Javascript Front-end: Manipulando Elementos

DigitalHouse>

JAVASCRIPT NO FRONT-END

É UM SUBSTITUTO AO

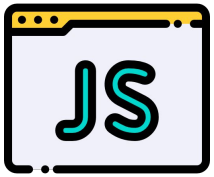
CSS ANIMATION?

Você se lembra...

Como vincular um script ao documento HTML?

- Um arquivo externo
  ```
  <script src="caminho_do_arquivo.js"></script>
  ```

- Presente no mesmo arquivo
  ```
  <script>  código js  </script>
  ```

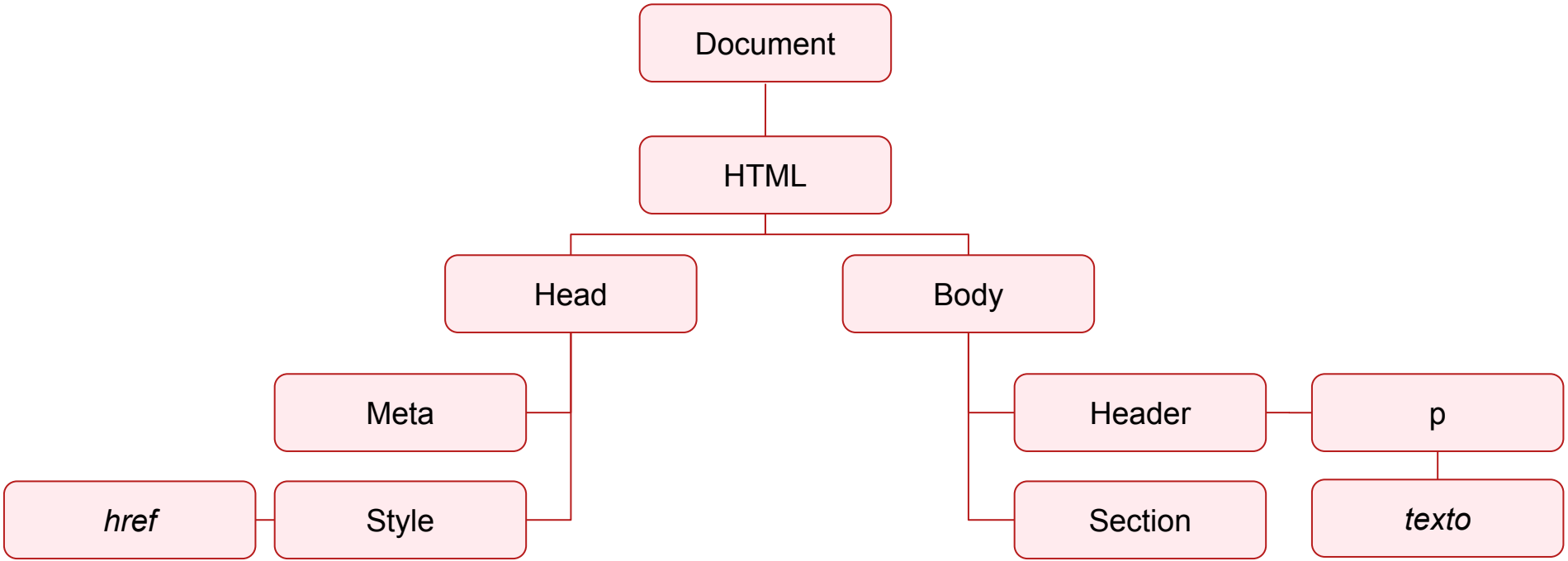


Você se lembra...

Da importância do Objeto Document para o front end?

- Permite acesso para **leitura** e **modificação** dos elementos contidos na janela **(*Window*)**
- Também popularmente conhecido como **DOM** ***(Documento Object Model)***

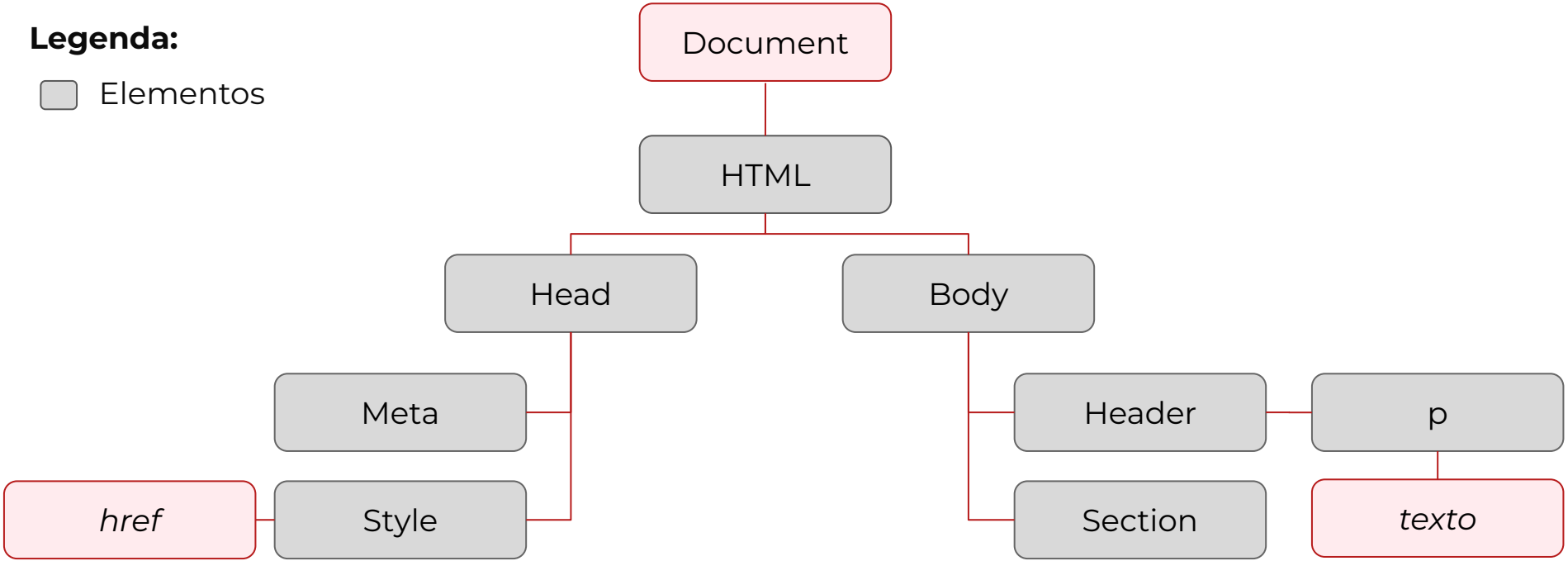
Representação Árvore DOM



Representação Árvore DOM

Legenda:

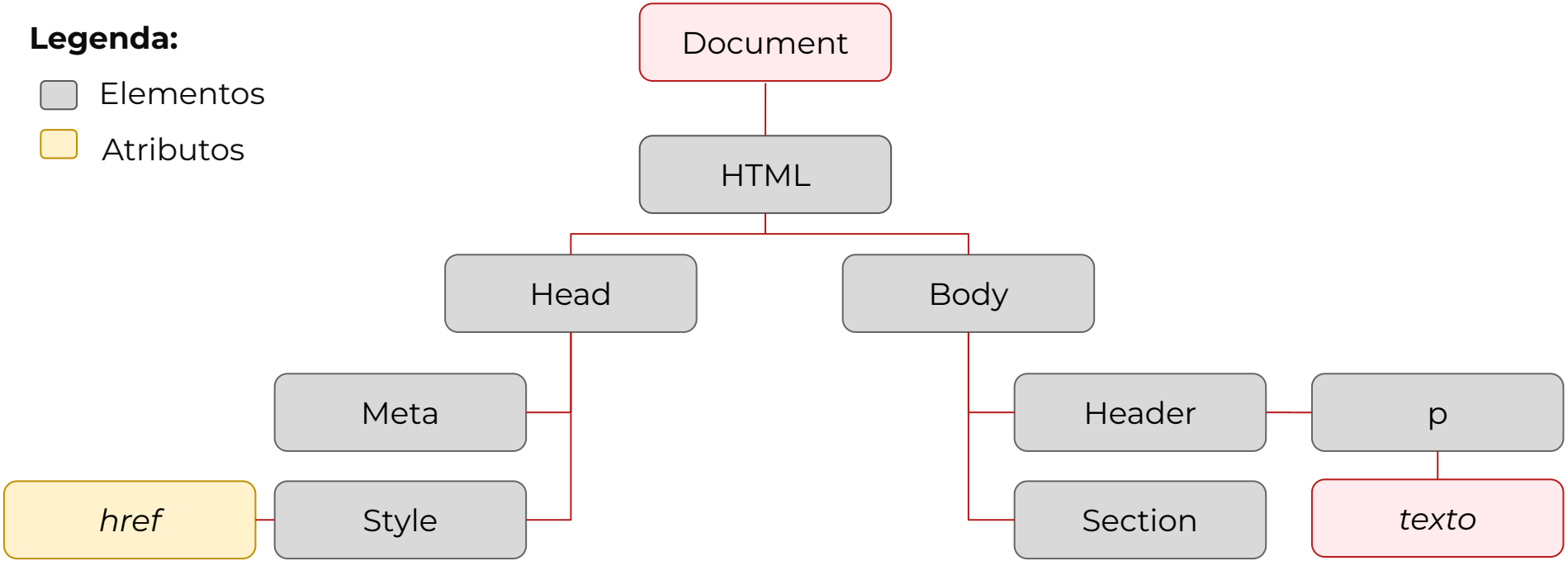
- Elementos



Representação Árvore DOM

Legenda:

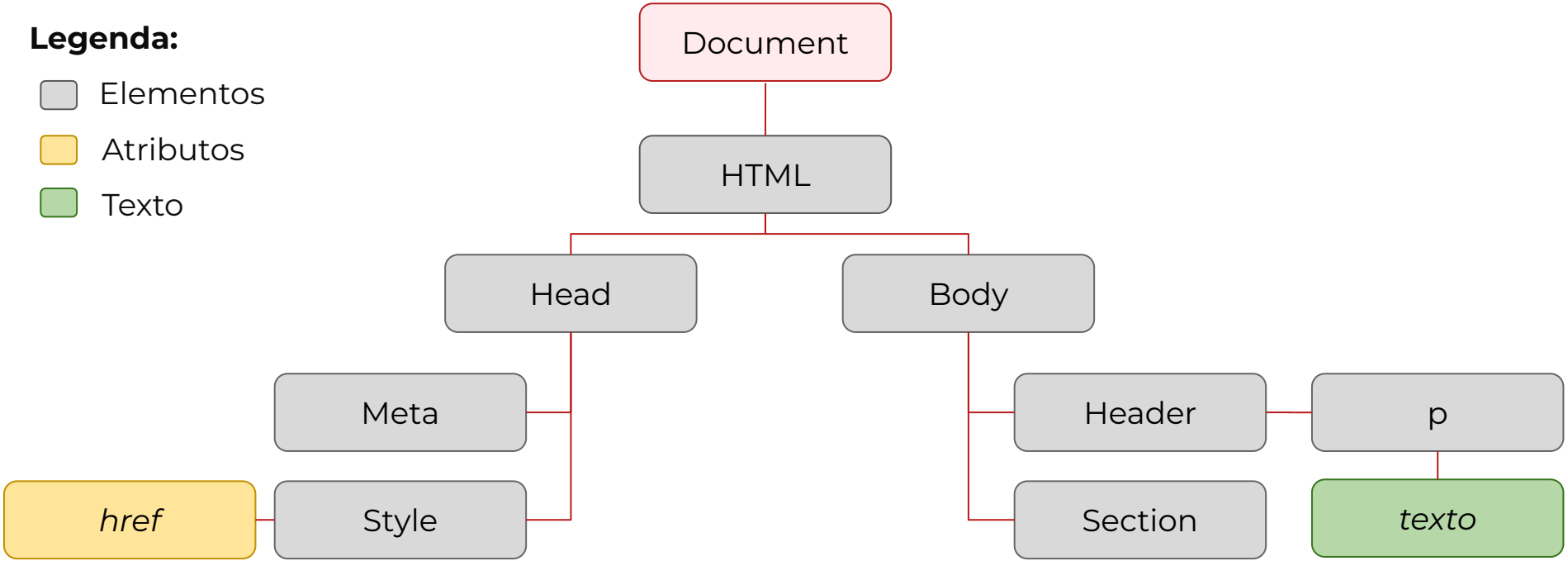
- Elementos
- Atributos



Representação Árvore DOM

Legenda:

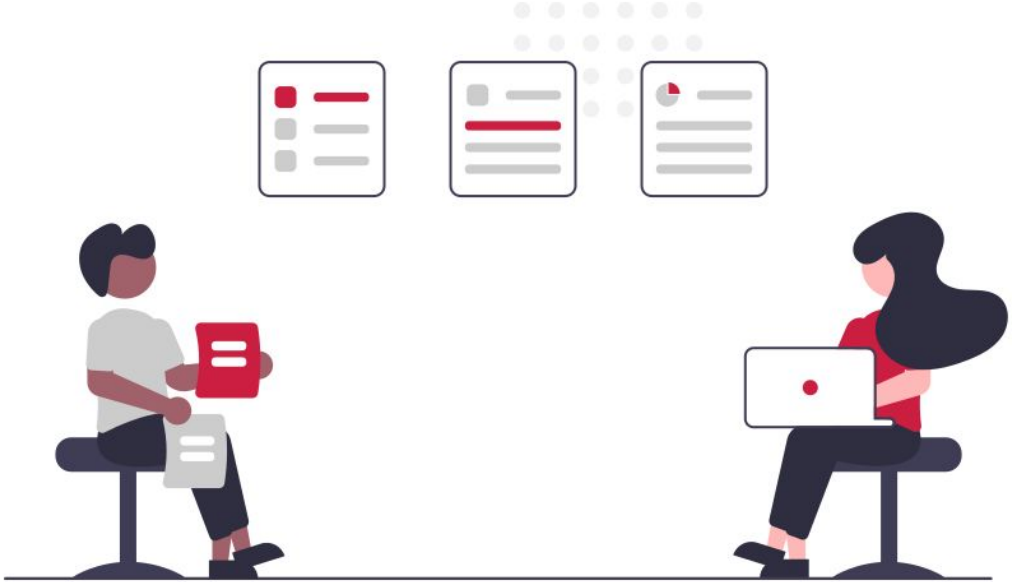
- Elementos
- Atributos
- Texto



Formas de selecionar:

querySelector()	seleciona primeiro elemento indicando o **seletor css + nome** (***.box***, ***#logo***, ***div***)
getElementById()	seleciona primeiro elemento indicando o **id**
querySelectorAll()	seleciona **lista de elementos** com base no **seletor css** indicado
getElementByTagName()	lista de elementos com base na tag indicada

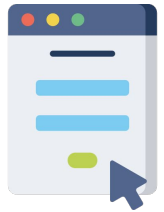
Bora praticar!



Atividade I - Adicionando estruturas HTML

Chegou uma demanda nova nessa semana: trabalharmos na construção de uma rede social.

Os conhecimentos adquiridos serão necessários para cumprir com os requisitos da tarefa dessa semana.



Veja mais detalhes no documento **Atividade I**.

Formas de Modificar o DOM:

selecionado.innerHTML permite criar, visualizar ou substituir conteúdo HTML de uma estrutura selecionada:

```
let verConteudo = selecionado.innerHTML
```

ou

```
selecionado.innerHTML =  "<p> novo conteúdo </p>"
```

selecionado.innerText permite criar, visualizar ou substituir conteúdo em texto de um elemento/tag:

```
let verConteudo = selecionado.innerText
```

ou

```
selecionado.innerText =  "novo conteúdo"
```

Formas de Modificar o DOM:

selecionado.style.elementoCSS permite alterar as propriedades CSS e seus valores para uma estrutura selecionada.

➔ seguindo o padrão **camelCase**, as nomenclaturas de propriedades perdem o hífen **-**

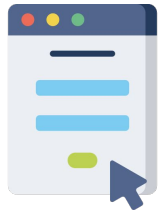
```
selecionado.style.borderColor = 'blue'
```

selecionado.style.cssText permite alterar uma lista de estilos em um único comando. Basta adicionar a listagem como é feito na linguagem CSS:

```
selecionado.style.cssText = "color:black; display: flex;
background-color:purple"
```

Atividade II - E quando não há acesso ao arquivo CSS?

- Altere/adicione propriedades CSS por meio da edição do DOM



Veja mais detalhes no documento **Atividade II**.

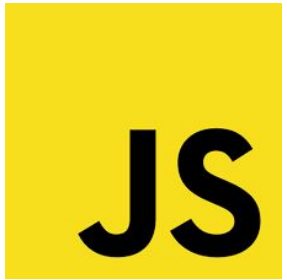
Para ir além

➔ **Documentação no site oficial MDN:**

https://developer.mozilla.org/pt-BR/docs/Web/API/Document_Object_Model/Introduction

https://developer.mozilla.org/pt-BR/docs/Web/API/Document/querySelector

https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/style



JS Front Vinculação

DigitalHouse >
Coding School

Índice

1 Vinculação Interna

JS Front

Vinculação Interna

Permite escrevermos o código Js diretamente na página HTML.

MAS, NÃO É O MAIS INDICADO!

{}

```
<body>
    ...
    <script>
        console.log(“Olá Mundo!”);
    </script>
</body>
```

2 Vinculação Externa

JS Front

Vinculação externa

Permite ligar nosso arquivo HTML com um arquivo JS externo.

```
<body>
    ...
  <script src="js/main.js"></script>
</body>
```

JS Front

Vinculação externa

Lembre-se de que, com o uso de links (vinculação) externos, não é necessário escrever as tags <script> dentro do nosso arquivo **.js**.

{}

```
let saudacao = 'Olá mundo!';


   console.log(saudacao);
```

JS Front Seletores

DigitalHouse >
Coding School

Índice

> Para acessar os elementos de uma página, usamos **seletores**. Cada seletor pode retornar **um único elemento ou uma lista de elementos**.

> Para fazer uso dos seletores, devemos fazer uso do objeto **document**, uma vez que os seletores são métodos dele.

1 querySelector()

JS Front

querySelector()

Esta função recebe uma String que indica o seletor CSS que estamos procurando.

Por exemplo:

```
{}  let titulo = document.querySelector('.title');
```

Ele retornará o **primeiro** elemento do HTML que contém a classe "título".

É importante declarar uma variável para armazenar os dados que o seletor nos traz, caso contrário, o perderíamos quando o programa prosseguir.

2 querySelectorAll()

JS Front

querySelectorAll()

Esta função recebe uma String que indica o seletor CSS que estamos procurando.

Por exemplo:

```
{}  let nombres=document.querySelectorAll('.name');
```

Ele retornará uma **lista** de elementos que correspondem à pesquisa especificada.

Também podemos usar seletores diretamente com elementos do documento, por exemplo:

```
let div=document.querySelectorAll('div');
```

3 | getElementById()

JS Front

getElementById()

Esta função recebe uma String com apenas o nome do **id** do elemento DOM que estamos procurando.

Por exemplo:

```
{}  let marca=document.getElementById('marca');
```

Ele retornará o elemento cujo **id corresponde ao desejado.**

Também podemos pesquisar elementos por seu id usando os seletores anteriores, mas devemos acrescentar um # para esclarecer que se trata de um id.

```
let marca=document.querySelector('#marca');
```

JS Front

Comparando Selectores

querySelector()	querySelectorAll()	getElementById()
Retorna o **primeiro elemento** do DOM que atende à condição que estamos procurando.	Retorna **todos os elementos** do DOM que atendem à condição que procuramos.	Retorna o elemento do DOM **que atende ao Id que estamos procurando**.

JS Front
Modificando o DOM

DigitalHouse >
Coding School

“Para fazer modificações no DOM, precisamos ter o objeto que queremos modificar selecionado. Podemos fazer isso usando seletores!”

Índice

1 .innerHTML

JS Front

.innerHTML

Se quisermos **ler ou modificar** o conteúdo de uma tag HTML, vamos usar esta propriedade:

```
{}  document.querySelector('div.nome').innerHTML ;
```

Nesse caso, nós a usamos para ler o conteúdo. Mas e se quisermos modificá-la?

Se quisermos **salvar** o valor, devemos atribuir essa linha de código a uma variável. Caso contrário, quando a execução continuar, o valor que procuramos será perdido.

JS Front

.innerHTML

Se quisermos **modificar** o conteúdo de uma tag HTML, vamos usar esta propriedade:

```
{}  document.querySelector('div.nome').innerHTML = '<strong>Daniel</strong>';
```

Se usarmos a propriedade dessa forma, todo o conteúdo que tínhamos na div com o nome da classe será alterado para a string "Daniel".

No entanto, também podemos modificar o conteúdo sem perder o que tínhamos anteriormente.

JS Front

.innerHTML

Se quisermos **modificar** o conteúdo de uma tag HTML, vamos usar esta propriedade:

```
{}  document.querySelector('div.compras').innerHTML += '<i>Pipoca</i>';
```

Dessa forma, estamos adicionando ao div com a classe de compras, a palavra "Pipoca" de tal forma que, se a lêssemos, diria:

```
{}  <div class="compras"> "Presunto, Queijo, Pão" <i>Pipoca</i></div>
```

Conteúdo anterior

Conteúdo adicionado

2 .innerText

JS Front

.innerText

Se quisermos **ler ou modificar** o texto de uma tag HTML, vamos usar esta propriedade:

```
{}  document.querySelector('div.nome').innerText ;
```

Nesse caso, se no meu div com a classe nome estivesse escrito "Leo", a propriedade retornaria a **String** "Leo".

Se quisermos **salvar** o valor, devemos atribuir essa linha de código a uma variável. Caso contrário, quando a execução continuar, o valor que procuramos será perdido.

JS Front

.innerText

Se quisermos **modificar** o texto de uma tag HTML, vamos usar esta propriedade:

```
{}  document.querySelector('div.nome').innerText = 'Maria';
```

Se usarmos a propriedade dessa forma, todo o texto que tínhamos na div com classe nome será alterado para a string "Maria".

No entanto, também podemos modificar o conteúdo sem perder o que tínhamos anteriormente.

JS Front

.innerText

Se quisermos **adicionar** conteúdo ao texto de uma tag HTML, usaremos esta propriedade da seguinte maneira:

```
{}  document.querySelector('div.nome').innerText += 'Messi';
```

Nesse caso, o que aconteceria é semelhante ao que acontece com o outro seletor, mas o texto seria incluído na tag div, deixando:

```
{}  <div class="nome">Leo Messi</div>
```

Conteúdo anterior

Conteúdo adicionado

3 Propriedade style

JS Front

Propriedade Style

Nos permite ler e sobrescrever as regras CSS que são aplicadas a um elemento que selecionamos.

```
{}
let titulo = document.querySelector('.title');
titulo.style.color = 'cyan';
titulo.style.textAlign = 'center';
titulo.style.fontSize = '12px';
titulo.style.backgroundColor = '#dddddd';
```

Observe que as regras CSS que tinham hifens como **font-size**, em Javascript são escritas em camelCase, ou seja, **fontSize**.

Modificando Estilos

DigitalHouse >
Coding School

PROPRIEDADE STYLE

Ela nos permite ler e sobrescrever as regras CSS que são aplicadas a um elemento que selecionamos.

{}

```
let titulo = document.querySelector(“.title”);

titulo.style.color = “cyan”;
titulo.style.textAlign = “center”;
titulo.style.fontSize = “12px”;
titulo.style.backgroundColor = “#dddddd”;
```

Observe que as regras CSS que possuíam hifens, como font-size, em Javascript são escritas em camelCase, ou seja, fontSize.

Modificando classes

DigitalHouse >
Coding School

Índice

> O Javascript possui uma propriedade e vários métodos que nos permitem realizar diversas ações com o atributo class de um elemento.

1 classList.add()

classList.add()

Adiciona uma nova classe ao elemento selecionado.

```
let citacao = document.querySelector('.cita');
citacao.classList.add('italico');
```

Antes

```
<p class="cita">Três
pratos de trigo para três
tigres tristes.</p>
```

Depois

```
<p class="cita italico">
Três pratos de trigo para
três tigres tristes.</p>
```

2 | classList.remove()

classList.remove()

Remove uma classe existente do elemento selecionado.

```
let citacao = document.querySelector('.cita');
citacao.classList.remove('cita');
```

Antes

```
<p class="cita">Três pratos de trigo para três tigres tristes.</p>
```

Depois

```
<p class="">Três pratos de trigo para três tigres tristes.</p>
```

3 | classList.toggle()

classList.toggle()

Checa se existe uma classe no elemento selecionado. Caso exista, remove a classe do elemento. Caso não, adiciona a classe ao elemento.

```
let citacao = document.querySelector('p');
citacao.classList.toggle('cita');
```

Antes

```
<p class="italico">Três pratos de trigo para três tigres tristes.</p>
```

Depois

```
<p class="italico cita">Três pratos de trigo para três tigres tristes.</p>
```

4 classList.contains()

classList.contains()

Permite perguntar se um elemento possui determinada classe. Retorna um **valor booleano**.

```
{}
let citacao = document.querySelector('.italico');
citacao.classList.contains('cita'); // false
```

```
{}
let citacao = document.querySelector('.italico');
citacao.classList.contains('italico'); // true
```

“Podemos usar o método **.contains** para realizar operações lógicas utilizando **if/else**.”

Resumo

.add()	.remove()	.toggle()	.contains()
Adiciona a classe ao elemento	Remove a classe do elemento	Adiciona a classe, caso ela não exista. Remove a classe, caso ela já exista.	Pergunta se o elemento possui a classe ou não. Retorna um valor booleano.

Javascript Front-end: Eventos

DigitalHouse>

O que vimos no Playground

- O que são eventos?
- Eventos de Mouse
- Eventos de Teclado

O que vamos ver hoje

- Consolidação conteúdo online - Prática

COMO VOCÊ EXPLICARIA O OBJETIVO E UTILIZADE DE EVENTOS JS?

Quais comportamentos/interações da interface de um web site já experimentou que podem ser resultado de

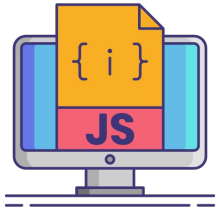
EVENTOS JS?

Eventos

- São ações *server-side* programadas para acontecer
- Possuem um gatilho (tempo, clique, movimentação do mouse, preenchimento de um campo, ...)
- Podem ser executados de duas formas no código:
    - selecionando o item gatilho + nome do evento com o prefixo **on**

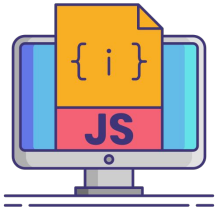
      ```
      menu.onClick = function () {}
      ```
    - item gatilho + addEventListener() e nome do evento como parâmetro

      ```
      menu.addEventListener('click', )
      ```



Eventos importantes:

click	clique do mouse sobre item
mouseOver	mouse sobre o item
mouseOut	ponteiro do mouse fora do item
keyDown	ao pressionar tecla do teclado
keyPress	durante o clique em tecla do teclado

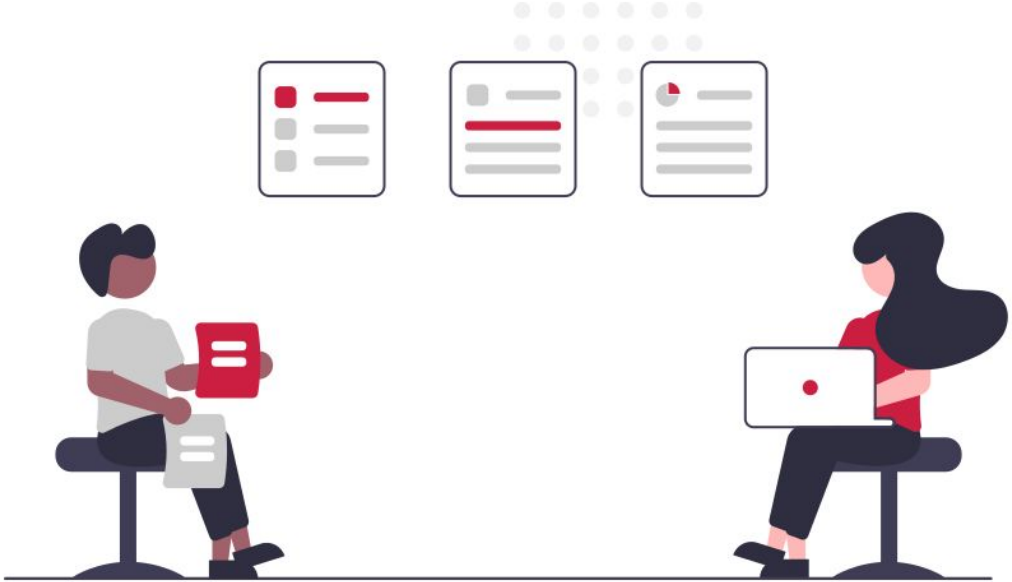


Existe evento que não produz ação na tela?

- **preventDefault()**

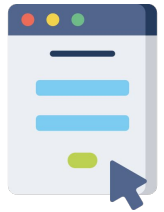
  Este método bloqueia o comportamento padrão de um evento indicado, permitindo que no escopo do callback façamos validações e alterações que julgamos necessárias.

Bora praticar!



Atividade I - Uma Homepage Diferente

Vamos praticar o uso de eventos combinados com alteraçõ̃es no DOM do feed do projeto que estamos desenvolvendo.

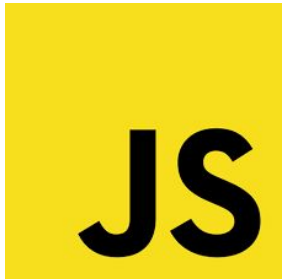


Veja mais detalhes no documento **Atividade I**.

Para ir além

➔ **Documentação no site oficial MDN:**

https://developer.mozilla.org/pt-BR/docs/Web/API/Event



JS Front Aplicando Eventos

DigitalHouse >
Coding School

> Um evento é uma ação que ocorre no navegador e que é disparada pelo usuário.

Índice

1 onload

JS Front

onload

O evento **onload** é um evento que permite que todo o script seja executado quando o objeto de **document** dentro do objeto de window estiver totalmente carregado.

```
{}
window.onload = function(){
    console.log('o documento está pronto');
}
```

O código js geralmente é escrito dentro desta função para **evitar erros** que podem ocorrer se o documento não estiver totalmente carregado quando o script for executado.

2 | onclick

JS Front

onclick

O evento **onclick** nos permite executar uma ação quando o elemento ao qual estamos aplicando a propriedade é clicado.

```
{}
btn.onclick = function(){
    console.log('Você clicou!');
}
```

3 | preventDefault()

JS Front

preventDefault()

O **preventDefault ()** nos permite evitar que o evento padrão ou nativo do elemento ao qual estamos aplicando seja executado.

Podemos usá-lo, por exemplo, para evitar que uma tag "a" se comporte nativamente e faça outra coisa.

Sempre temos que ter selecionado o elemento ao qual queremos aplicar o preventDefault() por meio dos seletores.

JS Front

preventDefault()

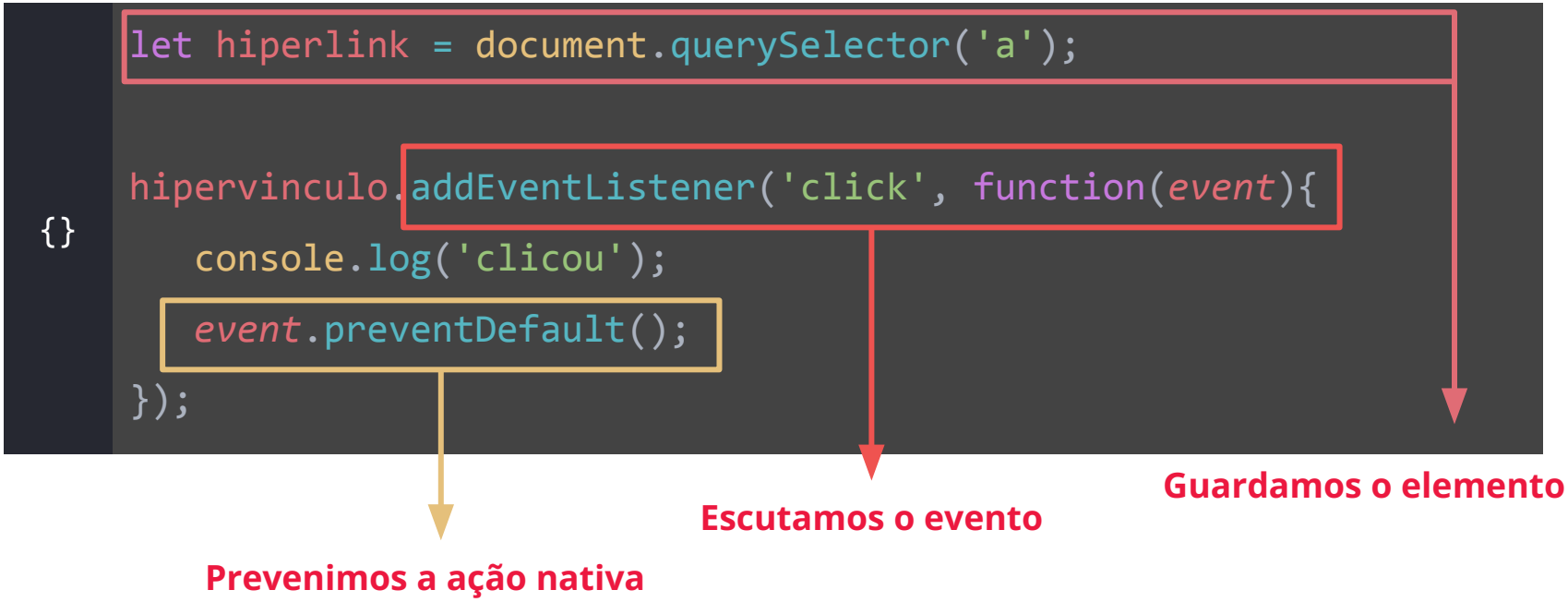
{}

```
let hiperlink = document.querySelector('a');


hiperlink.addEventListener('click', function(event){
   console.log('clicou');
   event.preventDefault();
});
```

JS Front

preventDefault()



JS Front

Eventos mais usados

- Onclick → Quando o usuário clica
- Ondblclick → Quando o usuário faz double click
- Onmouseover → Quando o mouse se move sobre o elemento
- Onmousemove → Quando o mouse se move
- Onscroll → Quando roda o scroll do mouse
- Onkeydown → Quando aperta uma tecla
- Onload → Quando a página carrega
- Onsubmit → Quando enviamos um formulário

Select

Documentação

Para mais informações, visite a documentação oficial, clicando no seguinte Link.

JS Front Aplicando Eventos

DigitalHouse >
Coding School

> Um evento é uma ação que ocorre no navegador e que é disparada pelo usuário.

Índice

1 onload

JS Front

onload

O evento **onload** é um evento que permite que todo o script seja executado quando o objeto de **document** dentro do objeto de window estiver totalmente carregado.

```
{}
window.onload = function(){
    console.log('o documento está pronto');
}
```

O código js geralmente é escrito dentro desta função para **evitar erros** que podem ocorrer se o documento não estiver totalmente carregado quando o script for executado.

2 | onclick

JS Front

onclick

O evento **onclick** nos permite executar uma ação quando o elemento ao qual estamos aplicando a propriedade é clicado.

```
{}
btn.onclick = function(){
    console.log('Você clicou!');
}
```

3 | preventDefault()

JS Front

preventDefault()

O **preventDefault ()** nos permite evitar que o evento padrão ou nativo do elemento ao qual estamos aplicando seja executado.

Podemos usá-lo, por exemplo, para evitar que uma tag "a" se comporte nativamente e faça outra coisa.

Sempre temos que ter selecionado o elemento ao qual queremos aplicar o preventDefault() por meio dos seletores.

JS Front

preventDefault()

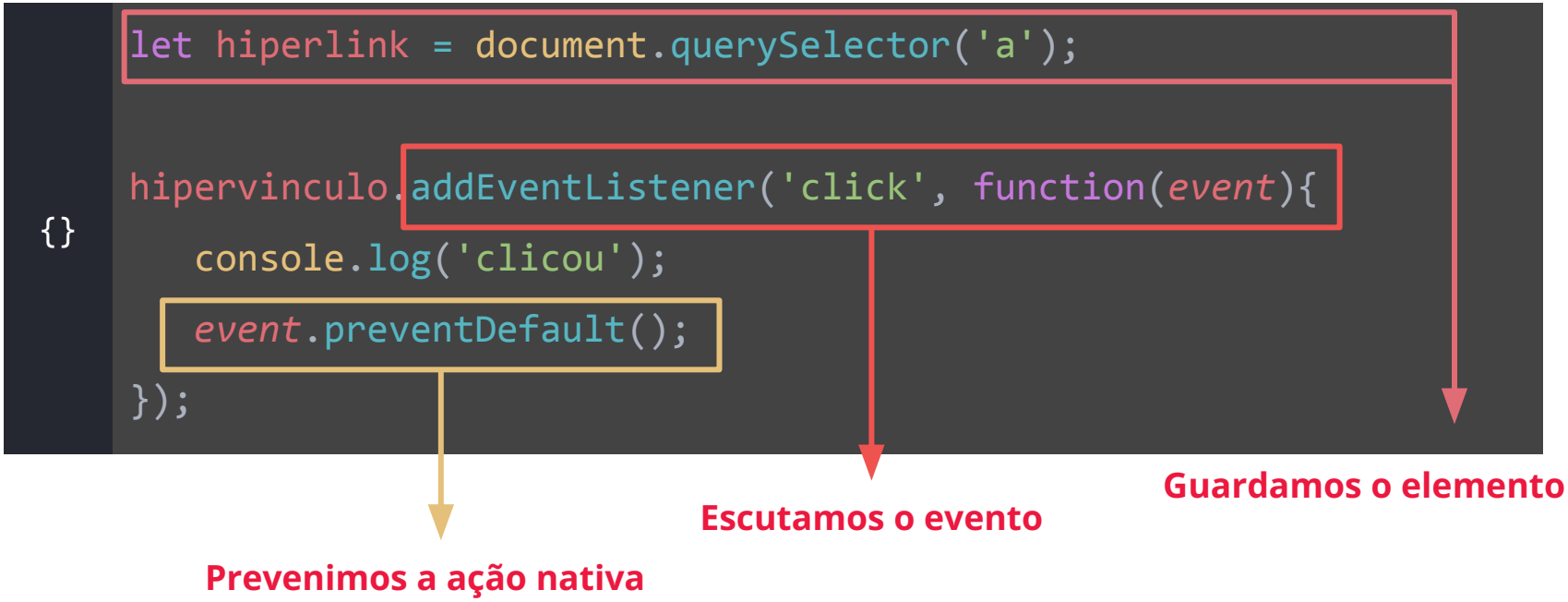
{}

```
let hiperlink = document.querySelector('a');


hiperlink.addEventListener('click', function(event){
   console.log('clicou');
   event.preventDefault();
});
```

JS Front

preventDefault()



JS Front

Eventos mais usados

- Onclick → Quando o usuário clica
- Ondblclick → Quando o usuário faz double click
- Onmouseover → Quando o mouse se move sobre o elemento
- Onmousemove → Quando o mouse se move
- Onscroll → Quando roda o scroll do mouse
- Onkeydown → Quando aperta uma tecla
- Onload → Quando a página carrega
- Onsubmit → Quando enviamos um formulário

Select

Documentação

Para mais informações, visite a documentação oficial, clicando no seguinte Link.

JS Front Eventos de Mouse

DigitalHouse >
Coding School

Índice

1 mouseOver

JS Front

mouseOver

```
let texto = document.querySelector('.text');
texto.onmouseover = function(){
   console.log('Você passou o mouse');
}
```

Também poderíamos fazer...

```
texto.addEventListener('mouseover', function(){
   console.log('Você passou o mouse');
});
```

2 | mouseOut

JS Front

mouseOut

```
let texto = document.querySelector('.text');
texto.onmouseout = function(){
    console.log('Retirou o mouse');
}
```

Também poderíamos fazer...

```
texto.addEventListener('mouseout', function(){
    console.log('Retirou o mouse');
});
```

3 | click

JS Front

click

```
let menu = document.querySelector('.menu');
menu.onclick = function(){
    console.log('Expandir menu hamburguer');
}
```

Também poderíamos fazer...

```
menu.addEventListener('click', function(){
    console.log('Expandir menu hamburguer');
});
```

“Como você deve ter notado, não há diferenças no funcionamento de um evento ao usar o **onEvent** ou **addEventListener**.”

JS Front Eventos de teclado

DigitalHouse >
Coding School

Índice

1 onkeydown

JS Front

onkeydown

O evento **keydown** é executado quando uma tecla é pressionada para baixo. A diferença entre ele e o evento **keypress**, é que o **keydown** é disparado tanto ao clicar em teclas que geram um caractere quanto para as que não geram um (exemplo de teclas que não geram caracteres são: *ALT*, *CTRL* e *Shift*).

{}

```
let input = document.querySelector('#input');
    input.onkeydown = function(event){
        alert("Você pressionou a tecla: "+ event.key);
}
```

2 | onkeyup

JS Front

onkeyup

O evento **onkeyup** é disparado uma única vez quando uma tecla é solta (após clicar na mesma).

```
{}
let input = document.querySelector('#input');
    input.onkeyup = function(event){
        alert("Você soltou a tecla: "+ event.key);
}
```

3 | onkeypress

JS Front

onkeypress

O evento **onkeypress** é disparado enquanto se está pressionando a tecla (funcionará apenas em teclas que geram caracteres).

```
{}
let input = document.querySelector('#input');
   input.onkeypress = function(event){
       alert("Você pressionou a tecla: "+ event.key);
}
```

DigitalHouse>
Coding School