Eventos

DigitalHouse >
Coding School

Eventos

EM JAVASCRIPT JS

DigitalHouse > Coding School

O que vimos até hoje no módulo?

- Conceitos básicos de JS
- Estrutura DOM
- Manipulação de elemento e atributos
- Terminal (console)

E o que temos para hoje?

Interação do usuário através de eventos

DOM - Eventos

“Eventos são **ações** ou **ocorrências** que acontecem **no sistema**”

-MDN

DOM - Eventos

Alguns exemplos:

- A página terminou de carregar
- A página foi rolada
- Um botão foi clicado

O JavaScript nos permite agir quando esses eventos acontecem.

DOM - Eventos

Existem **duas formas** de registrar um evento.

A primeira é estabelecendo uma propriedade diretamente no elemento:

```
elemento.onNomeDoEvento = function () {}
```

DOM - Eventos

Os eventos mais usados são:

- onclick
- onchange
- onmouseover
- onmouseout
- onkeydown
- onload

Lista com Todos Eventos (w3schools)

DOM - Eventos

A segunda é utilizando **addEventListener**.

```
elemento.addEventListener(tipoDeEvento, funcaoGerenciadora);
```

- **tipoDeEvento**: é uma string com o nome do tipo de evento
- **funcaoGerenciadora**: é uma função invocada quando o evento acontece.

DOM - Eventos

Por exemplo:

```
elemento.addEventListener("click", function(){
    alert("Ai! Você clicou em mim!");
});
```

Eventos - this

Podemos utilizar a palavra reservada **this**, que neste contexto faz referência ao objeto que executou o evento (à quem o evento pertence).

```
function minhaFuncao() {
    // this é o elemento que executou o evento
    console.log(this)
}

elemento.addEventListener('click', minhaFuncao);
```

Eventos - removeEventListener

Para remover um *addEventListener* inserido, utilizamos:

```
elemento.removeEventListener(tipoDeEvento, funcaoGerenciadora)
```

- **tipoDeEvento**: é necessário que seja o mesmo evento do addEventListener
- **funcaoGerenciadora**: é necessário que seja a mesma função do addEventListener

Eventos - preventDefault()

Para evitar a execução de um evento, por padrão, utilizamos:

```
document.querySelector("#link").addEventListener("click",
function(event){
    event.preventDefault();
    // o link não vai mais para o Google
});

<a id="link" href="http://www.google.com.br/">Google</a>
```

Eventos - Mouse

O objeto **event** associado ao mouse tem atributos que permitem saber a posição dele com **clientX** e **clientY**.

```
elemento.addEventListener('click', function(event)
{
    event.clientX;
    event.clientY;
});
```

Eventos - Teclado

Também é possível controlar os eventos disparados quando as teclas são pressionadas, utilizando os eventos keypress, keydown e keyup.

```
elemento.addEventListener('keypress', function(event) {
        let  tecla = event.keyCode;
        if (tecla == 27) {
                alert("Você apertou escape!!");
        }
});
```

Timers

Timers - setTimeout

O JavaScript tem funções nativas que permitem atrasar a execução de qualquer código.

A função **setTimeout** é utilizada quando queremos que o código seja executado uma vez depois de um tempo estabelecido.

setTimeout(funcao, tempoDeEspera);

Exibe um alert depois de **3** segundos (3000 milissegundos):

setTimeout(function(){ alert("Hello"); }, **3000**);

Timers - setInterval

Por meio dessa função, podemos executar o mesmo código várias vezes em um intervalo regular

setInterval(função, tempoDeEspera);

A cada 3 segundos, aparece o alert:

setInterval(function(){ alert("Hello"); }, **3000**);

Timers - clearTimeout / clearInterval

Para interromper um *timeout*, utilizamos:

```
let delay = setTimeout(function(){ alert("Hello"); }, 3000);
clearTimeout(delay);
```

Para interromper um *interval*, utilizamos:

```
let intervalo = setInterval(function(){ alert("Hello"); }, 3000);
clearInterval(intervalo);
```

Até a próxima aula!

DigitalHouse > Coding School