

INSTITUTO FEDERAL

Goiano

Campus Morrinhos

JAVASCRIPT EVENTOS JS

PROFESSOR MARCEL MELO

MARCEL.MELO@IFGOIANO.EDU.BR

Eventos em *javascript* são disparados por alguma ação, tanto do usuário quanto da própria página.

- Clique com o mouse
- Tecla do teclado pressionada
- Entrar ou sair de um campo do formulário

Qualquer evento js pode ser **capturado e tratado** pelos **manipuladores de eventos**

Tais manipuladores aceitam **uma função** para detalhar o que será feito quando tal evento for disparado.

Algo muito importante em Javascript é aprender a escutar a eventos que são disparados. Cada evento disponível possui um **manipulador de evento**, que é um bloco de código (função) que será executado quando o evento for disparado.

Praticamente todos elementos de uma página, incluindo a própria página, pode disparar algum tipo de evento.

Às vezes, o aplicativo dispara eventos automaticamente, como quando a página é carregada; às vezes, o aplicativo dispara eventos como uma reação a interação, como é o caso do click em um botão.

```
<button id="enviar" onClick="validar()" value="Enviar"/>
```

- **abort** - Quando uma **imagem** não pode ser carregada
- **blur** - Quando sai em um **campo do formulário**
- **focus** – Quando entra de um **campo do formulário**
- **change** - Quando altera o valor de uma **opção do formulário**
- **click** - Um clique com o mouse (**maioria dos elementos**)
- **dblclick** – Dois cliques com o mouse (**maioria dos elementos**)
- **contextmenu** - Clique com o botão direito do mouse na **página web**
- **error** - Quando a **página** ou uma **imagem** não pode ser carregada
- **keyup** - Para de pressionar uma **tecla do teclado**
- **keydown** – Pressionar uma **tecla do teclado**
- **keypress** - Pressionar uma **tecla do teclado**

- **load** - Quando a **página ou imagem** termina de seu carregamento
- **unload** – Ao sair da **página ou imagem** não consegue ser carregada
- **mousedown** - Pressionar o mouse (**maioria dos elementos**)
- **mouseup** – Para de pressionar o mouse (**maioria dos elementos**)
- **mouseover** - Quando o mouse está sobre algo; (**maioria dos elementos**)
- **mouseout** - Quando o mouse deixa de estar sobre algo; (**maioria dos elementos**)
- **reset** - Zerar os campos do **formulário**
- **resize** - A **janela** é redimensionada
- **select** - Seleção de texto em uma **área de texto ou entrada de texto**
- **scroll** - Quando rola a **página, quadro ou elemento** com overflow (scroll)
- **submit** - Quando envia um **formulário**

MANIPULADOR DE EVENTOS – INLINE

MANIPULADOR DE EVENTOS – INLINE

Os manipuladores de eventos Inline são adicionados como atributos diretamente nos elementos HTML. Eles recebem o prefixo “on” antes do nome do evento.

- **onload**
- **onclick**
- **onsubmit**

```
<button id="botao" onclick="validarForm()">Salvar</button>
```

```
<div id="quadrado" ondblclick="preto()" onclick="vermelho()" onmouseout="verde()" onmouseover="azul()">
```

IMPORTANTE!!!!

Essa forma de atribuição de manipuladores de eventos é totalmente desaconselhável!

MANIPULADOR DE EVENTOS – INLINE

```
function validarForm() {  
  
    let nome = document.getElementById("nome").value;  
    let idade = document.querySelector("#idade").value;  
  
    alert('Nome: ' + nome);  
    alert('Idade: ' + idade);  
}
```


MANIPULADOR DE EVENTOS – INLINE

- Este método de utilização de eventos em JS não é recomendável!!!

Inicialmente é uma péssima ideia misturar o seu código HTML e o seu código Javascript, pois será difícil de analisar e manter seu código fonte.

Caso você tenha vários elementos HTML que possuem o mesmo manipulador de evento você terá que replicar esse código em cada um destes elementos HTML. E se esse manipulador precisar ser alterado ?

Todo código Javascript, incluindo os manipuladores de eventos, devem estar nos arquivos javascripts.

MANIPULADOR DE EVENTOS – NÍVEL 0

Outro problema da abordagem Inline é não permitir utilizar um manipulador de eventos por vez.

```
<body>
  <label>Nome:
  |   <input type="text" id="nome" />
</label>
<br>
<label>Idade:
  |   <input type="number" id="idade"/>
</label><br>
<button id="botao" onclick="validarForm()">Salvar</button>
<div id="quadrado" ondblclick="preto()" onclick="vermelho()" onmouseout="verde()" onmouseover="azul()"></div>
</body>
```

EXEMPLO

```
function validarForm() {  
  
    let nome = document.getElementById("nome").value;  
    let idade = document.querySelector("#idade").value;  
  
    alert('Nome: ' + nome);  
    alert('Idade: ' + idade);  
}
```

EXEMPLO

```
function azul() {  
    document.getElementById("quadrado").style.backgroundColor = 'blue';  
}
```

```
function verde() {  
    document.getElementById("quadrado").style.backgroundColor = 'green';  
}
```

```
function vermelho() {  
    document.getElementById("quadrado").style.backgroundColor = 'red';  
}
```

```
function preto() {  
    document.getElementById("quadrado").style.backgroundColor = 'black';  
}
```

***PROPRIEDADE DO
MANIPULADOR DE EVENTOS***

MANIPULADOR DE EVENTO COMO PROPRIEDADE

Essa é a forma mais simples e correta de adicionar um manipulador de evento a um elemento HTML.

MANIPULADOR DE EVENTO COMO PROPRIEDADE

```
<body>
  <label>Nome:
  |   <input type="text" id="nome" />
  </label>
  <br>
  <label>Idade:
  |   <input type="number" id="idade"/>
  </label><br>
  <button id="botao">Salvar</button>
  <div id="quadrado"></div>
</body>
```


MANIPULADOR DE EVENTO COMO PROPRIEDADE

```
let elemento = document.querySelector('#quadrado')  
elemento.onclick = vermelho;  
elemento.ondblclick = preto;  
elemento.onmouseout = verde;  
elemento.onmouseover = azul;
```

```
document.querySelector("#botao").onsubmit = validarForm;
```


MANIPULADOR DE EVENTOS – NÍVEL 2

Cada objeto apresenta três métodos:

- **addEventListener** – Adiciona o tratamento de evento a um elemento
- **removeEventListener** – Remove o tratamento de evento a um elemento
- **dispatchEventListener** – Enviar um novo evento a um elemento

```
objeto.addEventListener('Evento', 'função', true ou false);
```

O primeiro parâmetro deve ser o evento (sem o prefixo "**on**"), o segundo será a função utilizada para tratar tal evento, o terceiro será *true* para transformar o *listener* em um modelo de captura de eventos em **cascata**, e *false* para transformá-los em **eventos de propagação**.

MANIPULADOR DE EVENTOS - NÍVEL 2

```
<body>
  <label>Nome:
    <input type="text" id="nome" />
  </label>
  <br>
  <label>Idade:
    <input type="number" id="idade"/>
  </label><br>
  <button id="botao">Salvar</button>
  <div id="quadrado"></div>
</body>
```

MANIPULADOR DE EVENTOS – NÍVEL 2

```
document.getElementById("botao").addEventListener('click', validarForm, false);  
document.getElementById("quadrado").addEventListener('click', vermelho, false);  
document.getElementById("quadrado").addEventListener('dblclick', preto, false);  
document.getElementById("quadrado").addEventListener('mouseover', azul, false);  
document.getElementById("quadrado").addEventListener('mouseout', verde, false);
```

MANIPULADOR NÍVEL 2 E REMOVER LISTENER

Uma das grandes vantagens do manipulador de eventos de nível 2 é a possibilidade de remover um manipulador de evento a qualquer momento. Para isso utiliza-se o método **.removeEventListener()**.

```
document.querySelector("#botao").removeEventListener('click', validarForm);
```

***PROBLEMAS COM O
CARREGAMENTO DA PÁGINA***

CARREGAMENTO DE PÁGINA

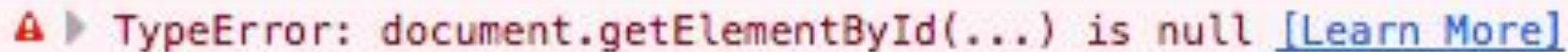
Ao usar o manipulador de eventos em arquivos externos, durante o carregamento da tela o arquivo de javascript pode ser carregado **antes** de todos elementos da tela terem sido de fato carregados.

Ao tentar recuperar o elemento HTML, pode ser que este ainda não esteja na tela e a função utilizada para recuperar tais elementos retorne **null**

Quando isso acontece, o **listener** não é adicionado ao elemento e a página HTML perde toda sua dinâmica.

MANIPULADOR NÍVEL 2 E CARREGAMENTO DE PÁGINA

Quando isso acontece, ao inspecionar a tela teremos o seguinte erro:



```
▲ ▶ TypeError: document.getElementById(...) is null \[Learn More\]
```

Assim, deve-se garantir que ao tentar recuperar qualquer elemento da tela, este já deve ter sido carregado na tela.

Ao mesmo tempo, os **listeners** devem ser adicionados ao final de carregamento da tela, garantindo que quando o usuário realizar um evento, a tela responda com a função apropriada.

Criar uma função de inicialização dos tratadores de eventos da tela, todos tratadores só serão vinculados aos elementos HTML ao final do carregamento da tela.

```
window.addEventListener('load', init);
```

```
function init() {  
    document.getElementById("botao").addEventListener('click', validarForm, false);  
    document.getElementById("quadrado").addEventListener('click', vermelho, false);  
    document.getElementById("quadrado").addEventListener('dblclick', preto, false);  
    document.getElementById("quadrado").addEventListener('mouseover', azul, false);  
    document.getElementById("quadrado").addEventListener('mouseout', verde, false);  
}
```

PROPAGACÃO DE EVENTOS

PROPAGAÇÃO DE EVENTOS

O terceiro parâmetro do `addEventListener` é um valor booleano que define se o evento vai ser tratado em **cascata** ou simplesmente **propagação de eventos**.

- Se o valor for *true*, transforma o *listener* em um modelo de captura de eventos em **cascata**, e *false* o transforma em **eventos de propagação**.

Quando clica-se em uma página web, não está clicando apenas no documento, mas também em um link, ou talvez em um elemento `div`.

Na maioria dos casos, não precisa se preocupar com o contêiner que inclui o link porque é mais **provável** que você configure o manipular de eventos para **apenas um elemento**

O que acontece, se você configurar o mesmo manipulador de eventos para múltiplos elementos aninhados ?

Em que ordem eles são disparados e como se evita o disparo do manipulador de eventos se só quiser que um elemento seja afetado de cada vez ?

A abordagem para gerenciar eventos em uma pilha de elementos é conhecida como propagação de eventos.

PROPAGANÇA DE EVENTOS

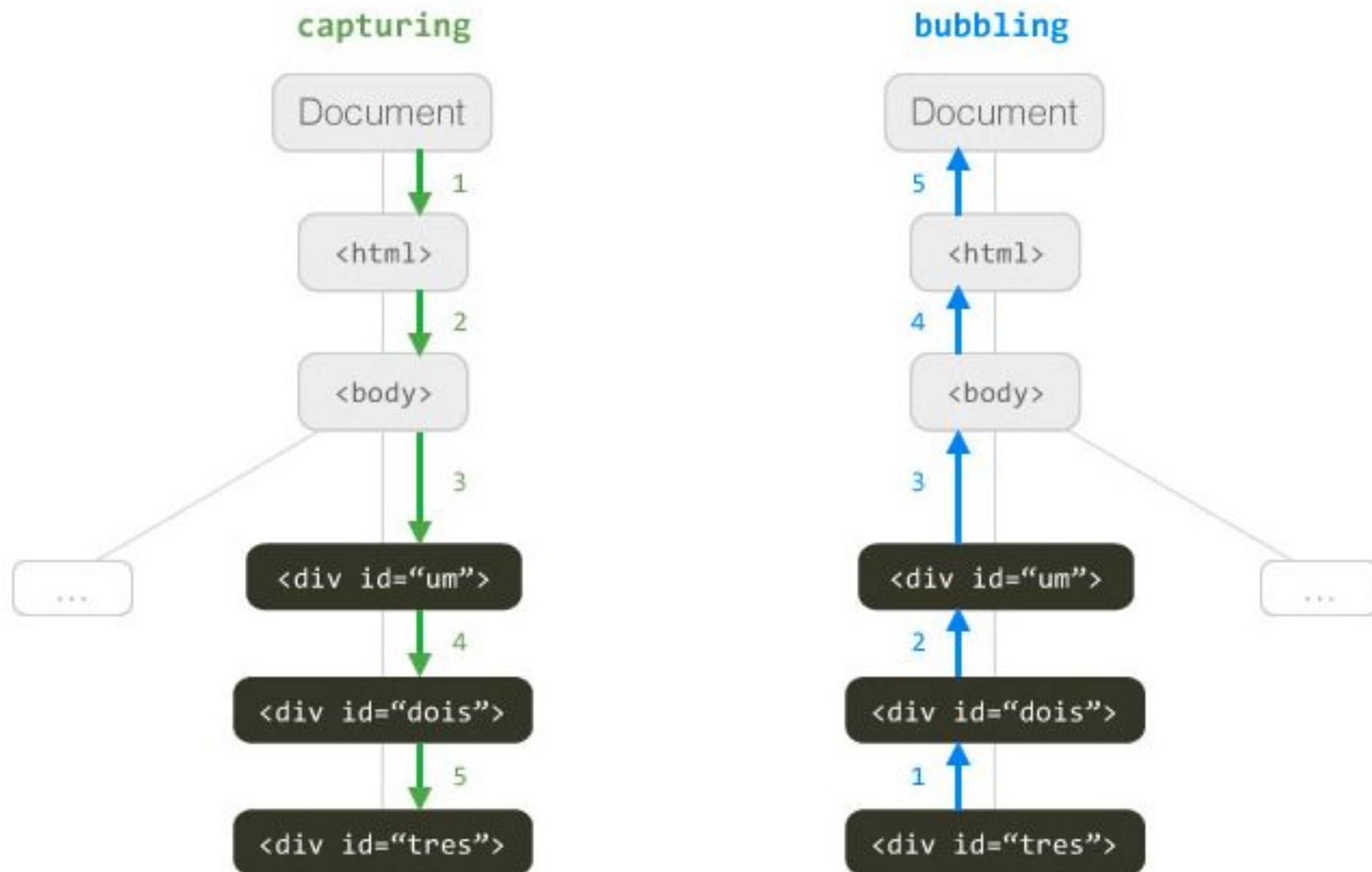
Na **propagação de eventos**, o elemento mais interno dispararia seu evento, seguido pelo próximo da pilha, e assim por diante, até alcançar o mais externo.

- Efeito *bubbling*

Se manipuladores de eventos fosse atribuídos a todos esses elementos, todos os seus elementos disparariam por sua vez.

No efeito **Cascata**, o elemento mais externo é disparado primeiro até que o elemento mais interno (que recebeu o evento) seja de fato disparado (último)

PROPAGAÇÃO DE EVENTOS



PROPAGAÇÃO DE EVENTOS

```
<div id="um" onclick="eventoUm()">
  <div id="dois" onclick="eventoDois()">
    <div id="tres" onclick="eventoTres()">
      <a id="google" onclick="clickGoogle()" href="http://www.google.com">Vai pro Google</a>
    </div>
  </div>
</div>
```

```
document.getElementById("um").addEventListener('click', eventoUm, false);
document.getElementById("dois").addEventListener('click', eventoDois, false);
document.getElementById("tres").addEventListener('click', eventoTres, false);
document.getElementById("google").addEventListener('click', clickGoogle, false);
```


PROPAGAÇÃO DE EVENTOS

```
function eventoUm() {  
    alert("div 1");  
}
```

```
function eventoDois() {  
    alert("div 2");  
}
```

```
function eventoTres(event) {  
    alert("div 3");  
}
```

```
function clickGoogle(event) {  
    alert("Google");  
}
```


PROPAGAÇÃO DE EVENTOS

```
document.getElementById("um").addEventListener('click', eventoUm, true);  
document.getElementById("dois").addEventListener('click', eventoDois, true);  
document.getElementById("tres").addEventListener('click', eventoTres, true);  
document.getElementById("google").addEventListener('click', clickGoogle, true);
```

PROPAGAÇÃO DE EVENTOS

- Os eventos serão executados de dentro para fora (propagação de eventos). Neste caso, quando eu clicar na **div#um**, tem-se apenas um alerta, porém, se eu clicar na **div#dois**, tem-se dois alertas, e clicando na **div#tres**, tem-se três alertas.

Isso ocorre porque a **div#tres** está dentro das **div#um** e **div#dois** e todas **divs** tratam o evento de click. Então ao clicar na **div#tres** indiretamente também está clicando nas **div#um** e **div#dois**.

Caso queira parar a propagação de eventos, basta utilizar a opção **event.stopPropagation()**

Para isso, deve configurar como parâmetro da função de tratamento do evento para receber o objeto event.

```
function eventoDois(event) {  
    alert("div 2");  
    event.stopPropagation();  
}
```

PROPAGAÇÃO DE EVENTOS E O OBJECT THIS

- Dentro de um manipulador de eventos, uma forma de acessar as propriedades do elemento que o contém em JS é usar a palavra-chave **this**.
 - A palavra-chave **this** representa o proprietário da função ou método sendo processado.
- Para uma **função global**, **this** representa a **janela**. Para um **método de objeto**, **this** representa a **instância do objeto**. E, em um **manipulador de eventos**, **this** representa o **elemento que recebeu o evento**.
- Usar **this** é uma forma útil de acessar **valores do formulário após eventos**, sem ter que seguir o caminho do **document**, **nomes do formulário**, **nome dos campos**, e assim por diante.

PROPAGAÇÃO DE EVENTOS E O OBJECT THIS

```
function azul() {  
    this.style.backgroundColor = 'blue';  
}
```

```
function verde() {  
    this.style.backgroundColor = 'green';  
}
```

```
function vermelho() {  
    this.style.backgroundColor = 'red';  
}
```

```
function preto() {  
    this.style.backgroundColor = 'black';  
}
```

OBJETO EVENT

- **Um objeto Event é associado a todos eventos. Ele tem propriedades que fornecem informações sobre o evento, como:**
 - Localização de um clique do mouse na página web

Para acessar o objeto Event, basta adicionar um parâmetro em sua função de tratamento de eventos. O Javascript injeta esse objeto de forma automática no parâmetro e te permite usá-lo como desejar.

OBJECT EVENT

```
function eventoDois(event) {  
    alert("div 2");  
    event.stopPropagation();  
}
```

```
function azul(event) {  
    this.style.backgroundColor = 'blue';  
    console.log(event.screenX) //X onde o cursor entrou na div  
}
```


PROPRIEDADES DO OBJECT EVENT

- **altKey** – Boolean se a tecla ALT estiver pressionada na hora do evento
- **clientX** – Coordenada X do cliente do evento
- **clientY** – Coordenada Y do cliente do evento
- **ctrlKey** – Boolean se a tecla CTRL estiver pressionada na hora do evento
- **keyCode** – Código (número) da tecla pressionada
- **screenX** – Coordenada X da tela do evento
- **screenY** – Coordenada Y da tela do evento
- **shiftKey** – Boolean se a tecla Shift estiver pressionada na hora do evento
- **type** – O tipo do evento

EVITAR COMPORTAMENTO PADRÃO

EVITAR COMPORTAMENTO PADRÃO

Muitas vezes, ao tratarmos algum evento via Javascript pode ser necessário que o comportamento padrão do elemento HTML não seja executado em sua totalidade.

- **Formulário**

- **Comportamento padrão:** Enviar dados para o servidor web (PHP)
- **Exceção:** Caso o formulário não esteja válido, os dados não podem ser enviados incorretos.

- **Link**

- **Comportamento padrão:** Ao ser clicado, o usuário é redirecionado para o endereço indicando no atributo **href**
- **Exceção:** De acordo com alguma regra definida, não é desejado que o usuário seja redirecionado.

EVITAR COMPORTAMENTO PADRÃO

Quando, por algum motivo, estivermos tratando algum evento específico e não se desejar que o comportamento padrão de qualquer elemento seja executado (após a execução da sua função de tratamento de erros) deve-se usar a função **event.preventDefault()** do objeto **event**

```
function clickGoogle(event) {  
    alert("Google");  
    event.preventDefault();  
}
```

Nesse caso, ao clicar no link o usuário não será redirecionado para o site do Google, pois em nossa função que trata o evento de click no link solicitamos que o comportamento padrão do link não seja executado.

GERAR EVENTOS PELO JS

- Eventos geralmente iniciam quando alguém faz algo na página web.
 - **Pressiona um botão**
 - **Clica em um link**
 - **Faz uma seleção**
- Às vezes, porém, pode-se querer disparar um evento em uma página ou elemento.
- Para disparar este evento, o evento deve estar associado ao tipo do elemento.
 - Pode disparar um clique em um botão do formulário, mas não em um campo de entrada de texto.

GERAÇÃO DE EVENTOS

- Para gerar o evento click para o botão do formulário, use o evento click e chame o objeto click do método

```
<input type="button" id="enviar" value="Enviar"/>
```

```
document.getElementById("enviar").click();
```

- Um motivo para chamar um evento diretamente é usar o evento focus em um campo de entrada para mover o cursor para esse campo

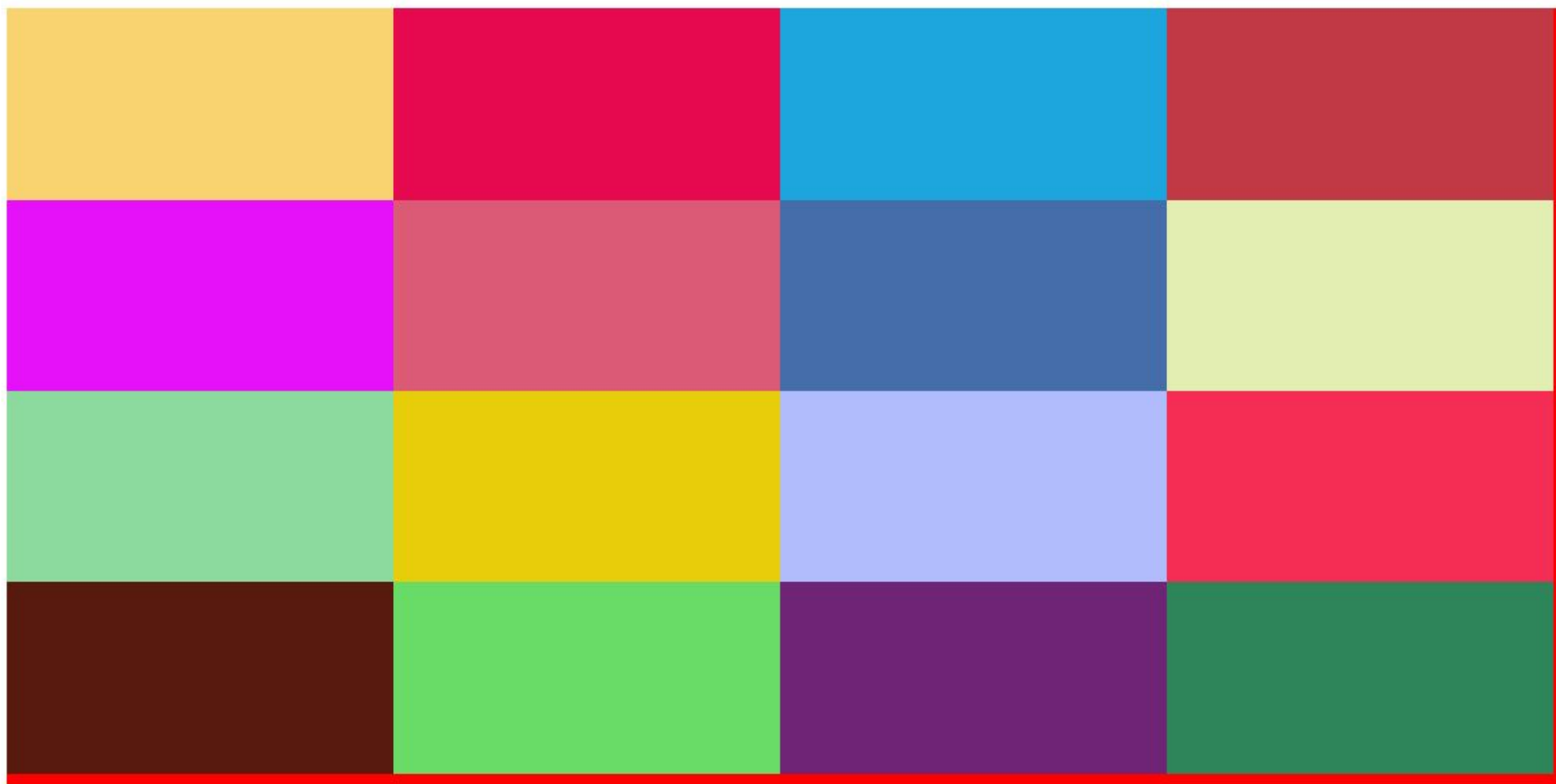
Em uma página web, crie uma div principal de tamanho 1000px de largura por 500px de altura.

Divida essa div principal em 16 pequenas divs em seu interior, tentando simular uma matriz com tamanho de 4 divs de largura por 4 divs de altura. Coloque cada uma desses divs com 25% da altura e largura.

Ao final, para cada uma das divs internas, adicione um manipulador de eventos qualquer (click, mouseover, mouseout....). Ao ser disparado, este evento deve criar uma cor aleatória e adicioná-la no background da div que disparou o evento.

EXERCÍCIOS

EXERCÍCIOS



MDN Web Docs - Introdução a eventos -

https://developer.mozilla.org/pt-BR/docs/Aprender/JavaScript/Elementos_construtivos/Events