corpo do tweet com o texto informado por você.

```
if (challenge)
  // start
```

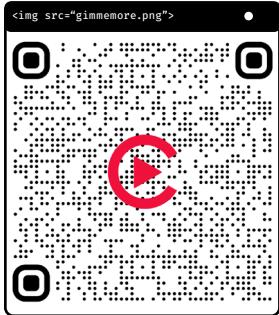
</body>



<title>

QUERO MAIS

</title>



Leia o artigo sobre DOM



Leia mais esse artigo sobre DOM



```
if (interested)
    // gimmemore
```





Anotações

- +
- +
- +
- +
- +
- +

010101
0001001100
0100010000001010
0101000101010010
00000100010010101000
10100001010101000100
10001010011100101110010
11110010110000101100101010