# Javascript

## Introdução à Manipulação de Eventos

# Eventos

## Trabalhando com *Event Handlers*

# Evento

- Eventos são disparados para avisar que algo aconteceu em uma aplicação.
  - Tratadores podem ser atribuídos aos eventos. Esses tratadores de eventos, ou *event handlers* são funções executadas quando eventos são disparados.

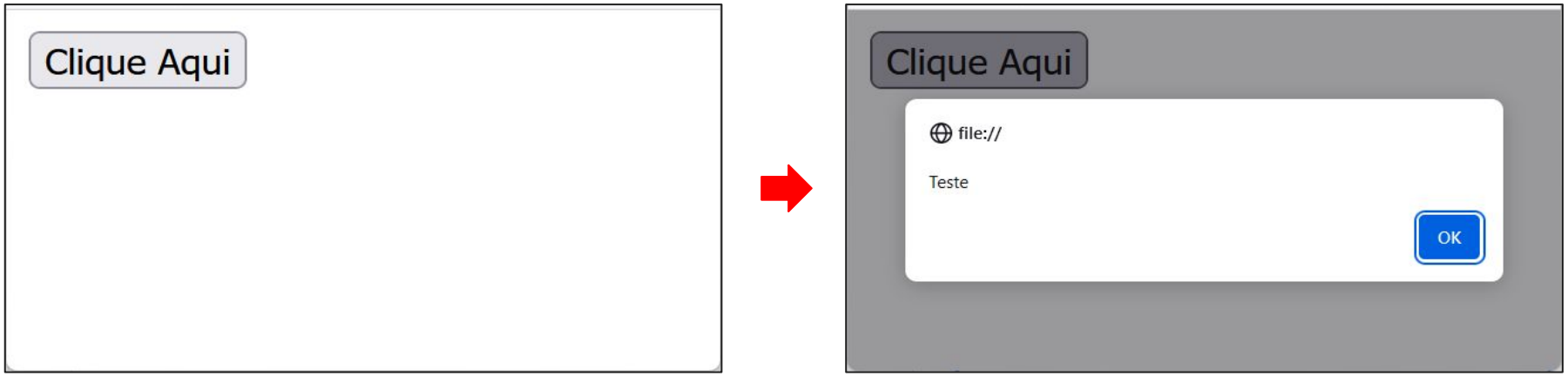
- Os elementos do DOM podem disparar eventos.
  - É importante destacar que eventos não são restritos ao DOM.

# Atribuindo *Event Handlers*

- Um *event handler* pode ser atribuído por meio dos atributos que representam cada evento:

```
<!DOCTYPE HTML>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
  <body>
    <button onclick="alert('Teste')">Clique
Aqui</button>;
  </body>
</html>
```

# Atribuindo *Event Handlers*



# Atribuindo *Event Handlers*

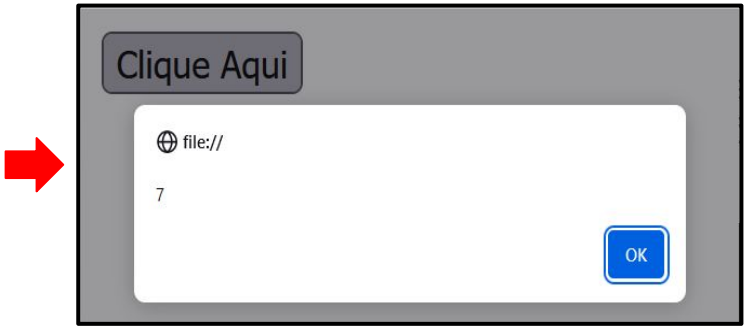
- Como adicionar todo código *Javascript* dentro do atributo não é adequado, é possível chamar funções próprias no *Javascript*:

index.html:

```
<!DOCTYPE HTML>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <script src="testes.js" defer></script>
  <title></title>
</head>
  <body>
    <button onclick="somar(5, 2)">
        Clique Aqui
    </button>
  </body>
</html>
```

teste.js:

```
const somar = (num1, num2) => {
  alert(num1 + num2);
};
```



# Atribuindo *Event Handlers*

- É possível adicionar um *event handler* pela propriedade que contém o nome do evento:

index.html:

```
<!DOCTYPE HTML>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <script src="testes.js" defer></script>
  <title></title>
</head>
  <body>
    <button id="botao">Clique Aqui</button>
  </body>
</html>
```

teste.js:

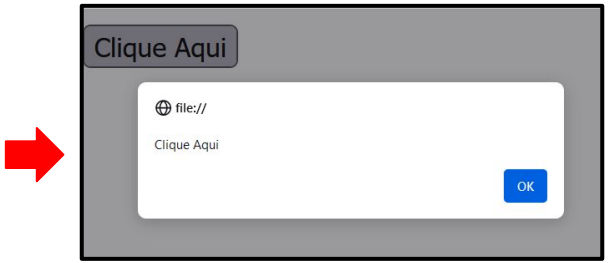
```
const apresentarSoma = (num1, num2) => {
  alert(num1 + num2);
};

const botao =
      document.getElementById('botao');

if (botao) {
  botao.onclick = () => {
    return apresentarSoma(5, 5);
  };
}
```

# Atribuindo *Event Handlers*

- Dentro do event handler, this representa o próprio elemento do DOM em questão:

index.html:

```
<!DOCTYPE HTML>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <script src="testes.js" defer></script>
  <title></title>
</head>
  <body>
    <button onclick="alert(this.innerHTML)">
        Clique Aqui
    </button>
  </body>
</html>
```



# Múltiplos *Event Handlers*

- Os métodos para atribuição de *event handlers* mostrados até este momento permitem apenas um *event handler* por evento.

- Para adicionar mais de um *event handler* por evento, é possível utilizar o método *addEventListener* do elemento do *DOM* em questão.

- Para remover um *event handler*, você pode utilizar o método *removeEventListener* do elemento do *DOM* em questão, porém é necessário ainda possuir a referência do *event handler* para isso.

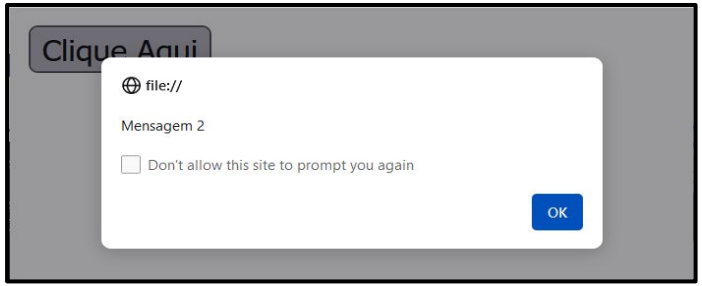
# Múltiplos *Event Handlers*

- Parâmetros da função *addEventListener*:
  - *event*: nome do evento
  - *handler*: função que será executada quando o evento for disparado.
  - *options*: parâmetro opcional que pode possuir propriedades, dentre as quais detacam-se:
    - *once*: se possuir o valor *true*, o *event handler* será automaticamente removido após a sua primeira execução.
    - *passive*: se *true*, o *handler* não chamará o método *preventDefault()*. Este método será explicado mais tarde.

# Múltiplos *Event Handlers*

teste.js:

```
const botao = document.getElementById('botao');

const tratador1 = () => {
  alert("Mensagem 1");
};

const tratador2 = () => {
  alert("Mensagem 2");
};

if (botao) {
  botao.addEventListener("click", tratador1);
  botao.addEventListener("click", tratador2);
}
```

# Removendo *Event Handlers*

- Teste esta alteração clicando uma primeira vez e, após os alertas, clicando uma segunda vez no botão.

```
const botao = document.getElementById('botao');

const tratador1 = () => {
  alert("Mensagem 1");
};

const tratador2 = () => {
  alert("Mensagem 2");
  botao.removeEventListener("click", tratador2);
};

if (botao) {
  botao.addEventListener("click", tratador1);
  botao.addEventListener("click", tratador2);
}
```

# *Event Object*

- Até o momento, nossos *event handlers* não possuem acesso a nenhum dado sobre o evento disparado. No caso do evento *click*, por exemplo, não conhecemos nada sobre as coordenadas do mouse.

- Para resolver isso, podemos receber no *event handler* um parâmetro que é um objeto contendo estas informações.

# Event Object

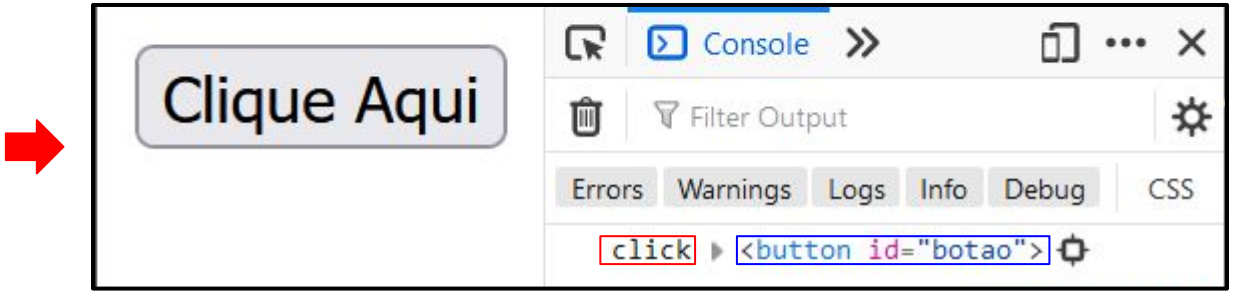
- Recebendo informações sobre o evento:

```
const botao = document.getElementById('botao');

const tratador1 = (event) => {
  console.log(event);
};

if (botao) {
  botao.addEventListener("click", tratador1);
}
```

# Event Object

- Conhecendo qual evento foi disparado e qual objeto.

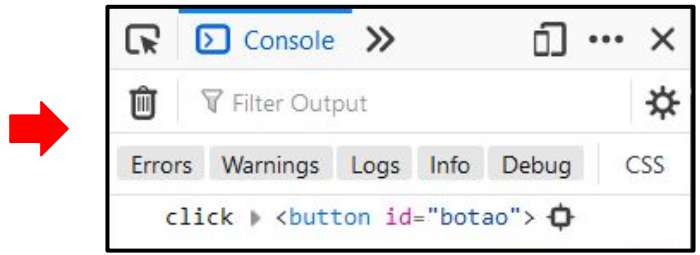
```
const botao = document.getElementById('botao');

const tratador1 = (event) => {
  alert(event.type + " em " + event.currentTarget);
};

if (botao) {
  botao.addEventListener("click", tratador1);
}
```



# Object Handlers

- Além de passar uma função para o método *addEventListener*, também é possível passar um objeto.
  - Neste caso, quando o evento for disparado, o método *handleEvent* do objeto será chamado.

```
const botao = document.getElementById('botao');

let objeto = {
  handleEvent(event) {
    console.log(event.type, event.currentTarget);
  }
};

if (botao) {
  botao.addEventListener("click", objeto);
}
```



# *Bubbling*

## Propagação de evento

# Bubbling

- Quando um evento ocorre, ele primeiro executa o *handler* deste elemento e, depois, o *handler* do elemento pai e continua executando os *handlers* dos ancestrais.

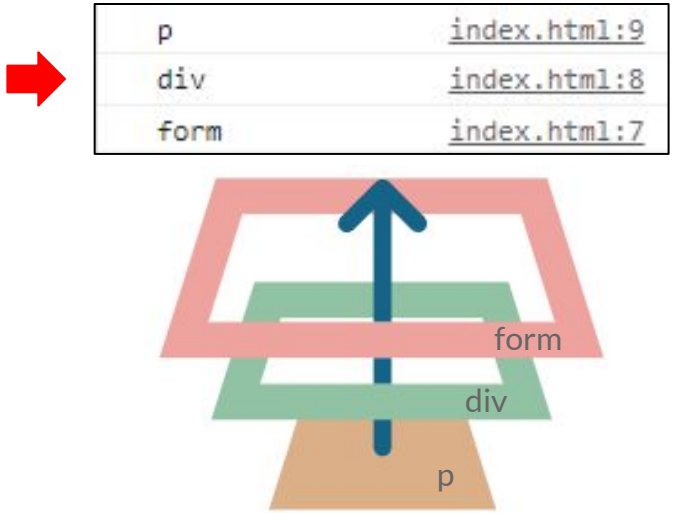
```
<form onclick="console.log('form')">
    <div onclick="console.log('div')">
        <p onclick="console.log('p')">
            Teste
        </p>
    </div>
</form>
```

```
p                    index.html:9
div                  index.html:8
form                 index.html:7
```

# Bubbling

- Quando um evento ocorre, ele primeiro executa o *handler* deste elemento e, depois, o *handler* do elemento pai e continua executando os *handlers* dos ancestrais.

```
<form onclick="console.log('form')">
    <div onclick="console.log('div')">
        <p onclick="console.log('p')">
            Teste
        </p>
    </div>
</form>
```



# Conhecendo o chamador do evento

- Quando um evento ocorre, ele primeiro executa o *handler* deste elemento e, depois, o *handler* do elemento pai e continua executando os *handlers* dos ancestrais.

```
let p = document.getElementsByTagName('p')[0];
let div = document.getElementsByTagName('div')[0];
let form = document.getElementsByTagName('form')[0];
const clicar = (event) => {
    console.log(event.target);
};
p.addEventListener('click', clicar);
div.addEventListener('click', clicar);
form.addEventListener('click', clicar);
```

```
<form>
    <div>
        <p>
            Teste
        </p>
    </div>
</form>
```

# *stopPropagation*

- Qualquer handler pode decidir que o evento foi totalmente processado e interromper o *bubbling*.

```
let p = document.getElementsByTagName('p')[0];
let div = document.getElementsByTagName('div')[0];
let form = document.getElementsByTagName('form')[0];
const clicar = (event) => {
    console.log(event.target);
    event.stopPropagation();
};
p.addEventListener('click', clicar);
div.addEventListener('click', clicar);
form.addEventListener('click', clicar);
```

```
<form>
    <div>
        <p>
            Teste
        </p>
    </div>
</form>
```

# Exercício

Faça uma página que contenha um parágrafo e um botão. O parágrafo deve possuir o conteúdo “0 cliques” (zero cliques). A cada vez que o botão for clicado, você deve adicionar 1 à quantidade de cliques, criando um contador de cliques.

# Fim de material

## Dúvidas?