AULA 3/4

JS: eventos

- Exemplos de eventos:
  - Página HTML terminou o carregamento
  - Um elemento HTML foi alterado
  - Um botão HTML foi clicado
  - Etc …

- Com o JS é possível interceptar estes eventos e aplicar alguma ação quando eles acontecerem.

```
<element event='some JavaScript'></element>
```

JS: eventos

```
<!DOCTYPE html>
<html>

<body>

    <button onclick="document.getElementById('demo').innerHTML=Date()">The
 time is?</button>

    <p id="demo"></p>

</body>

</html>
```

```
<!DOCTYPE html>
<html>

<body>

    <button onclick="this.innerHTML=Date()">The time is?</button>

</body>

</html>
```

JS: eventos

- Forma mais elegante...

```
<!DOCTYPE html>
<html>

<body>

    <p>Click the button to display the date.</p>

    <button onclick="displayDate()">The time is?</button>

    <script>
        function displayDate() {
            document.getElementById("demo").innerHTML = Date();
        }
    </script>

    <p id="demo"></p>

</body>

</html>
```

JS: eventos comuns

Evento	Descrição
onchange	Um elemento HTML mudou
onclick	O elemento HTML foi clicado
onmouseover	O mouse passou por cima do elemento HTML
onmouseout	O mouse saiu de cima do elemento HTML
onkeydown	O usuário apertou uma tecla
onload	O navegador terminou de carregar a página

JS: eventos comuns

A SMALL TASTE

- clicks
- drags
- drops
- hovers
- scrolls
- form submission
- key presses
- focus/blur

- mouse wheel
- double click
- copying
- pasting
- audio start
- screen resize
- printing

JS: eventos comuns

```
<body onload="myFunction()">
```

```
<img src="w3javascript.gif" onload="loadImage()"
 width="100" height="132">

<script>
function loadImage() {
  alert("Image is loaded");
}
</script>
```

```
<input type='text' id='num2' onchange="recalcular()”>
```

JS: addEventListener

```
my_element.addEventListener("click", function (e) {
    console.log(this.className); // logs the className of my_element
    console.log(e.currentTarget === this); // logs `true`
  });


  my_element.addEventListener("dblclick", (e) => {
    console.log(this.className); // WARNING: `this` is not `my_element`
    console.log(e.currentTarget === this); // logs `false`
  })
```

O addEventListener permite associar mais de uma função ao mesmo evento.
Por exemplo, fun1 e fun2, usando somente o onclick = fun1, nunca irá chamar as duas funções, só se colocássemos elas aninhadas.

Com o addEventListener, cada chamada ao método é um novo evento/função adicionado.

Como terceiro parâmetro, também é possível passar “options”.
Por exemplo, {once: true} vai permitir que a função execute somente 1x, apos isso, é removida da lista de funções a serem executadas no evento.

JS: BOAS PRÁTICAS

- Evite usar variáveis globais
  - Minimize o uso de variáveis globais.
  - Evite também objetos e funções globais.
  - Variáveis globais podem ser sobrescritas por outros trechos de código, levando ao caos.
- Use variáveis locais
  - Variáveis locais devem ser declaradas com var ou let, caso contrário, serão variáveis globais.
    - Var tem escopo de função
    - Let tem escopo de bloco

JS: BOAS PRÁTICAS

```
<script>
    var  x = 10;
    // Here x is 10
{
  var x = 2;
  // Here x is 2
}
// Here x is 2
</script>
```

```
<script>
    var  x = 10;
    // Here x is 10
{
        let x = 2;
  // Here x is 2
}
// Here x is 10
</script>
```

JS: BOAS PRÁTICAS

- Coloque todas as declarações no começo de cada trecho de código
  - Código limpo e claro
  - Local único para saber quais variáveis há
  - Fácil identificar o uso de variáveis globais
  - Reduz o risco de reescrever uma variável

JS: BOAS PRÁTICAS

```
// Declare no começo
var firstName, lastName, price, discount, fullPrice;

// Use depois
firstName = "John";
lastName = "Doe";

price = 19.90;
discount = 0.10;

fullPrice = price * 100 / discount;

// TAMBÉM VALE PARA LOOPS. Declare no começo.
var i;

// Use depois
for (i = 0; i < 5; i++) {}
```

JS: BOAS PRÁTICAS

- Inicialize as variáveis após a declaração
  - Provê código limpo
  - Evita valores undefined
  - Único lugar para consultar as inicializações

```
// Declare e inicialize no início
var firstName = "",
    lastName = "",
    price = 0,
    discount = 0,
    fullPrice = 0,
    myArray = [],
    myObject = {};
```

JS: BOAS PRÁTICAS

- Não declare tipos básicos com tipos complexos
  - Exemplo: não use new String(“”), isso irá criar um Object
- Use {} ao invés de new Object()
- Use "" ao invés de new String()
- Use 0 ao invés de new Number()
- Use false ao invés de new Boolean()
- Use [] ao invés de new Array()
- Use function (){} ao invés de new Function()

```
var x1 = {};           // new object
var x2 = "";           // new primitive string
var x3 = 0;            // new primitive number
var x4 = false;        // new primitive boolean
var x5 = [];           // new array object
var x6 = /()/;         // new regexp object
var x7 = function () { }; // new function object
```

JS: BOAS PRÁTICAS

- Cuidado com as conversões de String e Number

```
var x = 5 + 7;       // x.valueOf() is 12,  typeof x is a number
var x = 5 + "7";     // x.valueOf() is 57,  typeof x is a string
var x = "5" + 7;     // x.valueOf() is 57,  typeof x is a string
var x = 5 - 7;       // x.valueOf() is -2,  typeof x is a number
var x = 5 - "7";     // x.valueOf() is -2,  typeof x is a number
var x = "5" - 7;     // x.valueOf() is -2,  typeof x is a number
var x = 5 - "x";     // x.valueOf() is NaN, typeof x is a number
```

JS: BOAS PRÁTICAS

- Use === ao invés de ==
  - O operador == converte automaticamente os valores para o mesmo tipo antes de verificar
  - O operador === força que os tipos sejam iguais

```
0 == "";        // true
1 == "1";       // true
1 == true;      // true

0 === "";       // false
1 === "1";      // false
1 === true;     // false
```

JS: BOAS PRÁTICAS

- Use valores default para suas variáveis, um valor undefined pode quebrar o seu programa

```
function myFunction(x, y) {
    if (y === undefined) {
        y = 0;
    }
}

// OU

function (a = 1, b = 1) { /*function code*/ }
```

JS: BOAS PRÁTICAS

- Sempre termine os Switches com o valor default

```
switch (new Date().getDay()) {
    case 0:
        day = "Sunday";
        break;
    case 1:
        day = "Monday";
        break;
    case 2:
        day = "Tuesday";
        break;
    case 3:
        day = "Wednesday";
        break;
    case 4:
        day = "Thursday";
        break;
    case 5:
        day = "Friday";
        break;
    case 6:
        day = "Saturday";
        break;
    default:
        day = "Unknown";
}
```

JS: BOAS PRÁTICAS

- Evite usar a função eval()
  - Esta função serve para executar código texto como código JS, pode levar a diversos problemas de segurança...
- Reduza a quantidade de comandos dentro dos loops

```
var i;
for (i = 0; i < arr.length; i++) { }

//Melhor usar assim:
var i;
var l = arr.length;
for (i = 0; i < l; i++) {?
```

JS: Exercício

- Criar uma página HTML com somente 1 botão.
- Ao clicar no botão, deve-se alterar o background-color usando uma cor RGB aleatória.
- Para gerar números aleatórios no intervalo de 0-255
  - Math.floor(Math.random() * 255); //Math.random() retorna número entre 0 e 1, ao multiplicar por 255 temos o range desejado.

JS: Exercício

- Criar uma página Pokémon! A cada clique na página (em qualquer lugar) deve ser inserido um novo Pokémon.
- No exemplo abaixo, cliquei na página 9 vezes.


#1


#2


#3


#4


#5


#6


#7


#8


#9