

## 6 - JAVA SCRIPT II

---

### 6.1 - MÓDULO 01

#### 6.1.1 - Adicionar estilos com a propriedade .style

A linguagem JavaScript nos permite acessar e manipular diretamente a estilização de qualquer elemento capturado no DOM usando a propriedade **style**.

##### 6.1.1.1 SETUP DE ARQUIVOS

Para demonstrar a funcionalidade da propriedade style, usaremos um projeto simples com três arquivos como exemplo: index.HTML, style.css e script.js. O arquivo index.HTML segue o padrão da estrutura base HTML, ou seja, possui um único elemento dentro da tag `<body></body>`:

```
<div>

  Texto de exemplo

</div>
```

Além disso, dentro da tag `<head></head>`, incluímos outras para conectar o arquivo HTML com os arquivos style.css e script.js, que estão na mesma pasta que o index.HTML. Assim:

```
<link rel="stylesheet" href="style.css">

<script src="script.js" defer></script>
```

O arquivo CSS tem um seletor para `div` com as seguintes regras de estilização:

```
div {

  color: #C41818;

  font-size: 4rem;

}
```

Como temos o arquivo `script.js` vazio, é nele que aprenderemos a usar a propriedade `style`.

Usando o código do exemplo anterior e abrindo nosso projeto no navegador com a extensão Live Server, veremos o texto desse modo:



#### 6.1.1.2 Manipulando estilos

Para mudar a cor do texto com JavaScript, primeiro devemos ir no nosso arquivo script.js, **capturar o elemento div do DOM** e, por fim, salvá-lo em uma variável. Para fazer isso, usamos a seguinte linha de código:

```
let div = document.querySelector("div");
```

Após isso, usamos o **dot notation** (notação de ponto) para acessar a propriedade style que salvamos.

Por exemplo, para acessar a propriedade color e atribuir um novo valor a ela, usamos o operador de atribuição "=", seguido de uma string contendo o valor da propriedade. Veja o exemplo:

```
let div = document.querySelector("div");  
  
div.style.color = "blue"
```

A screenshot of a web browser displaying the text "Texto de exemplo" in a large, blue, serif font. The text is centered on the page.

Remover atributos.. atribuir a string vazia

```
Exercicio-mat-assincrono > 6.1.1-AddEstilos > js scripts > ...
1  let div = document.querySelector("div");
2
3  div.style.color = "blue";
4  div.style.color = ""
5
6
```

Document X

← → ↺ http://127.0.0.1:3000/Exercicio-mat-assincrono/6.1.1-AddEstilos/index.t

# Texto de exemplo

## 6.1.1.3 Sintaxe e convenções de escrita

Além dessas instruções, existe mais uma regra de sintaxe que deve ser respeitada quando usamos a propriedade style na linguagem JavaScript: a **convenção** de escrita [camelCase](#)



**IMPORTANTE:** As propriedades do objeto "Style" **são escritas com uma única palavra e redigidas da mesma forma que as escreveríamos em um arquivo CSS**, por exemplo: color, margin, display etc. Contudo, as propriedades, que têm nomes com mais de uma palavra, **seguem padrões de escrita diferentes**.

Na **linguagem CSS**, as escrevemos usando a padronização **snake-case**, já na linguagem **JavaScript**, usamos a **camelCase**. Dessa forma, as propriedades como background-color vira backgroundColor e text-decoration vira textDecoration.

Para manipular o tamanho da fonte do nosso texto, vamos acessar a propriedade fontSize e atribuir um novo valor. Isso é feito da seguinte forma:

```
div.style.fontSize = "16px"
```



Texto de exemplo

É importante lembrar: Que, nesse caso, a **convenção de escrita é uma regra de sintaxe**. Caso não a respeitemos, a linguagem JavaScript não interpretará o nosso código corretamente.

Leitura complementar [HORA DE CODAR. Como usar JavaScript para mudar propriedades CSS. \[S.d.\]](#).

[JAVASCRIPT TUTORIAL. JavaScript Style. \[S.d.\]](#).

Referência Bibliográfica

[W3 SCHOOLS. HTML DOM Element style](#)

#### 6.1.1.4 - Anotações Exercícios

1. Sintaxe correta pra acessar a cor de fundo de um elemento container?
  1. container.style.backgroundColor. Correto!Ao acessar as propriedades com nome de mais de uma palavras usamos a convenção de escrita camelCase.
2. Victor acessou um elemento do DOM e o salvou em uma variável chamada titulo. Como ele pode acessar a cor da fonte desse elemento com a linguagem JavaScript?
  1. titulo.style.color. Correto. Primeiro precisamos acessar a propriedade style do elemento, e depois alterar a propriedade pretendida.
3. Sintaxe correta pra definir margem 16px no formulário
  1. formulario.style.margin = "16px".Correto Atribuição =

#### 6.1.2 - Manipular classes com a propriedade .classList

A propriedade **style** é útil **quando queremos realizar estilizações menores**, ou seja, quando vamos **manipular uma propriedade CSS por vez**.

Porém, se quisermos **manipular várias propriedades de uma só vez**, é mais prático definir classes com várias regras de estilização e usar a linguagem JavaScript para manipular as classes que cada elemento inclui.

##### 6.1.2.1 - Setup de arquivos

Para demonstrar a propriedade classList, usaremos como exemplo um projeto simples com três arquivos:

`index.html`, `style.css` e `script.js`.

O arquivo `index.html` segue o padrão de estrutura base da HTML, com um único elemento dentro da tag `<body></body>`.

```
<div class="borda-azul">

    Manipulando listas de classes

</div>
```

Além disso, dentro da tag `<head></head>`, incluímos outras para conectar o arquivo HTML com os arquivos `style.css` e `script.js`, que estão na mesma pasta que o arquivo `index.html`.

```
<link rel="stylesheet" href="style.css">

<script src="script.js" defer></script>
```

Já o arquivo CSS tem um seletor de elemento, o `div`, e dois seletores de classe, `.borda-azul` e `.texto-novo`. Veremos as estilizações para cada um deles.

```
div {
    background-color: lightblue;
    width: 250px;
    height: 60px;
    display: flex;
    justify-content: center;
    align-items: center;
}

.borda-azul {
    border: solid 4px darkblue;
}

.texto-novo {
    color: #dc143c;
    font-weight: 800;
}
```

Como o arquivo `script.js` está vazio, é nele que aprenderemos a usar a propriedade `classList`.

Abrindo o nosso projeto no navegador com a extensão *Live Server*, vemos o **div** com a seguinte estilização já definida:

## Manipulando listas de classes

### 6.1.3 - Verificando se um elemento tem uma classe

Para manipular **listas de classes** com JavaScript, primeiro devemos ir no arquivo script.js, capturar o elemento div do DOM e, por fim, salvá-lo em uma variável.

Para fazer isso, usamos a seguinte linha de código:

```
let div = document.querySelector("div");
```

Após isso, use o **dot notation** para acessar a propriedade `classList` do elemento que acabamos de salvar. Essa propriedade é um **objeto** com métodos, que também podemos acessar usando o **dot notation**.

Agora, vamos usar o método `contains()`. Ele **verifica se um elemento possui ou não uma determinada classe e retorna um valor booleano**.

O método `contains()` **recebe** como argumento uma **string com o nome da classe**. Dessa forma, o retorno será `true` se o elemento conter essa classe e, caso não a possua, será `false`.

Nesse exemplo, passaremos o valor `borda-azul`, salvaremos o valor retornado pelo método em uma variável chamada `incluiClasse` e, por fim, usaremos a função `console.log()` para imprimir o resultado.

```
let div = document.querySelector("div")

let incluiClasse = div.classList.contains("borda-azul")

console.log(incluiClasse);
```

Ao salvar as mudanças e verificar o terminal no navegador, devemos ver o valor `true` impresso, pois o nosso elemento possui a **classe** `borda-azul`.

**VALE LEMBRAR:** A string passada nos métodos da `classList` **é composta pelos nomes das classes**, não pelos seletores CSS. Portanto, não é necessário colocar um ponto antes do nome da classe, como

fariamos com outros métodos, como `.querySelector()`.

#### 6.1.4 - Adicionando Classes

Para adicionar uma classe ao nosso elemento, vamos usar o método `add()`. Ele recebe como argumento uma string com o nome da classe que queremos adicionar.

Nesse exemplo, adicionaremos a classe `texto-novo`, que **irá alterar a cor e o peso da nossa fonte**.

```
div.classList.add("texto-novo");
```

**Manipulando listas de classes**

#### 6.1.5 - Removendo Classes

Para remover classes de um elemento, basta seguir os exemplos anteriores, mas aplicando o método `remove()`.

Dessa vez, removeremos a classe `borda-azul`, passando ela como argumento do método.

```
div.classList.add("texto-novo");  
  
div.classList.remove("borda-azul");
```

**Manipulando listas de classes**

#### 6.1.6 - Alterando classes

São comuns as situações onde queremos adicionar ou remover uma classe, mas **não temos certeza quais classes um elemento possui ou não**. Isso é devido à `classList` do elemento ter **sofrido muitas alterações** ou porque o **site permite o usuário interagir**, adicionando ou removendo classes de um determinado elemento.

Para resolver isso, poderíamos **criar uma função que recebe um elemento e o nome de uma classe como parâmetros**. Assim, o bloco de código verificaria se o elemento possui ou não a respectiva classe usando o método `contains()`. Além disso, usaria uma estrutura condicional para aplicar o método `remove()` caso o retorno seja `true`, ou `add()` se for `false`.

Contudo, a linguagem JavaScript nos oferece um método que realiza todo esse processo de forma automática, o `toggle()`.



Considere o método `toggle()` como um interruptor que nos permite alternar facilmente entre `add()` e `remove()`.

No momento, nosso elemento de exemplo perdeu a classe `borda-azul`. Então, se usarmos a função `toggle()` logo embaixo do bloco de código anterior, passando o nome da classe como variável, teremos o seguinte:

```
div.classList.add("texto-novo");  
  
div.classList.remove("borda-azul");  
  
div.classList.toggle("borda-azul");
```

Após salvar as mudanças e voltar ao navegador, podemos ver que a classe foi novamente removida. Isso ocorre porque estamos retirando a classe com o método `remove()`.

Além disso, a **primeira execução** do método `toggle()` verifica se o elemento possui a classe `borda-azul`. Após confirmar essa ausência, o método a **adiciona**. A segunda execução do método `toggle()` faz a mesma verificação, mas dessa vez o elemento possui a classe, então o método a **remove**.

#### Material Complementar

[SILVÉRIO, Henrique, Manipulando classes com classList API. Henrique Silvério, 23 mar. 2014](#)

#### Bibliografia

[JAVASCRIPT TUTORIAL. JavaScript classList. \[S.d.\]](#)

[W3 SCHOOLS. HTML DOM Element classList. \[S.d.\].](#)

### 6.1.7 - Anotações Exercícios

1. Gabriel usou uma estrutura condicional para verificar se o elemento `card` possui a classe `selecionada`. Como Gabriel pode fazer isso sem deixar o código mais complexo?
  1. `card.classList("selecionada" || "");` **Errado!** `classList` não é um método. Portanto, não podemos executá-lo como uma função.



2. `card.classList.toggle("selecionada")`. Correto. O método `toggle()` faz, automaticamente, a verificação de uma classe.
2. Gabriel é programador e deseja verificar se o elemento `inputEmail` possui a classe `input-invalido`. Qual é o comando que ele deve usar para fazer isso?
  1. `inputEmail.classList.contains("input-invalido")`. Correto. Método `.contains()`, da propriedade `classList` é uma função que recebe como argumento uma string com o nome da classe que se deseja verificar.
3. Agora, Gabriel deseja tirar a classe `visible` do elemento `sideNavBar`. Qual é o método que ele deve usar para isso?
  1. `sideNavBar.classList.remove("visible")`. Correto. o método `remove()` da propriedade `classList` nos permite eliminar uma classe de qualquer elemento.

## 6.2 - MÓDULO 02

### 6.2.1 - Função `.addEventListener()`

sabemos que, para fazer um site dinâmico com interação do usuário, devemos programar diversos eventos para serem acionados pelo próprio usuário, como clicar em um botão, apertar uma tecla, preencher um formulário etc. Para isso, usamos o método `addEventListener()`.

#### 6.2.1.1 - Setup de arquivos

Para demonstrar a função ou método `addEventListener`, usaremos como exemplo um projeto simples com dois arquivos: `index.html` e `script.js`. O arquivo `index.html` segue o padrão de estrutura de base HTML, ou seja, possui dois elementos dentro da tag `<body></body>`.

```
<h1>0</h1>

<button>+1</button>
```

incluir na `head` o `script.js`

#### 6.2.1.2 - Conceito de Eventos

Sempre que desejarmos implementar a **interatividade** nas nossas páginas, devemos nos fazer três perguntas: **"Quem?"**, **"O quê?"** e **"Quando?"**.

Ex: Sylvia é programadora de uma empresa de vendas e sua gestora solicitou a construção de um site para recolher informações dos clientes.

- **QUEM?**

- Se **refere aos elementos HTML que participam da interatividade**. Geralmente, precisamos de, no mínimo, **dois** elementos:
  - o que ativará a mudança
  - o que sofrerá as alterações.

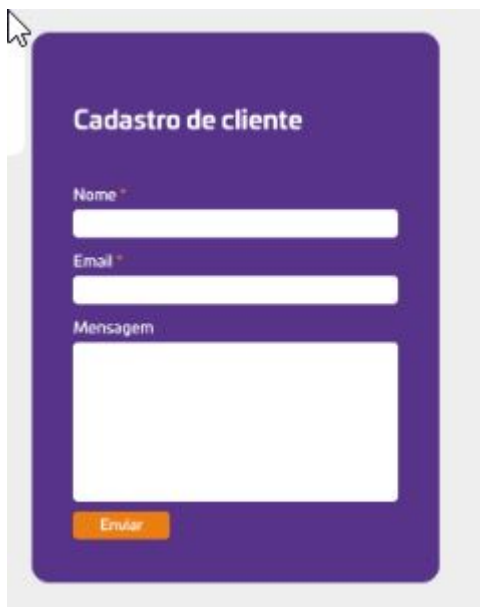
- Por exemplo, no caso de Sylvia, ela precisará **criar botões de Enviar** e campos no site para os usuários **inserir as informações**.

- **O QUE?**

- Se **refere ao comportamento da interatividade em si**. Nessa etapa, **usaremos a lógica de programação** para descrever aquilo que deve acontecer com os elementos definidos na primeira etapa.
  - No caso de Sylvia, ela precisará construir **códigos que verifiquem se algum dos campos disponibilizados, onde os usuários inserem seus dados, está vazio**. Caso sim, os botões de Enviar devem ser bloqueados. Mas, se estiverem preenchidos, os botões de Enviar devem ser liberados.

- **QUANDO?**

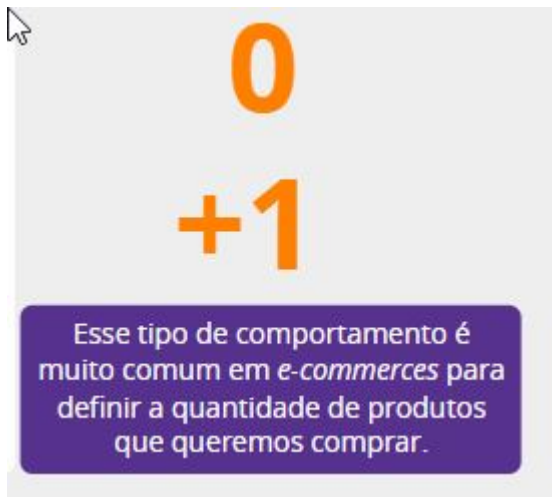
- Se **refere ao evento que acionará o comportamento** definido na etapa anterior.
  - Sylvia deverá **garantir que o envio do formulário**, ou o bloqueio dele, aconteça **assim que o usuário inserir seus dados corretamente**.



A mockup of a client registration form titled "Cadastro de cliente". The form has a purple header and a white body. It contains three input fields: "Nome \*" (Name), "Email \*" (Email), and "Mensagem" (Message). The "Nome" and "Email" fields are short text inputs, while the "Mensagem" field is a larger text area. At the bottom of the form is an orange button labeled "Enviar" (Send). A mouse cursor is visible at the top left of the form.

### 6.2.1.3 - Elementos

Faremos um site simples, com um número e um botão que incrementa esse valor em um cada vez que clicarmos no botão. Em outras palavras, o site começa exibindo o número zero, mas, ao clicar no botão, o valor do número será 1 e, ao clicar novamente, o valor será 2, pois sempre estará adicionando um.



Começaremos respondendo à pergunta “Quem?”. Para isso, iremos capturar os elementos do DOM que usaremos no nosso projeto.

Nesse caso, queremos capturar o elemento `h1`, que exibirá o número, e o elemento `button`, que ativará o comportamento.

```
let numero = document.querySelector("h1");  
  
let botao = document.querySelector("button");
```

Além desses dois elementos, declaramos uma **variável** contagem para salvar o valor que será incrementado. Isso é necessário **pois o número 0** exibido no nosso arquivo HTML **é uma string** com o caractere “0”, não um valor do tipo `number` ou `float`.

Existem outras formas de chegar no resultado desejado, mas, nesse exemplo, seguiremos a abordagem:

```
let numero = document.querySelector("h1");  
  
let botao = document.querySelector("button");  
  
let contagem = 0;
```

#### 6.2.1.4 - O Comportamento

Para definir o comportamento desejado, **encapsulamos nosso código em uma função** chamada `adicionarUm`. Começamos declarando ela com a estrutura padrão de uma função na linguagem JavaScript, ou seja, a função estará localizada embaixo das três variáveis criadas na etapa anterior.

```
let numero = document.querySelector("h1");
```

```
let botao = document.querySelector("button");

let contagem = 0;

function adicionarUm(){

}
```

Entre as **chaves**, descreveremos o **comportamento que queremos que o nosso site realize**. Nesse caso, a lógica será dividida em dois passos:

1. aumentar o valor da variável `contagem` em 1;
2. exibir o seu novo valor como o texto do elemento `numero`, usando a propriedade `innerText`.

```
let numero = document.querySelector("h1");

let botao = document.querySelector("button");

let contagem = 0;

function adicionarUm(){

    contagem = contagem + 1;

    numero.innerText = contagem;

}

adicionarUm(); // chamando a função e alterando a página
```

Agora, depois de salvar as mudanças e voltar ao navegador, devemos ver o número 1 no lugar do inicial 0, pois nossa função foi executada uma vez.

Após o teste, podemos apagar a execução manual da nossa função.

#### 6.2.1.5 - O Evento

Para que um usuário execute, precisamos usar o método, `addEventListener()`.

No nosso exemplo, o elemento responsável por disparar o evento é o botão +1, então, começaremos adicionando o método nele, logo embaixo da declaração da função `adicionarUm`.

Para fazer isso, basta escrever o nome do elemento, seguido de um ponto, o nome do método e um par de parênteses. Dessa forma:

```
botao.addEventListener();
```

O método `addEventListener()` é uma **função** que precisa de, **no mínimo, dois argumentos** para funcionar.

1. O primeiro é uma **string**, que define o tipo de evento que acionará a função.

1. Usaremos o **string click**

2. O segundo, é uma **função** com o comportamento que queremos executar.

No nosso exemplo, já declaramos nossa função na etapa anterior, portanto, basta escrever o seu nome sem os parênteses. Assim:

```
botao.addEventListener("click", adicionarUm);
```

Após salvar as mudanças e voltar no navegador, nosso contador deve estar funcionando da forma esperada. Ele precisa exibir um número inicial e aumentá-lo em 1 cada vez que clicamos no botão **+1**.

#### leitura Complementar

[MDN WEB DOCS. Introdução a eventos.](#)

#### Referência Bibliográfica

[LIANG, Joe. O método addEventListener\(\) - exemplo de código com listener de eventos em JavaScript. Free Code Camp, 19 out. 2022.](#)

[CFBCURSOS. Adicionando eventos com addEventListener em Javascript #P1 - Curso de Javascript Moderno - Aula 34](#)

### 6.2.1.6 - Anotações Exercícios

1. Qual é a forma correta de aplicar o método `addEventListener()` no elemento `btnSubmit`?

1. `btnSubmit.addEventListener("click", funcaoSubmeter)`.

1. Isso mesmo! A função `addEventListener` precisa receber, no mínimo, dois argumentos: uma string, que representa o tipo do evento, e uma função que define o comportamento a ser executado.

2. Para trabalhar com eventos, precisamos capturar, no mínimo, quantos elementos do DOM?

1. Dois elementos: o que ativa a mudança e o que sofre as alterações.

1. Isso mesmo! Podemos ter mais de um elemento que sofre alterações, mas precisamos tanto do elemento ativador quanto do que será alterado.

3. Qual a forma mais comum de organizar nosso código quando usamos `addEventListener`?

1. Declarando a função dentro dos parênteses do `addEventListener` como segundo parâmetro.

1. Muito bem! Isso é uma prática muito comum, especialmente usando arrow functions.

### 6.2.2 - Eventos do mouse

Já vimos como permitir que o usuário clique em um botão para modificar um elemento capturado do DOM. Relembrando que, para isso, usamos um dos eventos mais comuns no dia a dia do desenvolvedor: o **click**.

Contudo, existem outros **eventos do mouse** que podemos usar para adicionar interatividade aos nossos sites. Veremos alguns deles!

#### 6.2.2.1 - Setup de arquivos

Para demonstrar os diferentes tipos de eventos do mouse que podemos usar com a linguagem JavaScript, usaremos como exemplo um projeto simples com três arquivos: `index.html`, `style.css` e `script.js`.

O arquivo `index.html` segue o padrão de estrutura base da HTML, com os seguintes elementos dentro da tag `<body></body>`:

```
<button>Eventos de mouse</button>

<span>Passou o mouse por cima</span>

<section>

  Texto dinâmico

</section>
```

Além disso, dentro da tag `<head></head>`, incluímos outras tags para conectar o arquivo HTML com os arquivos `style.css` e `script.js`, que estão na mesma pasta que o arquivo `index.html`.

```
<link rel="stylesheet" href="style.css">

<script src="script.js" defer></script>
```

O arquivo `style.css` tem os seguintes blocos de declaração:

```
span {

  opacity: 0;

  transition: all 0.3s;

}

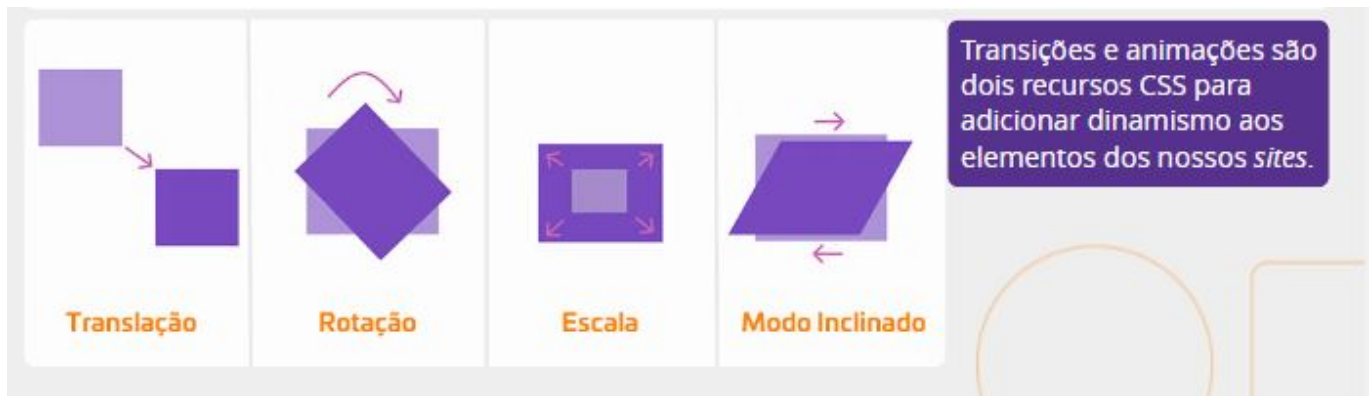
section {

  margin-top: 1rem;

}
```

Para o elemento `span`, usamos a propriedade `opacity` para definir a opacidade ou a transparência do elemento, passando valores entre 0 (completamente transparente) até 100 (completamente opaco).

Também usamos a propriedade `transition` para definir que todas as propriedades alteradas no elemento de valor `all` deverão ter uma transição de 0.3 segundos, com valor `0.3s`.



Finalmente, no arquivo `script.js`, usamos o método `querySelector()` para capturar três elementos do DOM e salvá-los em variáveis com os nomes das suas respectivas tags HTML. Para isso, as linhas de código devem ser:

```
let button = document.querySelector("button");

let span = document.querySelector("span");

let section = document.querySelector("section");
```

#### 6.2.2.2 - Evento `mouseover`

Nesse exemplo, mostraremos o uso dos quatro principais eventos do mouse, usando o *botão como elemento ativador* e os elementos *span* ou *section* como *elementos afetados*.

Primeiro, vamos trabalhar com o evento:

`mouseover`, que é **disparado sempre que o cursor do mouse passa por cima de um elemento**.

Agora, vamos criar a função `mostrarSpan`, que altera a propriedade `opacity` da propriedade `style` do elemento `span`.

```
function mostrarSpan(){

    span.style.opacity = "100"

}
```

Depois disso, logo embaixo da função criada, chamamos o elemento `button` e aplicamos o método `addEventListener()` nele, que **deve receber dois argumentos**. O primeiro será a *string* `mouseover` e o segundo, o nome da função `mostrarSpan`, sem os parênteses, pois estamos apenas passando-a como argumento. Dessa forma:

```
function mostrarSpan(){  
  
    span.style.opacity= "100"  
  
}  
  
button.addEventListener("mouseover", mostrarSpan);
```

Após salvar as mudanças e abrir o arquivo index.html no navegador, devemos ver apenas o botão e Texto dinâmico embaixo dele. Ao passar o mouse por cima do botão, o texto Passou o mouse por cima deve aparecer ao lado do botão.

Agora, o texto do nosso `span` aparece de forma dinâmica por cima do botão. Contudo, esse texto permanece visível mesmo se o cursor do mouse não estiver mais em cima do elemento.

Para corrigir isso, primeiro, devemos definir uma função para alterar a opacidade do `span` de volta para zero:

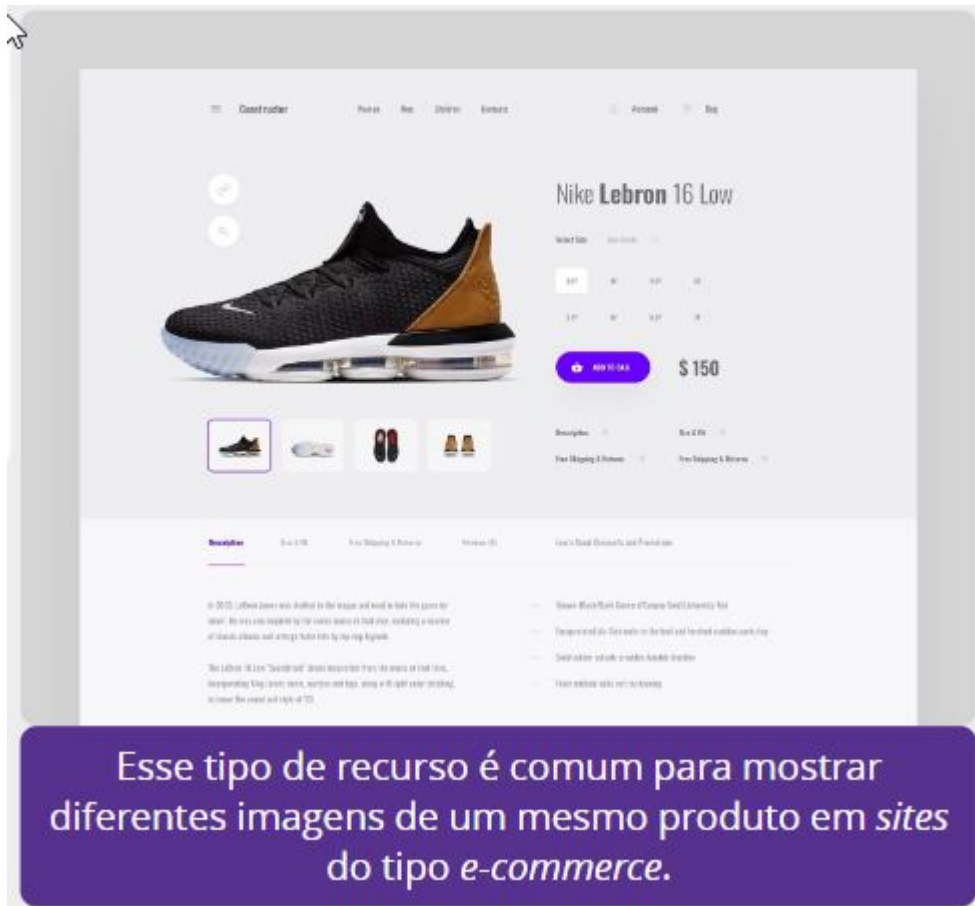
```
function ocultarSpan(){  
  
    span.style.opacity = "0"  
  
}
```

Logo embaixo da função, aplicamos o método `addEventListener` ao elemento `button`, passando a string `mouseout` como primeiro argumento e a função `ocultarSpan` como segundo argumento.

```
function ocultarSpan(){  
  
    span.style.opacity = "0"  
  
}  
  
button.addEventListener("mouseout", ocultarSpan);
```

Após salvar as mudanças e voltar ao navegador, devemos ver o texto Passou o mouse por cima aparecendo quando o cursor do mouse passa por cima do botão e sumindo quando o cursor sai de cima dele.





### 6.2.2.3 - Evento **click**

O evento **click** é a **junção de dois eventos do mouse**:

1. **mousedown**, que é disparado quando clicamos o botão esquerdo do mouse;
2. **mouseup**, que é disparado quando soltamos esse mesmo botão.

Assim, **click** é o *ato de apertar e soltar o botão esquerdo do mouse sobre algum elemento*. Para exemplificar, vamos começar criando uma função para alterar o conteúdo textual do nosso **section**.

```
function fazerUmClick(){
    section.innerText = "Fez um click simples!"
}
```

Agora, basta usar o método **addEventListener** no elemento **button**, passando a string **click** como primeiro argumento e a função **fazerUmClick** como segundo argumento. Assim:

```
function fazerUmClick(){
    section.innerText = "Fez um click simples!"
}
```

```
button.addEventListener("click", fazerUmClick);
```

#### 6.2.2.4 - Evento **dblclick**

O **duplo click**, ou **dblclick**, é o último evento que veremos. Ele não é muito comum, mas pode ser útil em determinadas situações. Seguindo o padrão dos outros três eventos, começamos declarando uma função para alterar o conteúdo textual do **section**.

```
function fazerDoisClicks(){  
    section.innerText = "Fez um duplo click!"  
}
```

Depois, aplicamos o método **addEventListener** ao elemento **button**, passando a string **dblclick** como *primeiro argumento* e a função **fazerDoisClicks** como *segundo argumento*.

```
function fazerDoisClicks(){  
    section.innerText = "Fez um duplo click!"  
}  
  
button.addEventListener("dblclick", fazerDoisClicks);
```

Existem outros eventos de click que podemos usar nos nossos sites. Podemos citar o **mouseout** e **mouseleave**.

#### Leitura Complementar

[ZEMEL, Tércio. Eventos JavaScript. DPW, 3 fev. 2014](#)

#### Bibliografia

[CFB CURSOS. JavaScript- 32- Eventos. \[S.d.\]](#)

[LUCAS NOVAES- PROGRAMAÇÃO. JAVASCRIPT DOS DESESPERADOS - Eventos de Mouse \(click, mouseout e mouseover](#)

#### 6.2.2.15 - Anotações Exercícios

1. quando o usuário passa o cursor do mouse por cima de um produto, surge uma caixa com o valor dele. Porém, após tirar o cursor, essas informações continuam visíveis. Qual evento precisa ser usado para corrigir isso?

1. `mouseout`.

1. Isso mesmo. Usamos o evento `mouseout` para definir comportamentos quando o cursor do mouse não está mais em cima de algum elemento.

2. Declarada a função `adicionarAoCarrinho()` em seu código e a adicionou como segundo argumento de uma `addEventListener`. O código ficou da seguinte forma:

`button.addEventListener("click", adicionarAoCarrinho());` Porém, o código deu erro após a execução.

1. R: Ao Passar a função `adicionarAoCarrinho` como argumento, não devemos incluir os parênteses após o nome dela.

1. Correto!. Se passarmos a função com os parênteses, **o navegador executará a função automaticamente ao invés de esperar ela ser chamada** com o evento.

3. Gabriela é programadora de uma empresa e deseja criar um site que, *quando o usuário passe o mouse por cima do botão Categorias, mostre, embaixo dele, quatro botões com links para as categorias disponíveis*. Para essa situação, qual é o evento do mouse mais apropriado?

1. R: `mouseover`.

1. Isso mesmo! O evento `mouseover` seria acionado quando o cursor do mouse passasse por cima de algum elemento.

## 6.3 - MÓDULO 03

### 6.3.1 - Capturar informações do evento

Usamos eventos apenas para acionar alguns comportamentos. Contudo, cada evento possui uma série de informações que podem ser necessárias para definir alguns comportamentos, como o tipo de evento (por exemplo, o `click`), a posição do mouse na tela quando o evento é realizado, sobre qual elemento foi feita a ação etc.



### 6.3.2 - Setup de arquivos

2 arquivos, `index.html`, que segue estrutura padrão e `script.js`. O `body` terá o seguinte conteúdo.



```
10 <body>
11 <h2 id="subtitulo">Você clicou no:
12   <span id="elemento-clicado"></span>
13 </h2>
14 <button id="botao-tempo">Tempo</button>
15 <p>Você clicou depois de
16   <span id="tempo"></span> milissegundos
17 </p>
18 </body>
```

Os elementos *span* são do tipo *inline*. Usaremos eles para inserir conteúdo textual em linguagem JavaScript dentro do elemento **h2**.

Definimos também o **id** de três elementos para facilitar a captura deles via DOM.

Para adiantar, no arquivo script.js, capturamos todos os elementos do DOM que usaremos, empregando o método `getElementById()`:

```
let elementoClicado = document.getElementById("elemento-clicado");

let botaoTempo = document.getElementById("botao-tempo");

let tempo = document.getElementById("tempo");
```

### 6.3.3 - Propriedade `timeStamp`

`timeStamp` **retorna um número** que representa o tempo transcorrido desde a renderização da página até o momento em que o evento aconteceu.

Essa informação pode ser útil para **provas on-line**, onde cada questão tem um **tempo máximo permitido** para escolher a resposta.

Para utilizá-la, começamos aplicando o método `addEventListener()` ao elemento `botaoTempo`, passando o evento do tipo `click` como primeiro argumento:

```
botaoTempo.addEventListener("click", );
```

Para o **segundo parâmetro**, *declaramos diretamente uma função* anônima dentro dos parênteses do `addEventListener()`. Lembrando que também é possível declarar essa função fora dos parênteses.

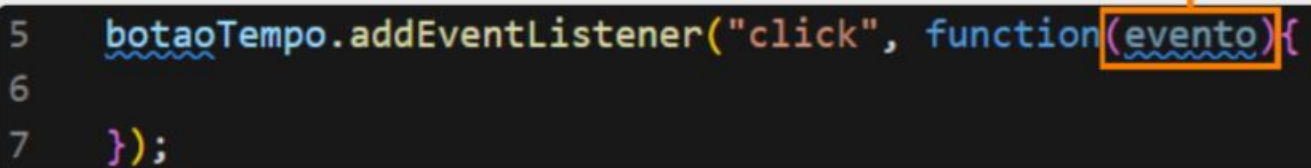
```
botaoTempo.addEventListener("click", function(){

});
```

1º parâmetro é a string com o tipo de evento que queremos **ESCUTAR** 2º parâmetro é a **função que define o que deve acontecer** quando o eventos for disparado.

```
botaoTempo.addEventListener("click", function(evento){  
  
});
```

Este parâmetro representa um **objeto com todas as propriedades do evento**. Por isso, é comum chamá-lo de **evento** ou de **e**.



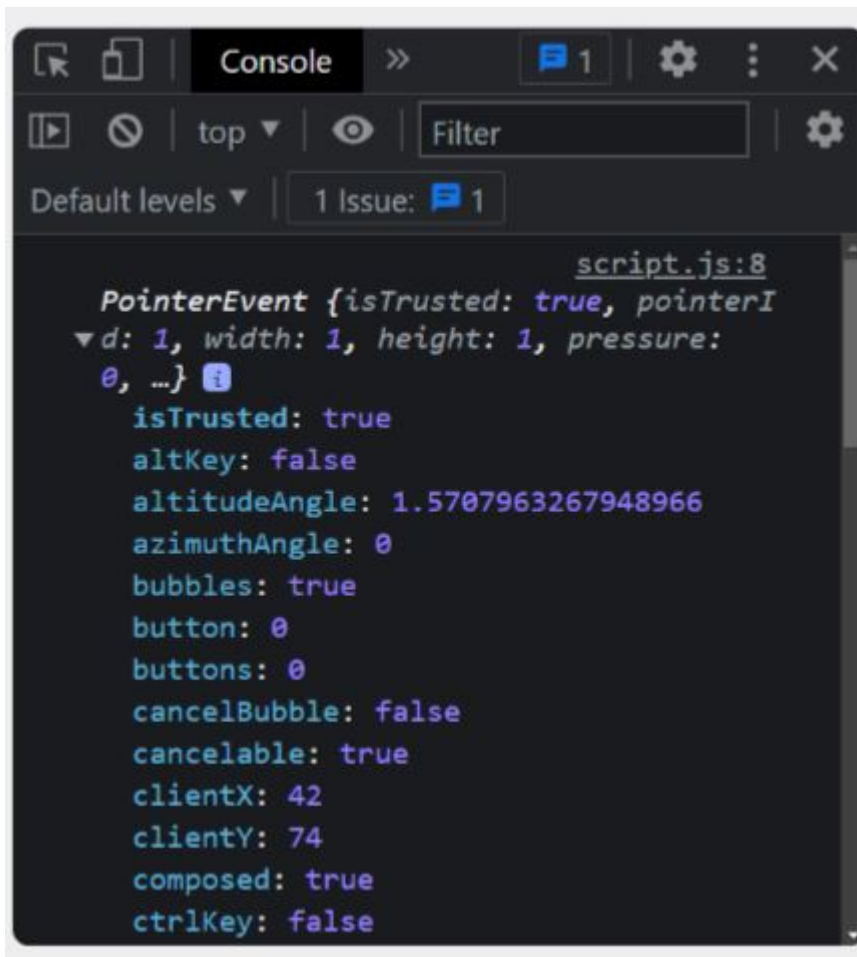
```
5 botaoTempo.addEventListener("click", function(evento){  
6  
7 });
```

No terminal, podemos imprimir todas as propriedades que o evento guarda. Para isso, basta fazer um **console.log( )** e passar o parâmetro **evento** como argumento:

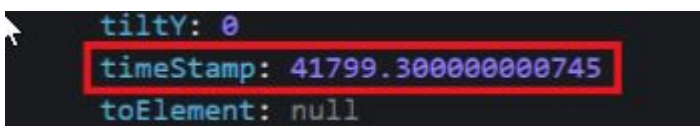
```
botaoTempo.addEventListener("click", function(evento){  
    console.log(evento);  
  
});
```

Depois de salvar as mudanças, abrir nosso projeto no navegador e clicar no botão **Tempo**, devemos ver um objeto chamado **PointerEvent** no terminal. Ele representa o evento em si.

Se clicarmos na seta à esquerda dele, veremos uma lista em ordem alfabética **com todas as propriedades que o evento guarda**. Observe a imagem.



A propriedade `timeStamp` está quase no fim da lista. Podemos vê-la retornando um número do tipo float, que representa a quantidade de **milissegundos** desde a renderização da página até o momento em que fizemos click no botão e disparamos o evento.



ATENÇÃO: O número retornado pode ser convertido em segundos. Para isso, divida ele por mil e arredonde o resultado

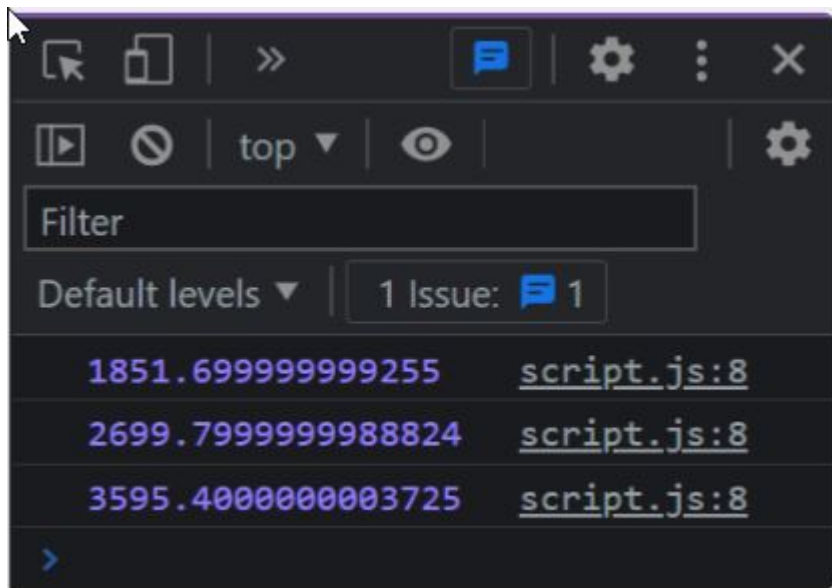
Voltando para o arquivo `script.js`, sabemos que o parâmetro `evento` **salva uma série de propriedades**.

Assim, para acessar a propriedade `timeStamp`, é preciso usar uma **dot notation**. Já para imprimir o valor da `timeStamp`, atualizamos o argumento que passamos no `console.log`:

```
botaoTempo.addEventListener("click", function(evento){
  console.log(evento.timeStamp);

});
```

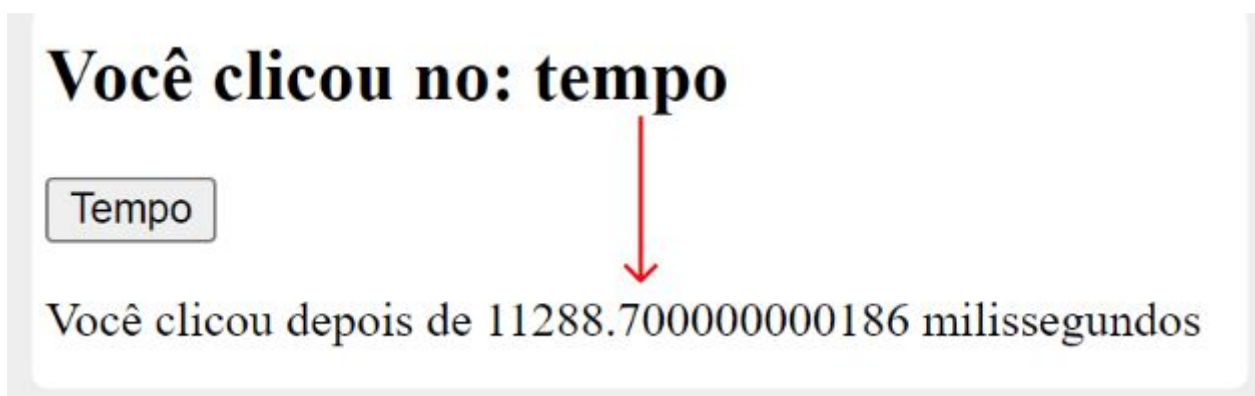
Após salvar as mudanças e voltar ao navegador, **cada vez que apertarmos o botão devemos ver um novo valor** em milissegundos no terminal:



Finalmente, precisamos **exibir esse valor no corpo da página**. Para isso, **atribuímos ele à propriedade innerText** do elemento **tempo**, que foi previamente capturado do DOM.

```
botaoTempo.addEventListener("click", function(evento){  
  console.log(evento.timeStamp);  
  tempo.innerText = evento.timeStamp;  
  
});
```

Depois de salvar as mudanças e voltar no navegador, cada vez que clicamos no botão devemos ver o valor em milissegundos sendo apresentado e atualizado no primeiro elemento `<p>`:

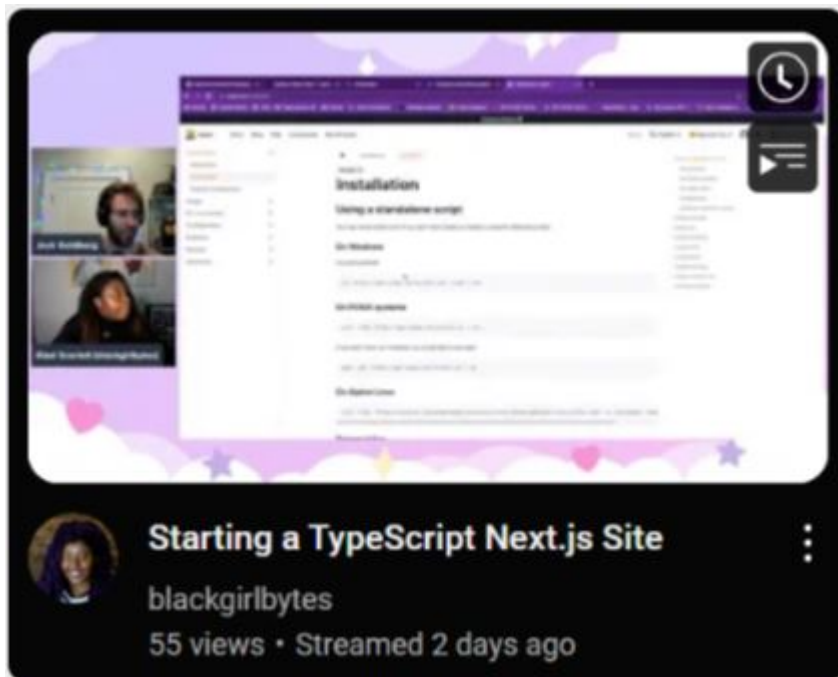


#### 6.3.4 - Propriedade `target`

A propriedade `target` nos permite *acessar informações e propriedades* do **elemento que ativou o evento** ou de qualquer um de seus *elementos filhos*, **que não necessariamente foram capturados do DOM**.



Para isso, é preciso usar um dos métodos estudados, o `getElementById` ou o `querySelector`, por exemplo. Uma vez que temos acesso ao elemento, podemos manipular ele à vontade.



Um exemplo do uso da propriedade `target` é quando passamos o mouse por cima de um elemento no YouTube e ele passa a mostrar uma prévia do vídeo.

Neste exemplo, usaremos *o próprio DOM como elemento pai*, mas poderíamos definir o mesmo comportamento, por exemplo, para uma lista ordenada de itens.

Começamos aplicando o método `addEventListener()` ao objeto `Document` e passando o evento do tipo `click` como primeiro argumento. Como segundo argumento, passamos uma *função anônima* com o argumento `e`, de *evento*.

```
document.addEventListener("click", function(e){  
  
});
```

Dentro da função, acessamos a propriedade `target` do *evento* (parâmetro chamado de `e`), e o imprimimos no terminal para entender o seu funcionamento:

```
document.addEventListener("click", function(e){  
  console.log(e.target)  
  
});
```

Após salvar as mudanças e voltar no navegador, podemos clicar nos elementos da página. Eles vão aparecendo no terminal conforme clicamos neles.



Você clicou no:

Tempo

Você clicou depois de 6402.300000000279 milissegundos

Em situações reais, no lugar do *Document* é recomendável usar um único elemento ou apenas um pai com os elementos filhos que queremos capturar.

OBSERVE: Como você pode ter observado, a propriedade `target` retorna o elemento que disparou o evento, no nosso caso, o `click`.

De volta ao VSCode, vamos querer mostrar na tela só o `id` daqueles elementos que foram clicados. Para isso, usamos a propriedade `id` da propriedade `target` (`e.target.id`) e atribuímos ele ao `innerText` do `elementoClicado`, O `span` que não tem texto, o qual foi previamente capturado no DOM:

```
document.addEventListener("click", function(e){  
  console.log(e.target)  
  elementoClicado.innerText = e.target.id  
})
```

Ao salvar as mudanças e voltar no navegador, deveríamos ver o `id` `subtitulo`, `botao-tempo` e `tempo` ao clicarmos nos seus **respectivos** elementos.

Você clicou no: tempo

Tempo

Você clicou depois de 11288.700000000186 milissegundos

CONCLUSÃO: Embora a propriedade `target` seja a **mais usada para acessar as informações do elemento que disparou o evento**, outras propriedades (como `timestamp`, `clientX`, `clientY`, `type`) podem ser usadas para trabalhar com informações do evento em si.

## LEITURA COMPLEMENTAR

JAVASCRIPT Básico: Eventos no JavaScript 14 (Parte 2). Mundo JS, 25 jun. 2019

## REFERÊNCIA BIBLIOGRÁFICA

ONYEJIAKU, Theodore Kelechukwu. What is event.target in JavaScript?. Educative

CFB Cursos. Entendo como usar o método target [event.target] em jQuery [jQuery] - Curso de jQuery - Aula 08. 24 fev. 2022

### 6.3.5 - Anotações Exercícios

1. Quando usamos o método `addEventListener()`, o que precisamos fazer na função passada como segundo parâmetro do `eventListener` para ter acesso às informações do evento?
  1. Adicionar um único parâmetro, geralmente, chamado de `event` ou apenas `e`.
    1. Correto. Muito bem! O parâmetro que passamos na nossa função representará o evento em si. Além disso, terá todas suas informações e propriedades.
2. Usando a propriedade `target`, como podemos acessar o id de um elemento que disparou um evento?
  1. `event.target.id`
    1. Correto. Isso mesmo! O atributo `id` e todos os outros do elemento (`class`, `value`, `name` etc.) são facilmente acessados com essa sintaxe.
3. Ao utilizar a propriedade `timestamp`, o que é retornado?
  1. Um número que representa o tempo transcorrido desde a página carregar até o evento ser disparado.
    1. Correto. Isso mesmo! O número representa a quantidade de milissegundos e, geralmente, vem com bastantes casas decimais.

## 6.4 - MÓDULO 04

### 6.4.1 - Eventos de teclado



Os eventos de teclado são muito populares na área de desenvolvimento de videogames.

## 6.4.2 - setup de arquivos

três arquivos:

- index.html;

```
<body>
  <h2>Tecla apertada:
    <span id="key-text"></span>
  </h2>
  <h2>Código da tecla:
    <span id="code-text"></span>
  </h2>
  <div id="quadrado"></div>

</body>
```

- style.css;

```
#quadrado {
  width: 24px;
  height: 24px;
  background-color: blue;
  position: relative;
}
```

No `id quadrado`, determinamos a altura, a largura e a cor de fundo do elemento para facilitar a visualização no navegador. Além disso, usamos a propriedade `position` com valor `relative` para

### mover o elemento.

- script.js

```
let keyText = document.querySelector("#key-text");
let codeText = document.querySelector("#code-text");
let quadrado = document.querySelector("#quadrado");
let distanciaDaEsquerda = 0
```

No arquivo `script.js`, iremos capturar todos os elementos do DOM que usaremos com o método `querySelector()`, que serve para retornar um valor. Além disso, vamos definir a variável `distanciaDaEsquerda`, a qual representará a distância entre o elemento `quadrado` e a borda esquerda do elemento pai.

Observe que temos três elementos principais: dois `h2` e uma `div`. Ambos elementos `h2` tem um próprio elemento `span` como filho.

As `spans` serão usadas para inserir texto com a linguagem JavaScript. Já a `div`, com `id="quadrado"`, será um elemento que moveremos usando as setas do teclado.

Esses `ids` e `classes` atribuídos servirão para referenciar os elementos nos arquivos `style.css` e `script.js`.



Conectar os arquivos

```
<link rel="stylesheet" href="style.css">

<script src="script.js" defer></script>
```

### 6.4.3 - Tipos de eventos de teclado

Quando trabalhamos com eventos de teclado, temos três tipos de evento que podemos usar como ativadores:

- **keypress;**
  - Tipo de evento que **só reconhece teclas de letras, números ou pontuação**, ou seja, **desconsidera** teclas como **Shift, Alt, setas** etc.
- **keydown;**
  - Tipo de evento disparado ao **apertar uma tecla**. Ele é *executado de forma repetida se a tecla permanece apertada*.
- **keyup;**
  - Tipo de evento disparado ao **soltar uma tecla**. Importante destacar que, *antes, ela precisa ser pressionada*.

Esses eventos são usados em diferentes situações. Eles nos **permitem ter mais controle sobre o tipo de interação** que queremos para o usuário.

Os eventos podem ser atribuídos a elementos do tipo **input**, se quisermos, por exemplo, ativá-los apenas quando o usuário inserir alguma informação em um local específico e travando se ele escrever em outro; Ou **diretamente na DOM**, se quisermos que o evento seja disparado em *qualquer parte da nossa página*.

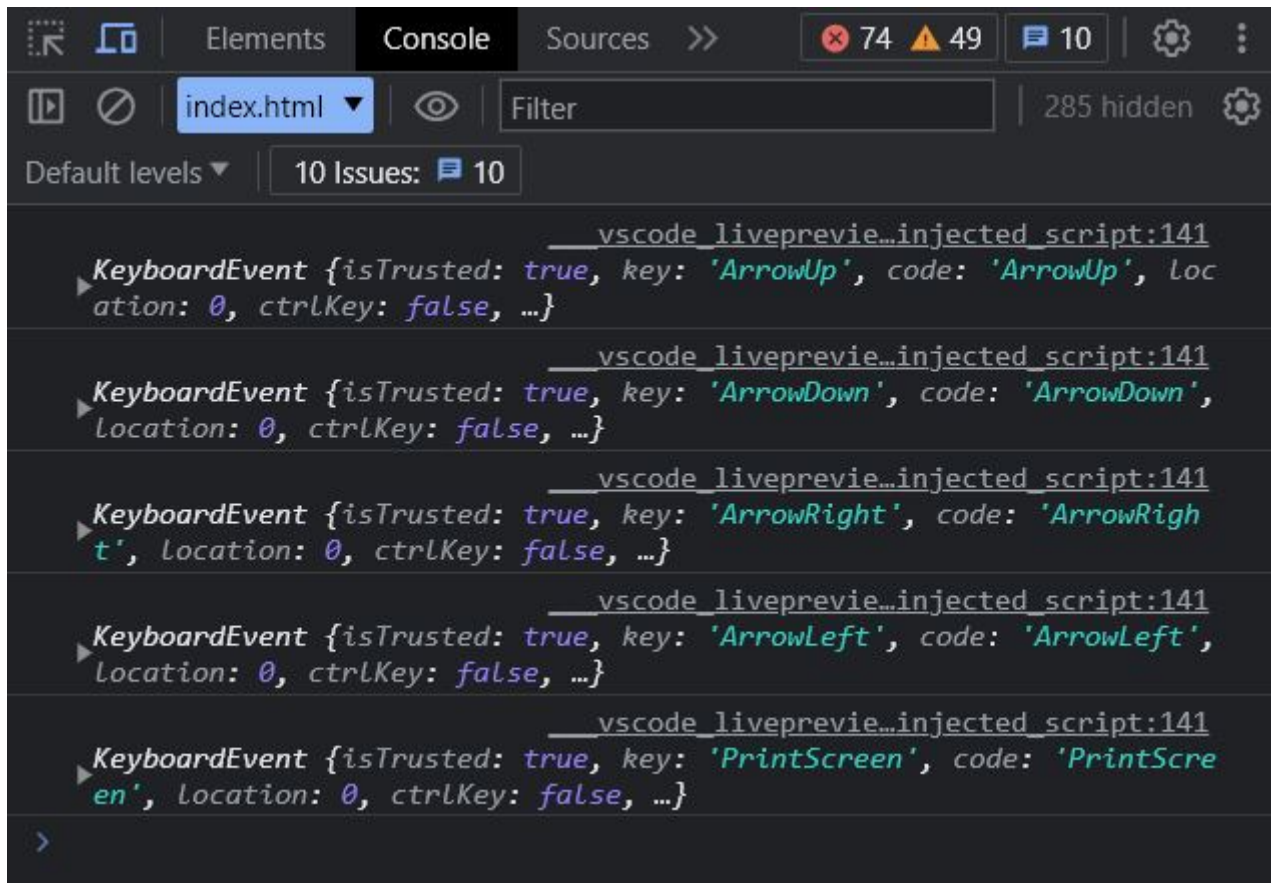
No nosso caso, o **evento de teclado** será aplicado diretamente no objeto **Document** para dispará-lo sem precisar selecionar nenhum elemento específico. Assim, o site reconhecerá o comando logo quando clicarmos em alguma tecla.

### 6.4.4 - Propriedades **key** e **code**

Primeiro, definimos um evento do tipo **keyup** e declaramos uma **arrow function** no **addEventListener**, que recebe o parâmetro **e** (evento). Dessa forma:

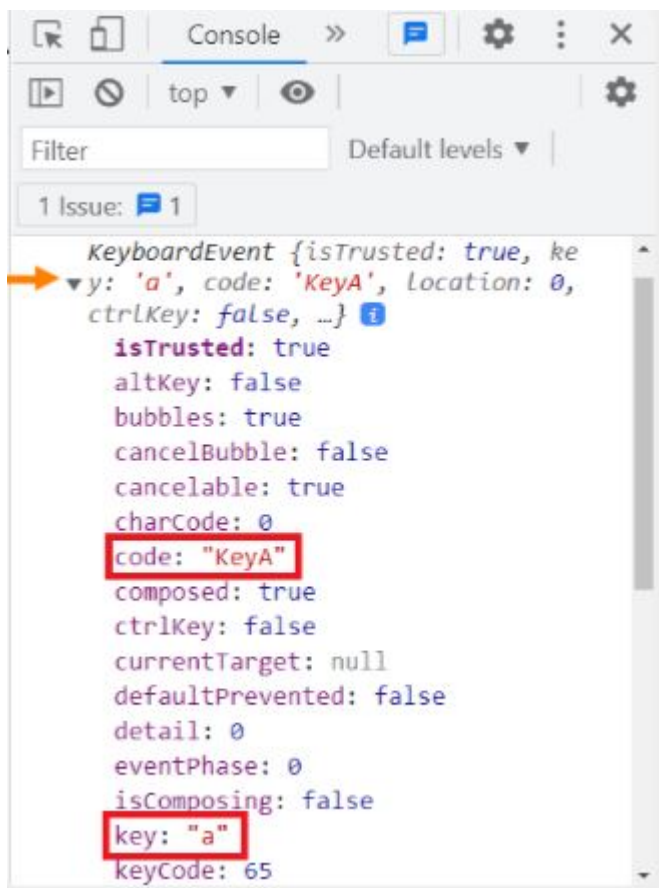
```
document.addEventListener("keyup", (e)=> {  
  
  }  
}
```

resultado no console:



O nome `KeyboardEvent` indica que o evento disparado é um evento de teclado.

Se clicarmos no triângulo à esquerda do `KeyboardEvent`, podemos ver todas as propriedades que os eventos de teclado possuem. As duas propriedades que mais nos interessam nesse momento são `code` e `key`.



Conforme o exemplo anterior, ao apertar a tecla **"A"** do teclado, serão retornadas duas linhas no terminal: a letra **"A"** minúscula e o código **KeyA**.



A **primeira linha** representa o *valor que essa tecla guarda*, neste caso, é **a**, e a **segunda linha** representa o **único código** que esse botão do teclado guarda.

ATENÇÃO: Podemos comparar **key** e **code** como variáveis, pois duas variáveis diferentes podem salvar o mesmo valor, mas **não podem ter o mesmo nome**.

Outra observação.

Propriedades key e code Para **teclas com valores únicos**, como as letras, **podemos usar qualquer uma das propriedades sem problemas**. Contudo, para **teclas que compartilham valores com outras**, é mais recomendável usar a propriedade **code**.

### 6.4.5 - Exibindo os valores das Propriedades

Por exemplo, se quisermos exibir os valores das propriedades **key** e **code** na nossa página, podemos chamar os elementos **keyText** e **codeText**, declarados no arquivo **script.js**, e atribuir a propriedade **innerText** deles aos valores das propriedades **key** e **code** respectivamente. Observe:

```
document.addEventListener("keyup",
(e)=> {
  keyText.innerText = e.key;
  codeText.innerText = e.code;
}))
```





Após salvar as mudanças e voltar ao navegador, os valores de ambas propriedades devem ser exibidos na página sempre que apertarmos qualquer tecla.

**Tecla apertada: a**

**Código da tecla: KeyA**

#### 6.4.6 - Deslocar um elemento

Para mover o elemento **div** com a classe **quadrado** no navegador, começamos adicionando outro **eventListener** ao objeto **Document**, dessa vez usando o tipo de evento **keydown**

```
document.addEventListener("keydown", (e) => {  
  
  })
```

Usaremos o evento **keydown** para poder *continuar deslocando o elemento ao manter apertada uma tecla*.

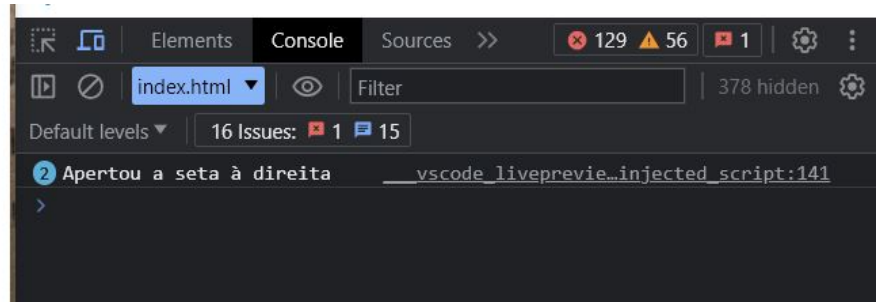
Queremos que nosso quadrado se desloque para a direita ao apertar a **tecla da seta à direita**, ou **arrow right**. Sabendo que o código desta tecla é **ArrowRight**, podemos usar uma estrutura condicional na nossa



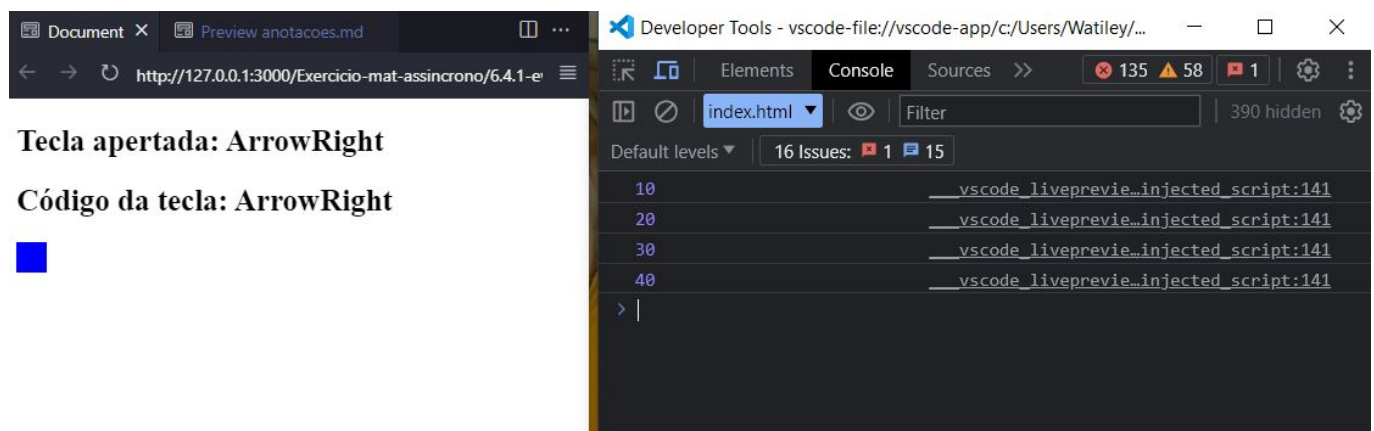
`arrow function` para verificar o valor do `code` do `evento`. Se o valor for o mesmo que `ArrowRight`, podemos imprimir uma mensagem com `console.log()`.

Tecla apertada: `ArrowRight`

Código da tecla: `ArrowRight`



Nesse momento, o elemento quadrado está a **zero pixels** de distância *da borda esquerda do seu elemento pai* dele, o `body`. Para deslocá-lo dez pixels à direita, no bloco de código da nossa estrutura condicional, **adicionamos dez** ao valor atual da variável `distanciaDaEsquerda` e imprimimos o seu o valor no terminal.



Agora, usaremos esse valor para deslocar o elemento `quadrado`. Para isso, precisamos acessar a propriedade `style` da variável `quadrado` e a propriedade `left` para definir a distância entre o elemento e a borda esquerda do seu elemento pai.

## Deslocar um elemento

Esta propriedade recebe uma *string* com o valor em *pixels*, *rem*, porcentagem ou qualquer unidade de medida CSS.

```
1 document.addEventListener("keydown", (e) => {  
2   if(e.code == "ArrowRight"){  
3     distanciaDaEsquerda = distanciaDaEsquerda + 10  
4     console.log(distanciaDaEsquerda);  
5  
6     quadrado.style.left = distanciaDaEsquerda + "px"  
7   }  
8 })
```

É necessário concatenar uma *string* usando o valor da variável *distanciaDaEsquerda* e a *string* *px* para a unidade de medida *pixels*.

The screenshot shows a web browser window on the left and a VS Code editor on the right. The browser window displays the text "Tecla apertada: ArrowRight" and "Código da tecla: ArrowRight" above a small blue square. The VS Code editor shows the source code of the page, which is a series of 16 lines of JavaScript code, all of which are identical: `__vscode_livepreview_injected_script:141`. The browser's address bar shows the URL `http://127.0.0.1:3000/Exercicio-mat-assincrono/6.4.1-e`. The VS Code editor's status bar shows "Port: 3000" and "Go Live".

Embora pouco usados no desenvolvimento web, os **eventos de teclado** podem destacar os nossos sites, oferecendo a possibilidade do usuário **interagir com a página** usando o teclado.

**Fechar um pop-up com a tecla Escape** ou **mostrar subseções da barra de navegação** apertando teclas numéricas são exemplos de interações práticas e simples de implementar nos sites.

LEITURA COMPLEMENTAR [LUIS TAVARES. JavaScript - Aula 16 - Eventos do teclado. 2 set. 2020.](#)

REFERÊNCIA BIBLIOGRÁFICA [KARIUKI, Benso. Introduction to Keyboard Events in JavaScript. Section,19 mar. 2021](#)

DEVMEDIA. Como usar a propriedade Position - CSS. [S.d.].

### 6.4.7 - Anotações Exercícios

1. Maria está desenvolvendo um jogo on-line e gostaria que o elemento carro se deslocasse pela tela quando o usuário pressionar as teclas de setas do teclado. Qual é o tipo de evento de teclado que Maria deve usar?
  1. keydown
    1. correta! Isso mesmo! O evento do tipo **keydown** é executado de forma contínua se o usuário manter apertada a tecla do evento.
2. Podemos usar os eventos de teclado em conjunto com outros conceitos de programação?
  1. Sim! Podemos usar os eventos de teclado com estruturas condicionais, loops, operadores lógicos etc.
    1. correta! Muito bem! Podemos usar os eventos de teclado em conjunto com qualquer outro conceito de programação.
3. Imagine que seu site tem uma seção com uma calculadora on-line, mas você gostaria que os usuários pudessem usá-la através das teclas numéricas do teclado NumPad. Qual propriedade é a mais indicada para isso?
  1. event.code
    1. correta! Isso mesmo! Ao usar a propriedade code, conseguimos diferenciar teclas que guardam valores iguais.

## 6.5 - MÓDULO 05

### 6.5.1 - Funções preventDefault() e alert()

Java oferece uma série de funções/métodos nativas que agilizam o trabalho.

### 6.5.2 - Setup de arquivos

Usaremos dois arquivos:

- index.html
  - estrutura base !

```
<head>
  <script src="script.js" defer></script>
</head>
<body>
  <h2>Visitar o site da

  <a id="link-proz" href="https://prozeducacao.com.br/"
target="_blank">Proz</a>

  </h2>
  <form action="">

    <input type="text" name="input">
```

```

    <button type="submit">Enviar</button>
  </form>

</body>

```

O atributo *target*, com o valor **\_blank**, serve para abrir um *link* em uma nova aba.

```

10 <body>
11 <h2>Visitar o site da
12 <a id="link-proz" href="https://prozeducacao.com.br/"
   target="_blank">Proz</a>
13 </h2>
14 <form action="">
15 <input type="text" name="input">
16 <button type="submit">Enviar</button>
17 </form>
18 </body>

```

O atributo *name* garante que a informação do *input* será enviada.

Os botões do tipo *submit* servem para enviar as informações preenchidas nos formulários.

- script.js

```

//CAPTURAR
let linkProz = document.getElementById("link-proz")
let btnSubmit = document.querySelector("button[type=submit]")

```

### 6.5.3 - Função `preventDefault()`

Pode ser traduzida como "impedir o padrão". Ela **serve para desativar o comportamento padrão de algum elemento HTML**. abrir index.html pelo LiveServer.

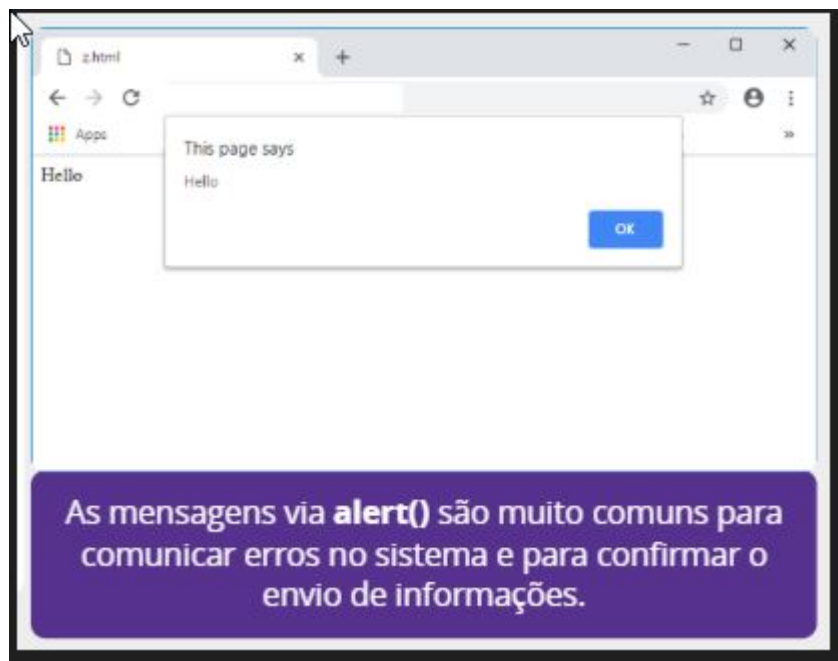
ELEMENTO	COMPORTAMENTO PADRÃO
<code>a</code>	é navegar para uma determinada URL, no nosso site ou externa. No nosso exemplo, ao clicarmos na palavra Proz, seremos direcionados ao site da Proz
botão tipo submit	é enviar as informações do nosso formulário

**Para impedir o comportamento padrão** do link, adicionamos um `eventListener` ao elemento `linkProz`. Como o evento que dispara o comportamento padrão do link é o `click`, o adicionamos como primeiro argumento e uma função anônima como segundo, a qual tem um argumento evento:

```
linkProz.addEventListener("click", (e)=> {  
  
  })  
  
btnSubmit.addEventListener("click", (e)=> {  
  
  e.preventDefault()  
  
  })
```

#### 6.5.4 - Função `alert()`

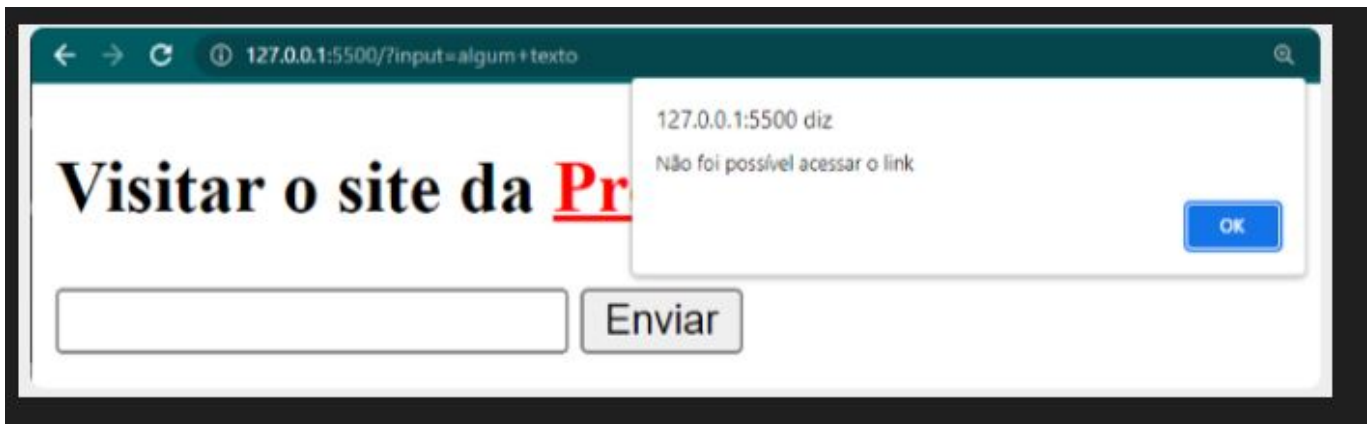
A função `preventDefault()` nos ajuda a controlar quais interações queremos fornecer ao nosso usuário e quando queremos liberá-las. Contudo, se não retornarmos um *feedback*, o site pode parecer que não está funcionando. Uma forma simples e eficiente de dar esses *feedbacks* é com a função nativa `alert()`.



No nosso exemplo, para adicionar uma mensagem de *feedback* ao clicar no elemento `linkProz`, chamamos a função `alert()` embaixo da `preventDefault()`. Depois, passamos uma *string* que tenha a mensagem que queremos exibir como *argumento*. Observe:

```
linkProz.addEventListener("click", (e)=> {  
  
  e.preventDefault();  
  
  alert("Não foi possível acessar o link");  
  
  })  
  
btnSubmit.addEventListener("click", (e)=>{  
  e.preventDefault()
```

```
    alert("Não foi possível acessar o link");  
  
  })
```



**Interromper** o comportamento padrão de um elemento e exibir mensagens de feedback pelo navegador são dois recursos que nos ajudarão a implementar uma camada de segurança nos nossos formulários e evitar problemas.

EVITAR COMPORTAMENTO PADRÃO E MOSTRAR UMA MENSAGEM, PARA QUANDO HOUVER ERRO NO CAMPO E EVITAR O ENVIO (COMPORTAMENTO PADRÃO), POR EXEMPLO UM LINK SEM OS PONTOS, OU EMAIL SEM ARROBA, OU LETRAS SENDO ENSERIDAS EM CAMPOS DE TEXTO.

Leitura Complementar

[RICARDO. Alert em JavaScript. Devmedia, 2016.](#)

Referência bibliográfica

[Event.preventDefault\(\). MDN Web Docs, 19 nov. 2022](#)

[NOLETO, Cairo. Javascript alert, confirm e prompt: caixas de diálogo Popup!. Be Trybe, 25 fev. 2022.](#)

## 6.5.5 - Anotações Exercícios

1. Sérgio tem um link em sua página com o texto "Inglês". Ele gostaria que, ao clicar nele, o texto de boas-vindas mudasse do português para o inglês. Porém, cada vez que ele clica no link, a página só recarrega. Como podemos ajustar isso?
  1. Usando o método `preventDefault()` para interromper o comportamento padrão do link.
    1. Resposta correta! Isso mesmo! O método `preventDefault()` interrompe o comportamento padrão de qualquer elemento HTML.
  2. Que tipo de argumento recebe a função `alert()`?
    1. Uma string com a mensagem que queremos exibir.
      1. Resposta correta! Muito bem! A string que passarmos como argumento será exibida como mensagem no pop-up da `alert()`
  3. Sérgio chamou a função `preventDefault()` dentro da função anônima `"( ) => { preventDefault( ) }"`. A `preventDefault()` está em um `eventListener`, mas não está funcionando. Por quê?

1. Porque é necessário passar um argumento que representa o evento e chamar um método a partir de um argumento. Por exemplo, o `event.preventDefault()`.

1. Resposta correta! Isso mesmo! A função `preventDefault()` **sempre está atrelada a um evento**. Portanto, devemos definir o evento como parâmetro da função.

### 6.5.6 - Eventos de formulários

*Eventos de mouse e de teclado* são úteis para adicionar diversas interatividades aos nossos sites. Porém, eles **possuem limitações no trabalho com formulários**.

JavaScript tem um grupo de eventos específicos para resolver isso:

EVENTO	DESCRIÇÃO
focus	O elemento é <b>focado</b> pelo usuário.
blur	O elemento <b>perde o foco</b> do usuário.
change	O elemento teve seu <b>valor alterado depois de perder o foco</b> .

#### 6.5.6.1 - Setup de arquivos

- index.html
  - estrutura padrão html !
  - adicionar no `head`

```
<script src="script.js" defer></script>
```

- composição do `body`

```
<form action="">
<label for="email">Email:</label>
<input type="email" id="email" name="email">
<br>
<label for="idade">Idade:</label>
<input type="number" id="idade" name="idade" value="35">
<br>
<button type="submit">Enviar</button>
</form>
```



### Setup de arquivos

```
9 <body>
10   <form action="">
11     <label for="email">Email:</label>
12     <input type="email" id="email" name="email">
13     <br>
14     <label for="idade">Idade:</label>
15     <input type="number" id="idade" name="idade" value="35">
16     <br>
17     <button type="submit">Enviar</button>
18   </form>
19 </body>
```

Neste caso, estamos usando a tag **br** para evitar criar mais um arquivo. Porém, para isso, é recomendável definir todas as estilizações com CSS.

O *input* de **idade** usa o atributo **value** para ser renderizado no navegador com um valor padrão de 35.

- script.js

```
//CAPTURAR ELEMENTOS DO DOM
let inputEmail = document.getElementById("email");

let inputIdade = document.getElementById("idade");

let formulario = document.querySelector("form");
```

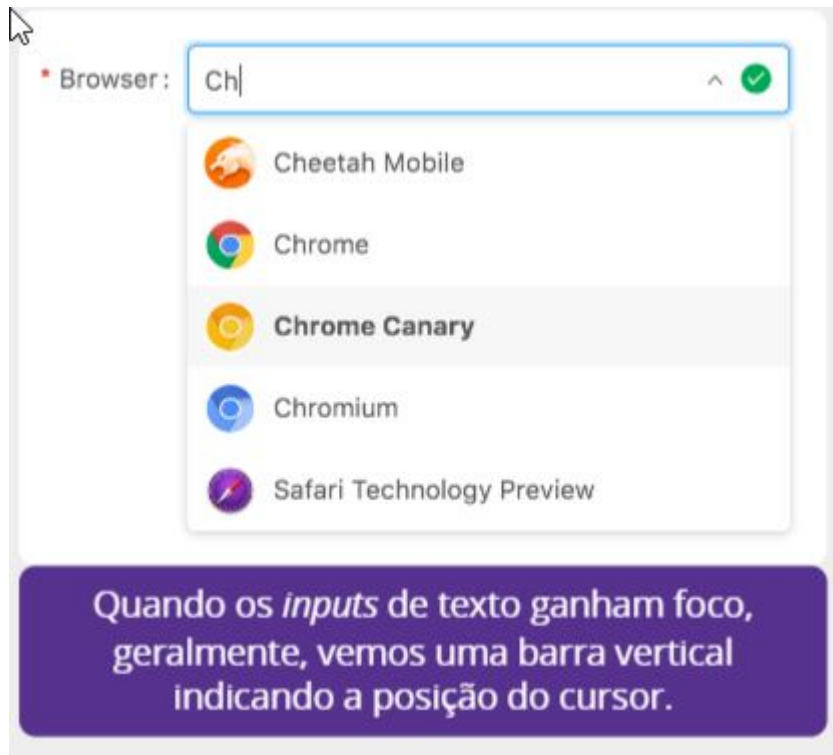
#### 6.5.6.2 - Evento **focus**

Para entender melhor como funciona o evento focus, vamos usar o aplicativo WhatsApp como exemplo.

Quando você quer mandar uma mensagem para alguém, primeiro você clica na caixa de texto vazia para aparecer a barra do teclado. Nesse momento, dizemos que o input ganhou foco.

Depois, você digita o texto e pressiona o botão de enviar. Quando o celular não mostra mais o teclado, ele indica que você não consegue mais inserir dados na caixa de texto, ou seja, o input de texto perdeu o foco.





Quando os inputs de texto ganham foco, geralmente, vemos uma barra vertical indicando a posição do cursor

Para demonstrar o uso do input, precisamos abrir o arquivo script.js e adicionar um eventListener ao elemento inputEmail.

O **primeiro argumento** será a string `focus` e o **segundo**, uma `arrow function`:

```
inputEmail.addEventListener("focus", ()=> {  
  });
```

Agora, vamos mudar a cor de fundo do input quando ele ganhar foco.

Podemos fazer isso chamando o próprio elemento `inputEmail` dentro do bloco do código da `arrow function`.

Outra opção é usar um argumento evento e acessar sua propriedade `target`. Nesse último caso, chamaremos o próprio elemento:

```
inputEmail.addEventListener("focus", ()=> {  
  inputEmail.style.backgroundColor = "lightgreen"  
});
```

O **focus** é comumente usado para exibir algum tipo de texto que ajude o usuário a preencher o campo. Por exemplo, a mensagem "Não use caracteres especiais nem espaços".



Formulario de login:

Email:

Idade:

#### 6.5.6.3 - Evento **blur**

Para definir o que deve acontecer com um elemento após perder o foco, usamos o evento **blur**.

Assim, chamamos o elemento **inputEmail** novamente e adicionamos mais um **addEventListener**.

Porém, desta vez, passaremos a string **blur** como primeiro argumento e uma **arrow function** como segundo. Dessa forma:

```
inputEmail.addEventListener("blur", (e)=> {  
    e.target.style.backgroundColor = ""  
});
```

Atribui uma string vazia como valor da propriedade **backgroundColor**, que **removerá** a estilização aplicada na etapa anterior.

voltar no navegador, devemos ver nosso input de e-mail ganhando e perdendo a estilização cada vez que clicamos dentro e fora dele, independentemente do valor que ele possui.

#### 6.5.6.4 - Evento **change**

O evento **change** é disparado **quando um elemento sofre uma alteração no seu valor**.

Contudo, isso **pode variar** de acordo com o *tipo de input* e da forma como alteramos os seus dados.

Para demonstrar isso, chamamos o elemento **inputIdade**, aplicamos o método **addEventListener**, passamos como primeiro argumento a string **change** e, como segundo, uma **arrow function**. Observe: Para visualizar o momento em que o evento **change** é disparado, executamos a função **alert()** com alguma mensagem dentro do bloco de código da **arrow function**.

```
inputIdade.addEventListener("change", ()=> {  
    alert("Certeza que quer alterar seus dados?")  
});
```

Atenção! Entretanto, se você digitar o valor da idade diretamente no **input**, verá que o evento só é disparado **após** o **input perder o foco**. Esse é um comportamento que devemos levar em consideração quando usamos o evento **change** com **inputs de texto**.

#### 6.5.6.5 - Evento **submit**

É sempre IMPORTANTE mostrar algum feedback ao clicar no botão de enviar. Seja para avisar que não foi possível efetivar o envio por falta de informações obrigatórias ou para confirmar que as informações mandadas estão corretas.

Para fazer isso, chamamos o elemento **formulário**, adicionamos nele o método **addEventListener**, passamos a string do evento **submit** como primeiro parâmetro e, como segundo, uma **arrow function**.

Depois disso, basta executar um **alert** dentro do bloco de código da **arrow function** com uma mensagem **confirmando o envio dos dados**.

```
formulario.addEventListener("submit", ()=> {  
    alert("Dados enviados com sucesso!")  
});
```



Conclusão Nesses exemplos, usamos os eventos de formulário apenas para aplicar algumas estilizações e mandar mensagens de alerta. Porém, existe uma infinidade de ações que podemos aplicar em conjunto com eles.

Alguns exemplos:

- Limitar um número mínimo ou máximo de caracteres,

- verificar se um e-mail possui ou não um domínio específico,
- destacar inputs com valores inválidos e impedir o envio de um formulário,
- caso algum campo obrigatório esteja vazio,

São apenas algumas das validações que podemos aplicar usando esses eventos.

#### Leitura Complementar

RICARDO. [Trabalhando com eventos em JavaScript](#). Dev Media, 2013.

GALLO, Vanessa. [Eventos com JavaScript](#). Computer Science Master, 6 mar. 2022

#### Referência bibliográfica

GoPHP. [34 - Form Events | Events in JavaScript | JavaScript Tutorial for Beginners](#). 11 jun. 2020.

### 6.5.6.6 - Anotações Exercícios

1. Estamos desenvolvendo um site para pessoas com deficiência visual e gostaríamos que, ao clicar em um **input**, o site reproduzisse um áudio explicando qual tipo de informação deve ser inserida. Qual é o evento que devemos usar?
  1. **focus**.
    1. Resposta correta! Isso mesmo! O evento focus será disparado assim que o usuário acessar o input e reproduzir o áudio correspondente.
2. Você capturou o elemento **button** de um formulário, usou o método **addEventListener**, passou **submit** como primeiro argumento e uma **função anônima** como segundo e, por fim, usou a função **alert( )**. Porém, o seu formulário não está mostrando a mensagem de **alert** ao enviar os dados do formulário. Por que isso está acontecendo?
  1. O evento do tipo submit deve ser atribuído ao formulário, não ao botão.
    1. Resposta correta! Muito bem! Mesmo o botão sendo do tipo submit, o evento precisa estar atrelado ao formulário, não ao botão.
3. Angélica estava escrevendo as validações de um **input** obrigatório de e-mail. Para isso, ela usou um evento e conseguiu disparar uma mensagem de erro caso o **input** *perdesse o foco e o texto não tivesse um @*. Porém, se clicar no **input**, **não digitar e, depois, clicar fora do input**, ele **não mostra ao usuário a mensagem** avisando que o **campo é obrigatório**. Qual evento Angélica usou e qual ela deveria ter usado?
  1. Ela usou change, mas deveria ter usado focus.
    1. Resposta incorreta! Na verdade, se o evento focus for usado, a mensagem será mostrada cada vez que o usuário clicar no input. Angélica quer que a mensagem seja escrita quando o input perder o foco e estiver vazia.
  2. Ela usou o change, mas deveria ter usado o blur.
    1. Resposta correta! Muito bem! O evento change funciona apenas quando o usuário muda o valor do input. Já o blur será chamado cada vez em que o input perder o foco, independentemente de se o valor dele mudou ou não.

## 6.6 - MÓDULO 06

### 6.6.1 - Validação de formulários

[Link arquivo Formulários](#)

## 6.6.2 - Revisão do módulo - JS II - Parte01

- PROPRIEDADE `style`

1. Essa propriedade é usada para **manipular as propriedades de estilização** dos elementos capturados do DOM:

```
let titulo = document.querySelector("h1");
titulo.style

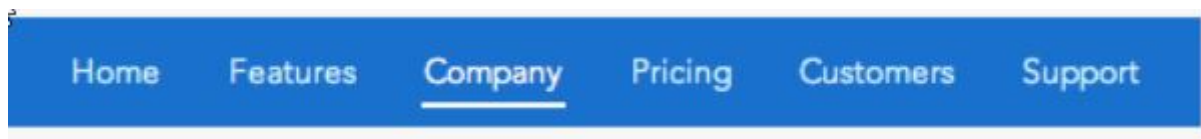
//Após acessar a propriedade style, podemos acessar qualquer propriedade CSS
e atribuir novos valores como strings a ela.

titulo.style.color = "#D4210D"

//Propriedades com mais de uma palavra no nome devem ser escritas usando o
padrão de escrita camelCase.

titulo.style.fontSize = "2.4rem"
```

A propriedade `style` serve para **adicionar estilizações simples** de forma dinâmica, como sublinhar o item selecionado em uma barra de navegação.



- PROPRIEDADE `classList`

Essa propriedade é usada para **manipular a lista de classes atribuídas a um elemento capturado** do DOM.

A propriedade `classList` possui uma série de **métodos** que nos permitem **consultar e manipular a lista de classes do elemento**. Esses métodos são executados escrevendo um par de parênteses logo após o nome do método e passando uma string como argumento.

```
let section = document.querySelector("section");
section.classList.
```

1. Método `contains()` Recebe uma string com o nome de uma determinada classe, verifica se o elemento possui ou não essa classe e retorna um valor booleano.

```
sections.classList.contains("container-produtos")
```

## 2. Método `add()`

Com o método `add()`, adicionamos uma classe ao elemento selecionado

```
sections.classList.add("container-produtos")
```

## 3. Método `remove()`

Usamos o método `remove()` para remover uma classe da lista de classes do elemento.

```
sections.classList.remove("container-produtos")
```

## 4. Método `toggle()`

O método `toggle()` funciona como um interruptor, ou seja, se o elemento possui a classe passada como argumento, o método a remove. Caso ele não a possua, o `toggle()` adiciona.

```
sections.classList.toggle("visible")
```

---

- IMPLEMENTAR INTERATIVIDADE

Para implementar a interatividade nos nossos *sites*, precisamos nos fazer três perguntas:

- **QUEM?**

Essa pergunta se refere aos **elementos que participarão da interatividade**. Geralmente, temos **dois elementos**:

1. um que **ativa** a interatividade
2. outro que **sofre** as alterações.

No código, esse passo equivale a **capturar os elementos do DOM** que serão usados.

```
let botao = document.querySelector("button");  
  
let texto = document.querySelector("p");
```



- **O QUE?**

Com essa pergunta, **definimos o comportamento da interatividade usando a lógica de programação**. Geralmente, está *encapsulado dentro de uma função*.

```
function mudarCor( ){  
  texto.style.color = "blue"  
}
```

- **QUANDO?**

Essa pergunta se refere ao **tipo de evento que acionará a interatividade**. Por exemplo: `click`, input ganhando `foco`, pressionar uma tecla ( `keyUp`, `keyPress` ou `keyDown`), entre outros.

- Método `.addEventListener()`

Recebe **dois** argumentos:

1. uma `string` com o **tipo de evento**
2. uma `função`, que define o **comportamento** a ser executado quando o evento for chamado.

```
botao.addEventListener("click", mudarCor)
```

**É IMPORTANTE:**

Lembrar que, ao chamar as funções que definem o comportamento como segundo argumento, **não abrimos nem fechamos os parênteses** depois do nome delas.

Também é possível definir o comportamento DA FUNÇÃO diretamente no segundo argumento do `addEventListener`. Para isso, usamos funções anônimas comuns ou `()=>` 'arrow functions'.

```
botao.addEventListener("click", ( ) => {  
  
    texto.style.color = "blue"})
```

## ◦ EVENTOS

Temos vários **eventos** que nos permitem *definir com precisão os gatilhos das interatividades* que queremos implementar. As três principais categorias são eventos de mouse, de teclado e de formulário.

Nome

Primeiro Último

Departamento

Nome da universidade

Período

Os eventos de formulário nos ajudam a validar nossos formulários antes de enviar as informações inseridas.

- Eventos de *mouse* eventos gerados por algum tipo de interação do usuário usando o mouse como referência.

Evento	Descrição
mouseover	Quando o cursor do mouse passa <b>por cima de um elemento</b>
mouseout	Quando o cursor do mouse <b>deixa de estar "em cima"</b> de um elemento
click	Quando <b>pressionamos e soltamos o botão esquerdo do mouse</b>
dblclick	Quando <b>clicamos duas vezes com o botão esquerdo</b> do mouse

## Leitura Complementar

FACUL IV2. JavaScript - addEventListener. YouTube, 25 fev. 2022.

MATHEUS BATTISTI - HORA DE CODAR. Curso JavaScript #50 - Eventos mouseover e mouseout. YouTube, 30 ago. 2020.



### 6.6.3 - Revisão do módulo - JS II - Parte02

Os **eventos** que usamos para executar comportamentos no nosso site também **possuem propriedades**. Sendo assim, podemos usá-las para personalizar ainda mais as interatividades.

- Capturar informações do evento;
- Eventos de teclado;
- Funções `preventDefault()` e `alert()`;
- Eventos de formulário

#### 6.6.3.1 - Informações do evento

Essas propriedades **fornecem informações específicas sobre o evento**, como o momento em que foi acionado, o elemento a partir do qual foi chamado, a posição do mouse quando ele foi executado etc.

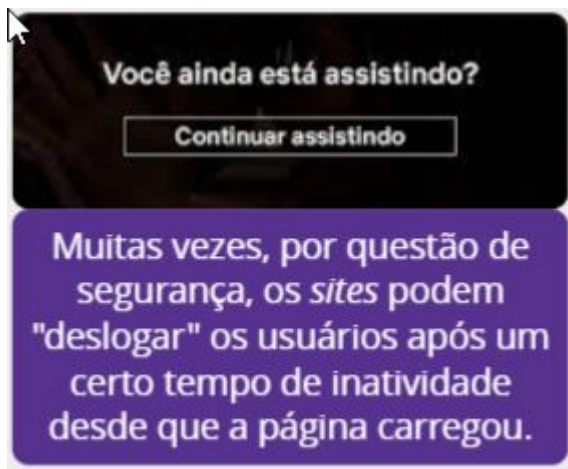
Para acessá-las, devemos definir um parâmetro na função que chamamos dentro do `addEventListener` que representa o evento em si.

```
botaoTempo.addEventListener("click", (evento) => {  
  
});
```

Além disso, podemos acessar propriedades do **evento** usando a **dot notation**.

```
botaoTempo.addEventListener("click", (evento) => {  
  console.log(evento.timeStamp)  
})
```

A propriedade `timeStamp` retorna um número que representa a quantidade de milissegundos desde a renderização da página até o momento em que o evento aconteceu.



```
botaoTempo.addEventListener("click", (evento) => {  
  console.log(evento.timeStamp)
```

```
})
```

A propriedade **target** nos permite acessar as informações e propriedades tanto do elemento que ativou o evento quanto dos seus elementos filhos. Para ativá-la, usamos:

```
botaoCor.addEventListener("click", (evento) => {  
  console.log(evento.target)  
})
```

### 6.6.3.2 - Eventos do Teclado

Os **eventos de teclado** são muito populares na área de desenvolvimento de videogames.

Esses tipos de eventos nos permitem adicionar interatividades mais sofisticadas aos nossos sites, como navegar usando os controles numéricos ou fechar elementos usando a tecla escape.

Evento	Descrição
keypress	Evento que <b>só reconhece</b> as teclas de <b>letras, números</b> ou <b>pontuação</b> , desconsiderando teclas como SHIFT, ALT, setas etc.
keydown	Evento disparado ao <b>pressionar uma tecla</b> . Ele é executado de forma repetida se a tecla permanecer pressionada.
keyup	Evento disparado <b>ao soltar uma tecla</b> após pressioná-la.

```
document.addEventListener("keyup", (e)=> {  
  console.log(e.key);  
  console.log(e.code);  
})
```

#### 6.6.3.2.1 - Propriedade **key**

Essa propriedade representa **o valor que a tecla guarda** e não necessariamente é um valor único.

Por exemplo, a tecla 1 do Numpad e a tecla 1 do teclado alfanumérico **guardam o mesmo valor** na propriedade **key**.

#### 6.6.3.2.2 - Propriedade **code**

Essa propriedade **guarda um código único** que *representa a tecla* e **não é repetido por outra tecla**. Podemos comparar com o atributo **id** dos elementos HTML.

Por exemplo, a tecla 1 do Numpad e a tecla 1 do teclado alfanumérico guardam valores distintos na propriedade **code**.

### 6.6.3.3 - Funções nativas

aprendemos também sobre o uso de duas funções nativas da linguagem JavaScript para **aplicarmos em conjunto com os eventos de formulário**.

### 6.6.3.4 - preventDefault()

O método `preventDefault()` **impede o comportamento padrão de qualquer elemento capturado do DOM**, como a navegação de links com a tag `<a>` e o envio de formulários.

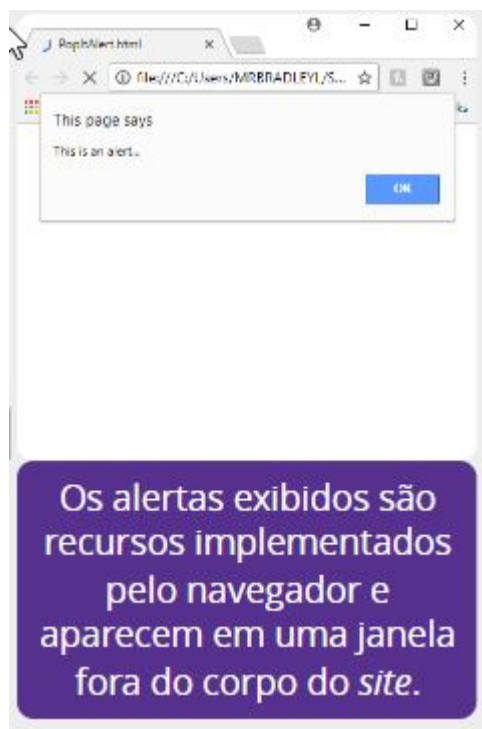
Esse método é acessado e executado a partir do parâmetro que representa o **evento nas funções**, que foram passadas como segundo parâmetro do `addEventListener`.

```
link.addEventListener("click", (evento) => {  
    evento.preventDefault()  
})
```

### 6.6.3.5 - alert()

A função `alert()` é **usada para exibir uma mensagem de feedback ao usuário**.

Para usá-la, basta executá-la passando uma string, que contém a mensagem que quer exibir, como argumento.



muito comum usar essas mensagens de feedback para informar aos usuários quando não foi possível executar uma ação do site ou quando uma ação foi executada de forma satisfatória, entre outras formas.

### 6.6.3.6 - Eventos de formulário

Eventos de formulário são usados, principalmente, para implementar ações de validação nos nossos formulários. Assim, evita o envio de informações erradas, informa aos usuários a inserção errada de dados e o status de envio de informações.

Evento	Descrição	Ativação
focus	Quando o <b>elemento é focado</b> pelo usuário	Dizemos que um elemento "ganha foco" quando habilitamos a inserção de dados, geralmente clicando nele;
blur	Quando um <b>elemento perde o foco</b> do usuário	
change	Quando um <b>elemento teve seu valor alterado</b> após perder o foco	Os eventos changes são executados de forma um pouco diferente, dependendo do tipo de input e como os dados são inseridos;
submit	Quando <b>enviamos as informações do formulário</b>	Os eventos do tipo submit devem ser executados no elemento do formulário em si, não no botão com tipo submit.

#### Tabela Estudo Dirigido

CONECTAR	EXPANDIR	DESAFIAR
Como os conteúdos trabalhados aqui te conectam com o que vc já sabia?	Como os conteúdos trabalhados aqui expandem o que você já sabe?	Quais conteúdos ainda te desafiam ou te confundem?

#### Leitura Complementar

[MATHEUS BATTISTI - HORA DE CODAR. Curso JavaScript #51 - Eventos keydown e keyup. YouTube, 30 ago. 2020.](#)

[ALÉM DO CÓDIGO. Como fazer VALIDAÇÃO DE FORMULÁRIO com Javascript | Validação Formulário Javascript.](#)

#### 6.6.2.1 - Anotações Exercícios

#### 6.6.2.2 - Anotações Exercícios

### 6.7 - MÓDULO 07

6.7.1 - Revisão Manipular estilos e classes com JS

### 6.8 - MÓDULO 08

6.7.1 - Revisão Eventos I