Introdução ao JavaScript – parte 2

Profª Mª Denilce Veloso
denilce.veloso@fatec.sp.gov.br
denilce@gmail.com

JavaScript – Outras formas de Mostrar dados na tela

É possível mostrar/imprimir dados nas seguintes formas:

✓Usando caixa de alerta **alert** ou window.alert() (já visto);

✓Escrevendo dentro de uma saída (na tela) usando **document.write()**; (funcional mas ele recarrega a página-ideal innerHTML)

✓Escrevendo dentro de um elemento (na tela) usando **innerHTML**;

✓Escrevendo dentro de um console do navegador usando **console.log()**.

JavaScript – Outras formas de Mostrar dados na tela

Exemplo1:

```
<script>
    document.write("Utilizando document.write!!!");
</script>
```

Exemplo2:

```
<body>
  <p id="demo"></p>
  <script>
    document.getElementById("demo").innerHTML = 5 + 6;
  </script>
</body>
```

** ver UsandoFormasMostrarTela, 2, 3.html

JavaScript – Outras formas de Mostrar dados na tela

Exemplo3:

```
let s = "teste usando console!!!";
    /** apertar f12 para ver o resultado no console **/

    try {
        console.log(s);
    } catch (e) {
        alert(e.message);
    }
```

** ver UsandoFormasMostrarTela, 2, 3.html

DDAOV6
DDAOV17

avaScript – TIPO DATE

O método Date.parse() aceita uma string como argumento, representando uma data. Formatos válidos:

–mês/dia/ano (*6/13/2004*)

–nome_do_mês dia, ano (*January 12, 2004*)

–dia_da_semana nome_do_mês dia ano horas:minutos:segundos zona_de_tempo (*Tue May 25 2004 00:00:00 GMT-0700*)

–Formato ISO 8601 extendido YYYY-MM-DDTHH:mm:ss.sssZ (*2004-05-25T00:00:00*)

Exemplo:

```
let  novaData = new Date(Date.parse("May 25, 2016"));
let NovaData = Date.parse("May 25, 2016");
// ou
let  novaData = new Date("May 25, 2016");
let novaData = new  Date(""2016-05-25“);
let novaData = new Date(“05/25/2016”);
```

Slide 5

DDAOV6 GMT significa "Greenwich Mean Time" e é uma referência para a hora do dia baseada no tempo solar médio no Meridiano de Greenwich, em Londres, Inglaterra. É um sistema de tempo que foi estabelecido como padrão internacional em 1884.
DENILCE DE ALMEIDA OLIVEIRA VELOSO; 16/04/2023

DDAOV17 O termo "GMT-0700" representa um fuso horário que está 7 horas atrás do Tempo Médio de Greenwich (GMT).
DENILCE DE ALMEIDA OLIVEIRA VELOSO; 06/04/2025

JavaScript – TIPO DATE

- Obter data e hora

Devolver data e hora no formato:
Dia Semana, Nome Mês, Dia, Mês, Ano, Hora:Minuto:Segundo

Exemplo:

```
let data= new Date();

alert(data); → Mon Mar 29 2021 12:26:48 GMT-0300 (Horário Padrão de Brasília)
alert(data.toDateString("dd/mm/yy")); → Mon Mar 29 2021
alert(data.toLocaleDateString('pt-BR'));
→02/04/2025
```

JavaScript – Métodos Date

método get:

getDate() - Obtém o dia do mês (numérico de 1 a 31)
getDay() - Obtém o dia da semana (0 a 6)
getMonth() - Obtém o mês (numérico de 0 a 11)
getYear() - Obtém o ano
getHours() - Obtém a hora (numérico de 0 a 23)
getMinutes() - Obtém os minutos (numérico de 0 a 59)
getSeconds() - Obtém os segundos (numérico de 0 a 59)

Exemplo:

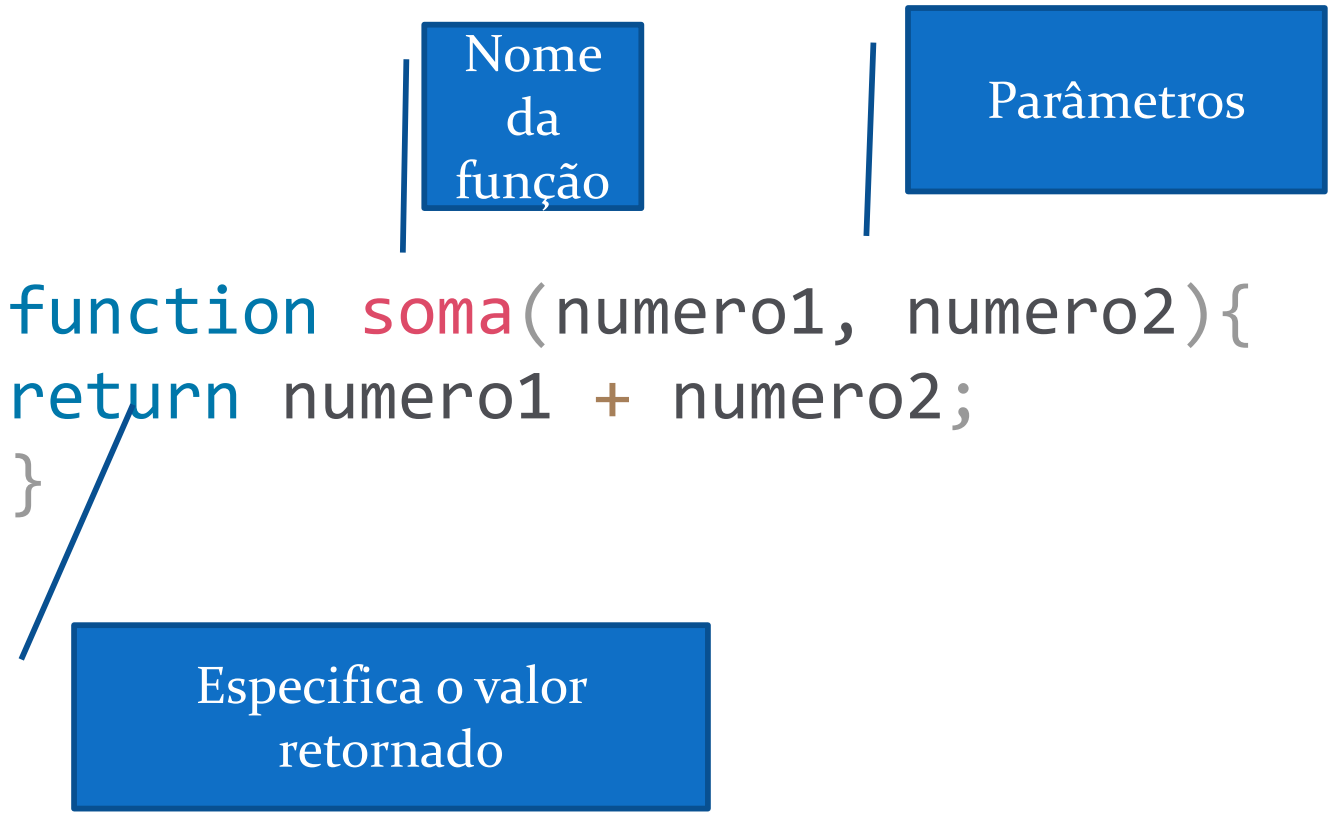
```
let dataCompleta = new Date();
let diaSemana = dataCompleta.getDay();

alert(dataCompleta);
alert(diaSemana);
```

Thu Mar 10 2016 15:09:49 GMT-0300 (Hora oficial do Brasil)

4 → quinta

JavaScript – Função

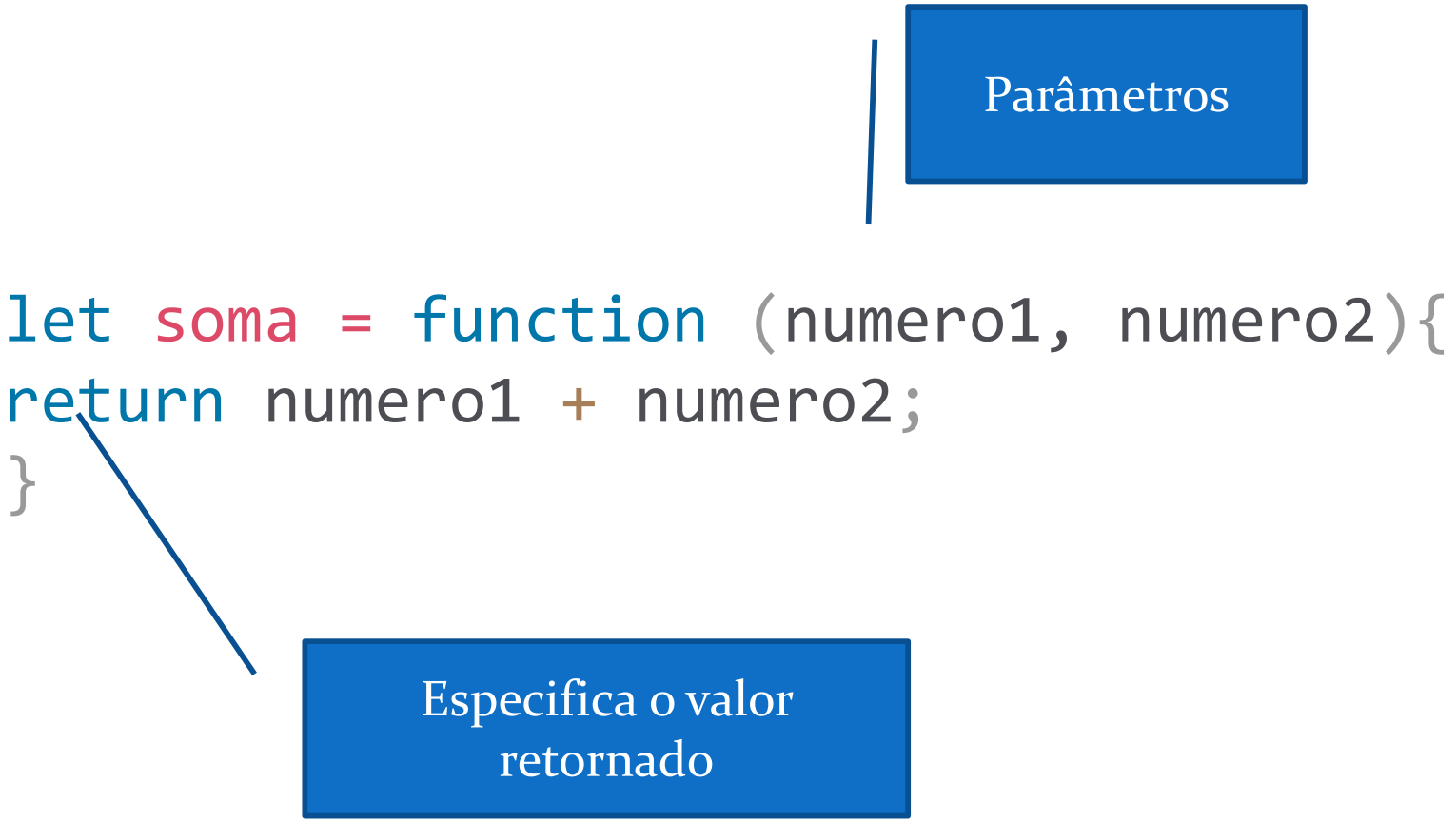
Uma função é um conjunto de instruções que executa uma tarefa ou calcula um valor e retorna um valor.



** Existem as funções intrínsecas: exemplos: parseFloat, eval, etc.

JavaScript – Expressão de Função

Funções também podem ser criadas por uma expressão de função, não precisa ter um nome.



** não é possível fazer sobrecarga de função → assinaturas diferentes

DV3

A finalidade de uma função anônima é exatamente a de permitir passá-la como se fosse um objeto qualquer, que você pode atribuir a uma variável, independentemente de haver um nome para a função.

Protegendo variáveis usando uma função anônima

Proteger variáveis contra mal uso, é uma das finalidades que acabou se encontrando para funções anônimas. Seria o equivalente a criar membros privados, como é possível em várias linguagens.

No exemplo do fibonacci, se você quiser proteger a variável usada para atribuir a função, poderia fazer assim:

```
var fibonacci = (function() {
   var fnc = function(num)
   {
      if(num==1 || num==2)
         return 1;
      else
         return fnc(num-1) + fnc(num-2);
   };
   return fnc;
})();
```

Dessa forma, não teria como alterar a dependência interna da função depois desta já ter sido criada. Não será mais possível alterar a variável fnc, pois ela está dentro do contexto da função anônima, cuja referência se perde, logo após chamar a mesma.

Estrutura básica:

```
var obj = (function() {
   // declarações a serem protegidas
   var a, b, c;
   // retornando um objeto construído a partir de a, b e c
   return obj;
})();
```

DENILCE VELOSO; 10/04/2019

JavaScript – Função – Arrow functions

Arrow functions foram introduzidas no ES6. É basicamente uma forma mais curta de definir function expressions e ajuda a tornar o código mais fácil de ler, principalmente nas expressões curtas..

```
let ispar=(n) => {
        if (n%2==0)
            return "par";
        else
            return "impar";
    }

// uma função pode retornar um alert
assim: return alert(“par”), para chamar
let x=ispar(4);
```

DV3

A finalidade de uma função anônima é exatamente a de permitir passá-la como se fosse um objeto qualquer, que você pode atribuir a uma variável, independentemente de haver um nome para a função.

Protegendo variáveis usando uma função anônima

Proteger variáveis contra mal uso, é uma das finalidades que acabou se encontrando para funções anônimas. Seria o equivalente a criar membros privados, como é possível em várias linguagens.

No exemplo do fibonacci, se você quiser proteger a variável usada para atribuir a função, poderia fazer assim:

```
let fibonacci = (function() {
   let fnc = function(num)
   {
      if(num==1 || num==2)
         return 1;
      else
         return fnc(num-1) + fnc(num-2);
   };
   return fnc;
})();
```

Dessa forma, não teria como alterar a dependência interna da função depois desta já ter sido criada. Não será mais possível alterar a variável fnc, pois ela está dentro do contexto da função anônima, cuja referência se perde, logo após chamar a mesma.

Estrutura básica:

```
let obj = (function() {
   // declarações a serem protegidas
   let a, b, c;
   // retornando um objeto construído a partir de a, b e c
   return obj;
})();
```

DENILCE VELOSO; 10/04/2019

JavaScript – Argumentos da Função

Não é necessário passar tipo de dados nos argumentos e também é possível passar quantos argumentos desejados sem declarar na função. Os argumentos são representados internamente como um array.

Exemplo:

```
<script>
    function alo() {
            return "Nº Argumentos:" + arguments.length + "\n" +
arguments[0] + " " + arguments[1];
}

let teste = alo("Prof", "Denilce");

alert(teste);   → Nº Argumentos:2
</script>          Prof Denilce
```

JavaScript – Exemplo:UsandoFuncaoFatorial.html (1)

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Testando IF</title>

    <script>
        function fatorial(n) {
            if ((n == 0) || (n == 1))
                return 1;
            else
                return (n * fatorial(n - 1));
        }
```

JavaScript – Exemplo:UsandoFuncaoFatorial.html (2)

```
        alert("primeiro caso=" + fatorial(4));   →24

        let fator = function (n) {
            if ((n == 0) || (n == 1))
                return 1;
            else
                return (n * fator(n - 1));
        }


        alert("segundo caso=" + fator(4)); →24
    </script>
</head>

<body>
</body>
</html>
```

JavaScript – Exemplo: Função Maior

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
        <meta charset="UTF-8">
        <title>teste</title>
</head>
<body>

        <script>
                function maior(a,b) {
                        return (a>b)?a:b;
                }
                alert(maior(20,5));
        </script>
</body>
</html>
```

JavaScript – Exemplo: UsandoFuncaoAninhada.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Funcao Aninhada</title>
    <script>
        function adicionadobro(a, b) {
            function dobro(x) {
                return x + x;
            }
            return dobro(a) + dobro(b);
        }
        a = adicionadobro(2, 3); // retorna  4+6=10
        alert(a);
        b = adicionadobro(3, 4); // retorna 6+8=14
        alert(b);
        c = adicionadobro(4, 5); // retorna 8+10=18
        alert(c);
    </script>
</head>
<body> </body> </html>
```

JavaScript – Resumo Tipos de função

Funções do próprio JavaScript

Exemplos: ParseInt(), ParseFloat(), Math.Round()
etc

Declaradas

```
function isDivisibleBy5(number) {
  if (number % 5 === 0) {
    return true;
  } else {
    return false;
  }
}
```

JavaScript – Resumo Tipos de função

Expressões de Funções

```
let isDivisibleBy5 = function(number) {
  if (number % 5 === 0) {
    return true;
  } else {
    return false;
  }
};
```

JavaScript – Resumo Tipos de função

Arrows

```
let isDivisibleBy5 = (number) => {
  if (number % 5 === 0) {
    return true;
  } else {
    return false;
  }
};
```

- Apenas funções declaradas podem ser usadas antes de serem declaradas no código (Hoisting), expressão de função e funções arrows não.
- Na expressão de função a recursividade é realizada com o nome da variável.
- Função arrow é muito usada em call back (uma função que é passada como argumento para outra função)

Criar **QUATRO** funções, usar arquivo externo para o arquivo js.

1) Para receber três números e retornar o maior deles.
2) Para receber três números e retorná-los em ordem crescente.
3) Para receber uma string e retornar se ela é palíndromo ou não. (converter para maiúsculas)
4) Para receber 3 valores, informar se formam um triângulo e qual é o tipo do triângulo.

Executa um bloco de código um número de vezes.

```
for (i = 0; i < 5; i++) {
    text += “o número é" + i + "<br>";
}
```

Como faria para escrever um laço para contar de 100 até 0, de 5 em 5. Não exibindo o zero?

A instrução *do ... while* executa enquanto um condição for verdade. A sintaxe: do instruções while (condição)

Exemplo1:

```
let numero = 1;
do {
  document.write (numero+" ");
  numero++;
}
while (numero<=10)
```

→ imprime de 1 até 10

Exemplo2:

```
<p id="teste"></p>
<script>
    let i = 0;
    do {

document.getElementById("teste").innerHT
ML += i + "<br>";
        i++;
     }
    while (i<=10) → imprime de 0 até 10
</script>
```

JavaScript – Instrução while

A instrução *while* executa enquanto uma condição for verdadeira. A sintaxe:
while (condição){ instruções }

Exemplo:

```
let i = 0;
while (i < 100) {
i += 100;
}
```

Instrução break

Serve para interromper instruções: *switch*, *for*, *do* ... *while* ou *while*.

Instrução continue

Obriga a uma nova iteração (loop) do ciclo. As instruções que estiverem depois do *continue* serão ignoradas na interação atual.

```
for (i = 0; i < 10; i++) {
    if (i === 3) {
        break;
      }
    alert("Primeiro: O número
é igual a:" + i);
 //→ imprime até o 2
}
```

```
for (j = 0; j < 10; j++) {
      if (j === 5) {
        continue;
       }
      alert("Segundo: O número
é igual a:" + j);
//→ vai pular o 5
}
```

JavaScript – Instrução SWITCH

A instrução *switch* usa-se para executar um código na base duma avaliação entre mais de duas alternativas. A sintaxe:

```
switch (expressão) {
  case valor1:
//Instruções executadas quando o resultado da expressão for igual valor1
    [break;]
  case valor2:
//Instruções executadas quando o resultado da expressão for igual  valor2
    [break;]
  ...
  case valueN:
 //Instruções executadas quando o resultado da expressão for igual valorN
    [break;]
  default:
  //Instruções executadas quando o valor da expressão é diferente de todos os anteriores
    [break;]
}
```

JavaScript – Instrução switch Exemplo:

```
let diaDaSemana = prompt("Digite o dia da Semana");
        switch (diaDaSemana) {
        case "Domingo":
                alert("Dia 1");
                break;
        case "Segunda":
                alert("Dia 2");
                break;
        case "Terça":
                alert("Dia 3");
                break;
        case "Quarta":
                alert("Dia 4");
                break;
```

JavaScript – Instrução switch Exemplo:

```
case "Quinta":
        alert("Dia 5");
        break;

case "Sexta":
        alert("Dia 6");
        break;

case "Sábado":
        alert("Dia 7");
        break;
default:
        alert("Dia desconhecido");
}
```

JavaScript – Instrução with

A instrução *with* **atribui o escopo do código com um objeto** em particular. A sintaxe:

with (expressão) instrução;

Exemplo1:

```
let a, x, y;
let r = 10;
with (Math) {
  a = PI * r * r;
  x = r * cos(PI);
  y = r * sin(PI / 2);
}
```

Exemplo2:

```
let obj = { a : 10 }
 with(obj) {
  alert(a) // 10
}
```

JavaScript – Instrução with - Exemplos

```
let ano = parseInt(prompt("Qual seu ano de nascimento?"));
let teste = true // os anos variam de acordo com a fonte
        switch (teste) {
             case ano <= 1945:
                 console.log('Geração silenciosa');
                 break;
             case ano > 1945 && ano <= 1964:
                 console.log('Baby Boomers');
                 break;
             case ano > 1964 && ano <= 1980:
                 console.log('Geração X');
                 break;
             case ano > 1980 && ano <= 1996:
                 console.log('Millennials ou Geração Y');
                 break;
             case ano > 1996 && ano <= 2009:
                 console.log('Geração Z');
                 break;
             default:
                 console.log('Geração Alfa');
         }
```

Um array pode ser criado com a definição de seu tamanho inicial ou não:

let notasMusicais = new Array();

OU let meuArray = [];

//os dois tem comprimento(propriedade length)=0 e são tipo (typeOf) ***OBJECT***

let notasMusicais = new Array(7);

//comprimento 7 e todos com valores undefined

let notasMusicais = [7];

//comprimento 1 e posição 0=7

→ let notasMusicais = (7); // 7 (number)

✓armazenam listas de dados;

let meusCarros = new Array("Gol", "Uno", "Celta"); *OU let meusCarros = ["Gol", "Uno", "Celta"];*

let a = new Array(8); //comprimento 8, undefined

let b = new Array(8, 9); //comprimento 2, valores 8 e 9 nas 1ª e 2ª posições

let c = [8]; //comprimento 1 e dado 8

✓armazenam diferentes tipos de dados ao mesmo tempo (número, string,booleano,objeto);

```
let meuArray = [7,14];
let myObject={ra:123};
let bol=true;
let myArray1=[1,bol,"Fatec",myObject,meuArray];
alert(myArray1); →
```

1,true,Fatec,[object Object],7,14

→**Posição de cada dado é fixa**

Observação:

```
let notasMusicais = new Array(7);
   notasMusicais[0] = "do";
   notasMusicais[1] = "ré";
   notasMusicais[2] = "mi";
   notasMusicais[3] = "fá";
   notasMusicais[4] = "sol";
   notasMusicais[5] = "lá";
   notasMusicais[6] = "si";
   alert(notasMusicais); → do,ré,mi,fá,sol,lá,si
```

```
let notasMusicais1 = ["dó","ré","mi","fá","sol","lá","si"];
notasMusicais1[7] = "xx";    → PERMITIDO
alert(notasMusicais1);  → do,ré,mi,fá,sol,lá,si,xx
```

EXERCICIO

•Suponha o seguinte vetor V, qual será o resultado do mesmo após a execução do algoritmo?

	1	2	3	4	5	6	7	8
V	5	1	4	2	7	8	3	6

```
Para i de 8 até 5 passo -1 Faça
    aux ← v [i]
    v [i] ← v [8 - i + 1]
    v [8 - i + 1] ← aux
Fim_Para

v [3] ← v [1]
v [v [3]] ← v [v [2]]
```

DDAOV5 6 3 6 7 2 6 1 5

DENILCE DE ALMEIDA OLIVEIRA VELOSO; 30/04/2022

JavaScript – ARRAYS - Exemplos:

```
//declarando  com os dados - 3 strings
let nomes = ["Clarice Lispector", "Carlos Drumond", "José de Alencar"];

alert("nomes: " + nomes + " comprimento:" + nomes.length);
```

→nomes:Clarice Lispector,Carlos Drumond,José de Alencar: Comprimento:3

```
//omitindo o operador New, literal
let idades = Array(4);
idades[0] = 10;
idades[1] = 20;
idades[2] = 30;
idades[3] = 40;

alert("idades:" + idades + " comprimento:" + idades.length);
```

→ idades:10,20,30,40:Comprimento:4

→ atribuição dos valores

JavaScript – ARRAYS - Exemplos:

```
//omitindo o operador com dados, notacao literal
let cidades = Array("Sorocaba");
alert("cidades:" + cidades); → cidades:Sorocaba


//tipos de dados diferentes
let misturados = [34, "doce", "azul", 11];
alert("misturados:" + misturados); →
misturados:34,doce,azul,11

// array vazio
let nada = [];
alert("nada:" + nada); → nada  (Comprimento seria 0)
```

A propriedade length não é somente de leitura, mas pode ser utilizada para remover(altera comprimento) ou adicionar itens:

Exemplo1:


```
let cores1 = ["vermelho", "azul", "verde"];
cores1.length = 2;
//excluiu
alert(cores1[2]); //undefined
```

** se colocar cores1.length = 1 {só vai ficar com o vermelho}

Exemplo2:


```
let cores2 = ["vermelho", "azul", "verde"];
cores2.length = 4;
//incluiu
alert(cores2[3]); //undefined
```

Array Bidimensional

```
//com dados
let  meuArray = [[1, 2], [3, 4], [5, 6]]; → 3 linhas e 2 colunas

let novoArray = [[1,2,0],[3,4,0],[5,6,0]] → 3 linhas e 3 colunas

//sem dados
let meuArray = new Array(3); → 3 linhas e 2 colunas
for(let i = 0; i < meuArray.length; i++) {
  meuArray[i] = new Array(2);
}
meuArray[0][0] = 1;
meuArray[0][1] = 2;
meuArray[1][0] = 3;
meuArray[1][1] = 4;
meuArray[2][0] = 5;
meuArray[2][1] = 6;
```

O método **reverse**() inverte a ordem dos itens:

```
let numeros = [1, 2, 3, 4, 5, 6];
numeros.reverse();
alert(numeros); //6,5,4,3,2,1
```

O método **sort**() coloca os itens em ordem ascendente (converte para string).

```
let valores = [0, 1, 5, 10, 15, 2];
valores.sort();
alert(valores); //0,1,10,15, 2, 5 – segue Unicode
```

http://www.unicodetables.com/

//para fazer o Sort em ordem ascendente (valor)

```
let points = [40, 100, 1, 5, 25, 10];
 points.sort(function(a, b){return a-b});
 alert(points); // 1,5,10,25,40,100
```

Tipos de retorno:

- se a comparação for menor que zero, a é posicionado antes de b
- se a comparação for maior que zero, a é posicionado depois de b
- se a comparação for igual a zero, a e b permanecem com as posições inalteradas

//para fazer o Sort em ordem descendente(valor)

```
let points = [40, 100, 1, 5, 25, 10];
 points.sort(function(a, b){return b-a});
 alert(points); // 100,40,25,10,5,1
```

Reduce

O reduce itera sobre cada elemento do array.
A cada iteração, ele compara o valor atual (representado por b) com o valor acumulado até aquele momento (representado por a).

```
function maior(um, dois, tres) {
    numeros = [um, dois, tres]
    result = numeros.reduce((a, b) => {
        return a > b ? a : b
    })
    return result
}
```

DDAOV13

Passo a passo:
Criação do array: numeros = [1, 2, 3].
Primeira iteração:
a (valor inicial) é o primeiro elemento do array: 1.
b é o segundo elemento: 2.
A comparação a > b é falsa (1 não é maior que 2).
O reduce retorna b, que é 2.
a agora é 2 para a próxima iteração.
Segunda iteração:
a é 2 (resultado da iteração anterior).
b é 3 (terceiro elemento do array).
A comparação a > b é falsa (2 não é maior que 3).
O reduce retorna b, que é 3.
a agora é 3 para a próxima iteração.
Terceira iteração:
Como só há três elementos, o reduce termina.

DENILCE DE ALMEIDA OLIVEIRA VELOSO; 06/08/2024

O método POP remove o último elemento do array e retorna aquele elemento.

O método PUSH adiciona um (ou mais) elemento(s) ao final de um array.

```
function incrementaFinal(_array)  {
    let newArray = [];
    newArray = _array.slice(0);  //extrai do início até o fim
    let lastNumber = newArray.pop(); //exclui o último -> 5
    newArray.push(lastNumber + 1); //5+1 → incluindo o 6
    return newArray;
}
let x= Array(1,2,3,4,5)
 alert(incrementaFinal(x)); → retorna 1,2,3,4,6
```

No exemplo do slide anterior se:

let x= Array(6,4,3,2,1)

alert(incrementaFinal(x));

→ retorna o quê????

O método POP remove apenas um elemento, se quiser excluir mais do que um, usar método **SPLICE**.

```
let arr = [1, 2, 3, 4, 5];
arr.splice(2, 2); // Remove 2 elementos a partir da posição 2 (3 e 4)
console.log(arr); // [1, 2, 5]
```

É possível usar o método **splice** para excluir de um array e montar outro.

```
let meses = ["Janeiro", "Fevereiro", "Segunda", "Terça"];
let dias = meses.splice(2, 1);

console.log(dias); // ["Segunda"]
console.log(meses);  //  ["Janeiro",  "Fevereiro", "Terça"]
```

Qual é o resultado da variável total?

```
let alunos=
 ["Viviane","André","Helio","Denise","Junior","Leonardo",
"Jose","Nelma", “Pedro"];
        let total = 0;
        alunos.pop();

        for (i = 0; i <= alunos.length - 1; i++) {
           total = total + alunos[i].length;
           }

        document.write("O total é: "+total);
```

O método foreach() **executa uma função callback para cada elemento do array.** Uma função callback corresponde a uma chamada de retorno, ou seja, ela retorna o processamento de um comando ou de um conjunto de comandos para que esse conteúdo seja utilizado no ponto de chamada.

```
// exemplo
 let arrayExemplo = [1, 2, 3, 4];

      function NumeroPar(num) {
         if (num % 2 == 0)
            alert(num);
      }

      arrayExemplo.forEach(NumeroPar); → imprime 2 4
```

```
//ou
 let arrayExemplo2 = [1, 2, 3, 4];
 arrayExemplo2.forEach(function NumeroPar(num) {
                if (num % 2 == 0)
                    alert(num);
            }); → imprime 2 4


//ou

 let arrayExemplo3 = [1, 2, 3, 4];
 arrayExemplo3.forEach(num => {
                if (num % 2 == 0)
                    alert(num);
            }); → imprime 2 4
```

JavaScript – FUNCTION COM PARÂMETRO ARRAYS

```
function soma() {
    let soma = 0;
    for (var i = 0; i < arguments.length; i++) {
        soma += arguments[i];
    }
    return soma;
}
//O método apply()
// permite que você chame uma função
//com um determinado valor e argumentos
fornecidos como um array.
    let arr1 = [1, 2,3, 4];
    alert(soma.apply(null, arr1));  → 10
```

DDAOV14
DDAOV15
DDAOV16

JavaScript – FUNCTION COM PARÂMETRO ARRAYS

```
function soma() {
    let soma = this;
    for (var i = 0; i < arguments.length; i++) {
        soma += arguments[i];
    }
    return soma;
}
//enviando um parâmetro no lugar do nuyll
let arr1 = [1, 2,3, 4];
alert(soma.apply(10, arr1));  → 20
```

Slide 52

DDAOV14 seu segundo parâmetro recebe um Array dos parâmetros da função, enquanto o primeiro parâmetro continua recebendo o valor que será atribuído ao this.
DENILCE DE ALMEIDA OLIVEIRA VELOSO; 16/10/2024

DDAOV15 Outro exemplo
DENILCE DE ALMEIDA OLIVEIRA VELOSO; 16/10/2024

DDAOV16 function sumNumbers(firstNumber, secondNumber) { const sum = this + firstNumber + secondNumber; console.log(sum) } sumNumbers.apply(5, [2, 7]) // 14
DENILCE DE ALMEIDA OLIVEIRA VELOSO; 16/10/2024

Passando um array na função Max (ou outra função que receba vários dados)

```
let arr1 = [1, 2,3, 4];
alert(Math.max.apply(null, arr1));
```

JavaScript - Objetos

Os objetos em JavaScript podem ser agrupados em três categorias:

- ✓objetos internos da linguagem (tipos já existentes na linguagem), como: strings, arrays, datas e etc;
- ✓objetos do navegador como: window e document;
- ✓objetos personalizados/criados pelo desenvolvedor.

JavaScript – Criando novos Objetos

Um objeto é uma a entidade independente, com propriedades (atributos que define suas características) e métodos (comportamentos).
Uma propriedade de um objeto pode ser entendida como uma "variável" ligada ao objeto acessada pela sintaxe:

nomeDoObjeto.nomeDaPropriedade

JavaScript – Criando Objetos

```
let meuCarro = new Object();  // usando construtor
meuCarro.make = "Ford";  //cria propriedade e atribui valor
meuCarro.modelo = "Mustang";
meuCarro.ano = 1969;

//Ou

let meuCarro = {}; //o mesmo que new Object()
meuCarro.make = "Ford"; //cria propriedade e atribui valor
meuCarro.modelo = "Mustang";
meuCarro.ano = 1969;
```

** **O tipo Object é a base para todos os outros objetos**

JavaScript – Criando Objetos

Criando o objeto de forma literal:

```
let aluno = {
name : "Manoel", //cria propriedade e atribui valor
ra: 1234,
turma:”A”
};
```

JavaScript – Criando Objetos - Propriedades

```
let meuObj = new Object(),
        str = "minhaString",
        aleat = Math.random(),
        obj = new Object();

    meuObj.tipo = "Sintaxe de ponto";
    meuObj["data de criacao"] = "String com espaco";
    meuObj[str] = "valor de String";
    meuObj[aleat] = "Numero Aleatorio";
    meuObj[obj] = "Objeto";
    meuObj[""] = "Mesmo uma string vazia";

    alert("tipo=" + meuObj.tipo + "str="+meuObj[str]);
                alert("aleat=" + meuObj[aleat]);
                alert("data de criação=" + meuObj["data de criacao"]);
                alert("obj=" + meuObj[obj]);
```

** ver UsandoObjetos1.html

** propriedades com *identificadores inválidos (no nome ou no tipo)*, são acessadas através de colchetes [ ]

JavaScript – Criando Objetos – Propriedades – dot notation

```
let objeto = {};
let nome_propriedade = 'nome';
objeto[nome_propriedade] = 'Peter';
alert(objeto.nome);

let objeto2 = {};
objeto2["nome"] = "Meg";
alert(objeto2.nome);
```

JavaScript – Criando Objetos

O método **Object.create()** cria um novo objeto, utilizando um outro objeto existente como protótipo para o novo objeto a ser criado.

```
const objEmpresa = { empresa: 'ZF do Brasil' };
const funcionario = Object.create(objEmpresa, { nome: {
value : 'José Carlos' }, endereco : { value : 'Av. São Paulo'} });
console.log(funcionario.empresa); // ZF do Brasil
console.log(funcionario.nome); // José Carlos
console.log(funcionario.endereco); // Av. São Paulo
```

***Alterações no objeto protótipo serão refletidas no objeto recém-criado.*

JavaScript – Criando Objetos

O método **Object.assign()** é usado para copiar os valores de todas as propriedades de um ou mais objetos origem para um objeto de destino.

```
const objEmpresa = { empresa: 'ZF do Brasil' };
const objVeiculo = { placa : 'XXX 1234'};

const motorista = Object.assign({}, objEmpresa,
objVeiculo);
console.log(motorista);// empresa: 'ZF do Brasil', placa:
'XXX 1234
```

JavaScript – EXERCÍCIO PARA ENTREGA

DISPONIBILIZAR no GITHUB: seuusuario/PWEB/ATIVIDADE10

Criar uma aplicação para calcular e mostrar o IMC de uma pessoa.

Dados de entrada:

- Altura
- Peso Atual da pessoa

Calcular e retornar o IMC da pessoa, e uma mensagem conforme abaixo:

IMC	CLASSIFICAÇÃO	OBESIDADE (GRAU)
MENOR QUE 18,5	MAGREZA	0
ENTRE 18,5 E 24,9	NORMAL	0
ENTRE 25,0 E 29,9	SOBREPESO	I
ENTRE 30,0 E 39,9	OBESIDADE	II
MAIOR QUE 40,0	OBESIDADE GRAVE	III

Atenção: Utilize funções

Exercício para Entrega

Criar o Objeto:
Funcionario01(propriedades: Matrícula, Nome, Função). Atribua valores para as propriedades.

** Utilizar 3 formas diferentes

Disponibilizar como Atividade11 no GITHUB.

→ Seunome/PWEB/Atividade11

JavaScript – Criando Objetos - JSON

JSON (*JavaScript Object Notation*) ou notação de objeto JavaScript é um formato simples e popular para armazenar e transferir dados aninhados ou hierárquicos. Ele é tão popular que muitas linguagens de programação tem bibliotecas capazes de analisar e gravar JSON (como a biblioteca JSON do Python). As requisições GET e POST da internet, geralmente, transmitem dados no formato JSON. O JSON permite que objetos (ou dados de outros tipos) sejam facilmente encapsulados em outros objetos.

Com uma mistura de chaves de abertura aninhadas, colchetes e vírgulas, é fácil cometer erros com o JSON. Se o JSON estiver sendo gerado manualmente, poderá ser analisado com um linter JSON como **jsonlint.com** para localizar erros de sintaxe com rapidez e facilidade. Um linter é um software que analisa código para verificar a existência de erros de sintaxe.

JavaScript – Criando Objetos - JSON

Elementos básicos do JSON.

{ e } - delimita um objeto.
[e] - delimita um array.
: - separa chaves (atributos) de valores.
, - separa os atributos chave/valor.

JavaScript – Criando Objetos - JSON

```
let texto = '{"nome": "Maria", "rg": 123}';
        // convertendo para objeto
        let objeto = JSON.parse(texto);

       // listando cada atributo
        alert(objeto.nome+'-'+objeto.rg);
```


```
       // convertendo o objeto para string
       alert(JSON.stringify(objeto));
```

JavaScript – Criando Objetos - JSON

Supondo Json:

```
let txt = '{"funcionarios":[' +
'{"nome":“Pedro","sobrenome":“Antunes" },' +
'{"nome":“José","sobrenome":“Egea" },' +
'{"nome":“Maria","sobrenome":“Santos" }]}’;
```

Para pegar os dados:

```
let obj = eval ("(" + txt + ")"); // eval avalia expressões

alert(obj.funcionarios[1].nome);
alert(obj.funcionarios[1].sobrenome);
```

*** ver exemplo UsandoJSON.html*

JavaScript – Criando Objetos - Propriedades

→ Usando o for para imprimir as propriedades

```
let languages = {
    english: "Hello!",
    french: "Bonjour!",
    notALanguage: 4,
    spanish: "Hola!"
};

for(var myVariable in languages) {
  if (typeof languages[myVariable] == "string")
  console.log(languages[myVariable]);
}
```

JavaScript – Criando Objetos - Desestruturação

Desestruturação é um recurso que permite extrair os valores de um array ou objeto em variáveis individuais.

```
const usuario = { nome: "João", idade: 30, cidade: "São Paulo" };
const { nome, idade } = usuario;
console.log(nome); // Imprime "João"
console.log(idade); // Imprime 30

const frutas = ['maçã', 'banana', 'laranja'];
const [fruta1, fruta2, fruta3] = frutas;
console.log(fruta1); // Imprime : maçã
console.log(fruta2); // Imprime : banana
console.log(fruta3); // Imprime : laranja
```

JavaScript – Palavra reservada this

✓ No **CONTEXTO DE EXECUÇÃO**

Toda função JavaScript, ao ser executada, gera uma associação do objeto criado pelo interpretador através da palavra reservada **this**, esse valor é constante e existe enquanto este contexto de execução existir.

No browser, o **this** "padrão" referencia o objeto global **window**. Toda função declarada no escopo global também vai possuir o objeto **window** como valor do **this**.

Exemplos:

```
function myFunc () {
alert(this);
}

let myFunc2 = function () {
alert(this);
}
let  teste1=myFunc(); //  imprime Window
let teste2=myFunc2(); // imprime Window
```

JavaScript – Relembrando a palavra reservada this

✓**OBJETOS**

Quando uma função representa um método de um objeto, o valor do **this** passa a ser o objeto referenciado. Por exemplo:

```
let myObj = {
init: function () {
alert(this);
}
};
```

```
myObj.init(); //  imprime Object
```

JavaScript – Objetos – Usando Função Construtora

Javascript não possuia definição formal de classe até ES6. Podia se utilizar um protótipo (ou função construtora).

```
function Carro(marca, modelo, ano) {
  this.marca = marca; // a marca do objeto que for criado
  this.modelo = modelo;
  this.ano = ano;
  let ativo = true;   →VARIAVEL PRIVADA (let foi criado a partir do ES6 antes era var)
this.getAtivo = function () { → MÉTODO PÚBLICO
  return ativo;
};
}
```

*** Observar uso de this para atribuir valores às propriedades do objeto com base nos valores passados para a função.*

Criando um objeto (instância):

```
let meucarro = new Carro("Fiat", "Uno", 2015);
alert(meucarro.getAtivo());
```

** ver UsandoObjetos2.html
UsandoObjetos3.html

JavaScript – Objetos – Exemplo:UsandoFuncaoConstrutora.html

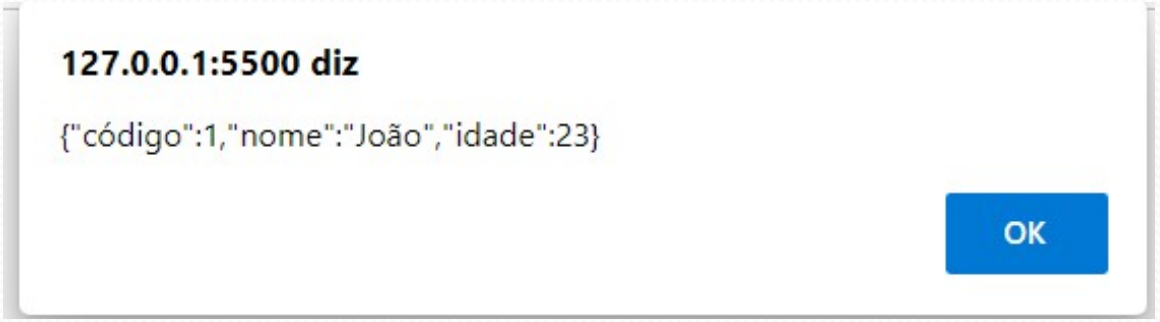
```
function Person(first,last,age) {
   this.firstname = first;
   this.lastname = last;
   this.age = age;
   let bankBalance = 7500;

   let returnBalance = function() { → MÉTODO PRIVADO
      return bankBalance;
   };
         // CRIANDO MÉTODO PÚBLICO PARA RETORNAR MÉTODO PRIVADO
   this.askTeller = function() {
   return returnBalance;
   }
}
let john = new Person('John','Smith',30);
console.log(john.returnBalance); → UNDEFINED
let myBalanceMethod = john.askTeller();
let myBalance = myBalanceMethod();
console.log(myBalance); → 7500
```

JavaScript – Objetos – Mais uma forma de criar objetos

O método Object.fromEntries pega um array de pares de chave-valor e retorna um objeto com as chaves e valores correspondentes.

Exemplo:

```
// a partir do ES10
const arr = [['código', 1], ['nome',
"João"],
 ['idade', 23]];
const obj = Object.fromEntries(arr);
alert(JSON.stringify(obj));
```



DDAOV2
DDAOV4

JavaScript – Objetos – Prototype

Protótipos são o mecanismo pelo qual objetos JavaScript herdam recursos uns dos outros.

Um protótipo permite predefinir propriedades, incluindo métodos.

https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Objects/Object_prototypes

DDAOV2 Quando você adiciona alguma função ou atributo ao prototype, esta função (ou atributo) fica disponível a todas as instâncias da classe (criadas com o new).
Quando você altera sem o prototype, somente o objeto em questão possui a função/atributo. As instâncias não o possuem.
DENILCE DE ALMEIDA OLIVEIRA VELOSO; 13/04/2022

DDAOV4 Seria uma espécie de "modelo"
DENILCE DE ALMEIDA OLIVEIRA VELOSO; 13/04/2022

JavaScript – Objetos – Prototype

```
let Pessoa = function(nome, email) {
    this.nome =  nome;

    // verifica se o e-mail foi preenchido
    if (email) {
        this.email = email;
    }
}

//não posso fazer isso antes da instanciar o Pedro
//Pessoa.email="contato@fatec.sp.gov.br.br"; // vai voltar undefined
// pois se trata de instância
//mas posso fazer isso, significa utilize o prototipo Pessoa e atribua o email

Pessoa.prototype.email = "contato@fatec.sp.gov.br.br";

let  pedro= new Pessoa("Pedro Marcos"); //não foi colocado o e-mail aqui mas já tinha
alert(pedro.email); // imprime contato@fatec.sp.gov.br

let joao = new Pessoa("Joao da Silva", "joao@da.silva"); // aqui passa e-mail  forma comum
alert(joao.email); // joao@da.silva
```

** ver UsandoObjetos5.html

Slide 76

DDAOV3 Quando você adiciona alguma função ou atributo ao prototype, esta função (ou atributo) fica disponível a todas as instâncias da classe (criadas com o new).
Quando você altera sem o prototype, somente o objeto em questão possui a função/atributo. As instâncias não o possuem.

DENILCE DE ALMEIDA OLIVEIRA VELOSO; 13/04/2022

JavaScript – Objetos – Prototype

```
// parâmetro é a raça

function Dog (breed) {
    this.breed = breed;
  };


// adicione o metodo sayHello a “classe” Dog
// para que todos os cachorros possam dizer alo

Dog.prototype.sayHello =   function() {
    console.log('Alô, este é um cachorro ' + this.breed );
    };

let yourDog = new Dog("golden retriever");
yourDog.sayHello();

let myDog = new Dog("dachshund");
myDog.sayHello();
```

** ver UsandoObjetos6.html

JavaScript – Objetos – Métodos – Exemplo 1

✓Métodos são funções associadas a objetos.

```
function Aluno() {
        let nome; // fica encapsulado
        let ra;

        this.setNome = function (vNome) {
                this.nome = vNome;
        }

        this.setRa = function (vRa) {
                this.ra = vRa;
        }

        this.getNome = function () {
                return this.nome;
        }

        this.getRa = function () {
                return this.ra;
        }
```

** ver UsandoObjetos4.html

JavaScript – Objetos – Métodos – Exemplo 1

```
                this.mostraDados = function () {
                        alert("Nome do aluno: " + this.nome +
"\nRa: " + this.ra);
                }
        }

        let objAluno = new Aluno();

        objAluno.setNome("Joaquim");
        objAluno.setRa("1234");
        objAluno.mostraDados();
        alert(objAluno.getNome()); //volta somente o nome
```

JavaScript – Objetos – Métodos – Exemplo 2

** ver UsandoObjetos2.html

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
        <meta charset="UTF-8">
        <title>Objetos</title>
</head>
<body>
        <script>
                function car1(make, model, year, owner) {
                        this.make = make;
                        this.model = model;
                        this.year = year;
                        this.owner = owner;

                        this.displayCar = function () {
                                let result = " Big Car " + this.year + " " +
this.make + " " + this.model;
                                alert(result);
                        }
                };
```

JavaScript – Objetos – Métodos

```
                    // função criada  internamente
                    let meuCarro1 = new car1("marca1", "modelo1", "ano1", "proprietario1");
                    meuCarro1.displayCar();
```

→ Big Car ano1 marca1 modelo1

```
    //criando a função externamente
                    function displayCar() {
                            let result = " Big Car " + this.year + " " + this.make + " " +
this.model;
                            alert(result);
                    };

                    function car2(make, model, year, owner) {
                            this.make = make;
                            this.model = model;
                            this.year = year;
                            this.owner = owner;
                            this.displayCar = displayCar; //atribuindo a função
                    };

                    let meuCarro2 = new car2("marca2", "modelo2", "ano2",
"proprietario2");
                    meuCarro2.displayCar();
```

→ Big Car ano2 marca2 modelo2

```
            </script>
</body>
</html>
```

Exercício – Valor de Z ?

```
let obj = {
        x: 10,
        y: 2,
 z: function () { return this.x **
this.y }
};
alert(obj.z());
```

JavaScript – Objetos – Copiando Objetos

Quando a operação de cópia é realizada, duas variáveis apontam para exatamente o mesmo objeto, assim, mudanças em uma são refletidas na outra:

```
let obj1 = new Object();
let obj2 = obj1;
obj1.name = “Oscar”;
alert(obj2.name); //”Oscar”
```

JavaScript – Objetos - Herança – Exemplo: ExemploUsandoHeranca.html (1)

```
<script>
        function Pessoa() {    → "superclasse"
                let nome;
                this.getNome = function () {
                        return nome;
                };
                this.setNome = function (value) {
                        nome = value;
                };
        }
```

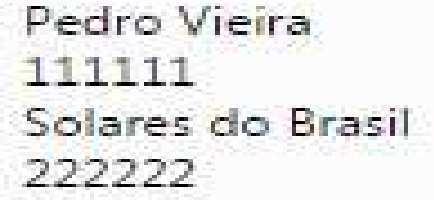
JavaScript – Objetos - Herança – Exemplo: ExemploUsandoHeranca.html (1)

```
function PessoaJuridica() { → "subclasse"
    let cnpj;
    this.getCNPJ = function () {
        return cnpj;
    };
    this.setCNPJ = function (value) {
        cnpj = value;
    };
}

function PessoaFisica() {→ "subclasse"
    let cpf;
    this.getCPF = function () {
        return cpf;
    };
    this.setCPF = function (value) {
        cpf = value;
    };
}
```

JavaScript – Objetos - Herança – Exemplo: ExemploUsandoHeranca.html (1)

```
// herança
PessoaFisica.prototype = new Pessoa();
PessoaJuridica.prototype = new Pessoa();

>>>> Criando novo objeto <<<<<<
nPessoaFisica = new PessoaFisica();
nPessoaJuridica = new PessoaJuridica();
nPessoaFisica.setCPF('11111');
nPessoaFisica.setNome('Pedro Vieira');
nPessoaJuridica.setCNPJ('222222');
nPessoaJuridica.setNome('Solares do Brasil ');
alert(nPessoaFisica.getNome() + '\n' + nPessoaFisica.getCPF() + '\n' + nPessoaJuridica.getNome() + '\n' + nPessoaJuridica.getCNPJ());
</script>
```

→



JavaScript – Objetos - Herança – Exemplo 2

```
// “classes”
    function Animal(name, numLegs) {
        this.name = name;
        this.numLegs = numLegs;
        this.isAlive = true;
    }

    function Penguin(name) {
        this.name = name; //repetiu name aqui porque não tem metodo no animal para receber o nome
        this.numLegs = 2;
    }

    function Emperor(name) {
        this.name = name; // repetiu name aqui porque não tem metodo no animal para receber o nome
        this.saying = "Oi Oi ";
    }
    Penguin.prototype = new Animal();
    Emperor.prototype = new Penguin();
    let myEmperor = new Emperor("Juli ");
            console.log(myEmperor.name); // deve imprimir “Juli”
            console.log(myEmperor.saying); // deve imprimir “Oi Oi"
            console.log(myEmperor.numLegs); // deve imprimir 2
            console.log(myEmperor.isAlive); // deve imprimir true
```

JavaScript – Classes – a partir ES6

** ver arquivo ExemploClasses.html

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
    <meta charset="UTF-8">
    <title>Exemplo Classes </title>
</head>

<body>

    <script>//você deve declarar seus atributos dentro do método construtor,
        {//Só pode existir um método especial com o nome "constructor"
            class Retangulo {
                constructor(altura, largura) {
                    this.altura = altura;
                    this.largura = largura;
                }
```

JavaScript – Classes – a partir ES6

```
calculaPerimetro() {
            return 2*(this.altura + this.largura);
        }
    }

    objRetangulo = new Retangulo(10, 5);

    alert(objRetangulo.calculaPerimetro());

}
</script>

</body>

</html>
```

JavaScript – Classes – a partir ES6

** ver arquivo ExemploClasses2.html

```
<!DOCTYPE html>
<html lang="pt-br">

<head>
    <meta charset="UTF-8">
    <title>Exemplo Classes </title>
</head>
<body>
    <script>
        {
            class Retangulo {
              constructor() {
                  this._altura;
                  this._largura;
              }
```

JavaScript – Classes – a partir ES6

```
// ATRIBUTOS NAO PODEM TER OS MESMOS NOMES DE GET/SET E METODOS
    get altura() {
        return this._altura;
      }

      get largura() {
        return this._largura;
      }

    set altura(value) {
    this._altura = value;
      }

      set largura(value) {
        this._largura = value;
      }
```

JavaScript – Classes – a partir ES6

```
        calculaPerimetro() {
            return 2*(this.altura + this.largura);
         }
      }

objRetangulo = new Retangulo();
objRetangulo.altura = 50;
objRetangulo.largura = 20;
alert(objRetangulo.calculaPerimetro());
alert("altura="+objRetangulo.altura+"largura="+objRetangulo.largura);

 }
   </script>
</body>
</html>
```

JavaScript – Classes – a partir ES6

** ver arquivo ExemploClasses3.html

```
// exemplo1 - classe com nome
    class Quadrado {
        constructor(lado) {
            this.lado = lado;

        }
        calculaArea() {
            return this.lado * this.lado;
        }
    };

    objQuadrado = new Quadrado(3);
    alert("exemplo1:" + objQuadrado.calculaArea());
```

JavaScript – Classes – a partir ES6

** ver arquivo ExemploClasses3.html

```
// exemplo2 - expressao de classe sem nome
let Quadrado1 = class {
    constructor(lado) {
        this.lado = lado;
    }
    calculaArea() {
        return (this.lado * this.lado);
    }
};

objQuadrado1 = new Quadrado1(3);
alert("exemplo2:" + objQuadrado1.calculaArea());
```

JavaScript – Classes – a partir ES6

** ver arquivo ExemploClasses3.html

```
//   exemplo3 -  classe com nome com atributo sem receber parâmetros no constructor
      class Quadrado2 {
          #_descricao; // encapsulamento válido a partir do ES9
          constructor() {
              this._lado; // encapsulamento
          }
          get lado() {
              return this._lado;
          }
          set lado(value) {
              this._lado = value;
          }
          calculaArea() {
              return (this._lado * this._lado);
          }
```

JavaScript – Classes – a partir ES6

** ver arquivo ExemploClasses3.html

```
//   exemplo3 -  classe com nome com atributo sem receber parâmetros no constructor

        set descricao(value) {
           this._desricao = value;
        }
        get descricao() {
            return this._descricao;
        }
    }

    objQuadrado02 = new Quadrado02();
    objQuadrado02.lado = 3;
    alert("exemplo3:" + objQuadrado02.calculaArea())
    objQuadrado02._descricao = "quadrado azul";
    alert("exemplo3=" + objQuadrado02.descricao);
```

JavaScript – Herança de Classes – a partir ES6

** ver arquivo ExemploHeranca.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
   <meta charset="UTF-8">
   <title>Exemplo Heranca </title>
</head>
<body>
   <script>
      {
         class Animal {
            constructor(nome) {
               this.nome = nome;
            }
```

JavaScript – Herança de Classes – a partir ES6

** ver arquivo ExemploHeranca.html

```
    som() {
        alert(this.nome + ' emite som ');
    }
}

class Galinha extends Animal {
    som() {
        alert(this.nome + ' cacareja ');
    }
}
```

JavaScript – Herança de Classes – a partir ES6

** ver arquivo ExemploHeranca.html

```
        class GalinhaAngola extends Galinha {
            som() {
                alert(this.nome + ' fala tô fraco ');
            }
        }
        let objGalinha = new Galinha('Leka');
        objGalinha.som();

        let objGalinha2 = new GalinhaAngola('Gray');
        objGalinha2.som();
}
    </script>
</body>
</html>
```

JavaScript – Herança de Classes

** ver arquivo ExemploHeranca2.html

```
class Poligono {
        constructor(comprimento) {
            this.comprimento = comprimento;
            this.nome = "Polígono";
        }
    }

    class Quadrado extends Poligono {
        constructor(comprimento,nome) {
            // super chama o construtor da classe pai
            super(comprimento);
        // Nas classes filhas, super() deve ser chamado antes de usar o this, senão dá erro
        //  this  se refere a classe filha Quadrado
            this.nome = nome;
        }

        areaQuadrado = function() {
            return (this.comprimento * this.comprimento);
        }
    }
```

JavaScript – Herança de Classes

**** ver arquivo ExemploHeranca2.html**

```
let objQuadrado = new Quadrado(2, "Quadrado Vermelho");
        alert(objQuadrado.areaQuadrado());
        alert(objQuadrado.nome);
```

JavaScript – Métodos e Atributos Privados

```
class ContaBancaria {
        #cpf; // atributo privado
            constructor(nome, saldoInicial) {
                this.nome = nome; //atributo publico
                this.saldo = saldoInicial;
            }

            setcpf(value) {
                cpf = value;
            }
            getCpf() {
                return cpf;
            }
            getSaldo() {
                return this.saldo;
            }
```

JavaScript – Métodos e Atributos Privados

```
        // metodo público
        depositar(valor) {
            this.saldo += valor;
            this.#registrarTransacao(valor);    // Utilizando o
modificador # para método privado
        }

        // metodo privado
        #registrarTransacao(valor) {
            console.log(`Depósito de R$ ${valor} realizado.
            Saldo R$ ${this.saldo}`);
        }
    }
    const conta = new ContaBancaria("Mary",1000,234);
    conta.setcpf(123); // acessa conta.nome mas não conta.cpf
    conta.depositar(50);
    // ver no console se aparece
```

JavaScript - Objetos – Exercício

Disponibilizar como Atividade12 no GITHUB.
→ Seunome/PWEB/Atividade12

1. Utilize uma função construtora para o Retângulo receber (x,y) ou seja, base e altura, com um método para calcular a área. Criar um objeto e executar o método que calcula a área. Não precisa utilizar get e set na função construtora.

2. Crie uma Classe tipo Conta, com as propriedades *nome correntista, banco, numero da conta e saldo*. Crie utilizando herança duas novas classes: Corrente com Saldo Especial e Poupanca com Juros, Data Vencimento. Receber os dados via get e set. Criar um objeto de cada uma: Corrente e Poupanca e mostrar os seus dados.

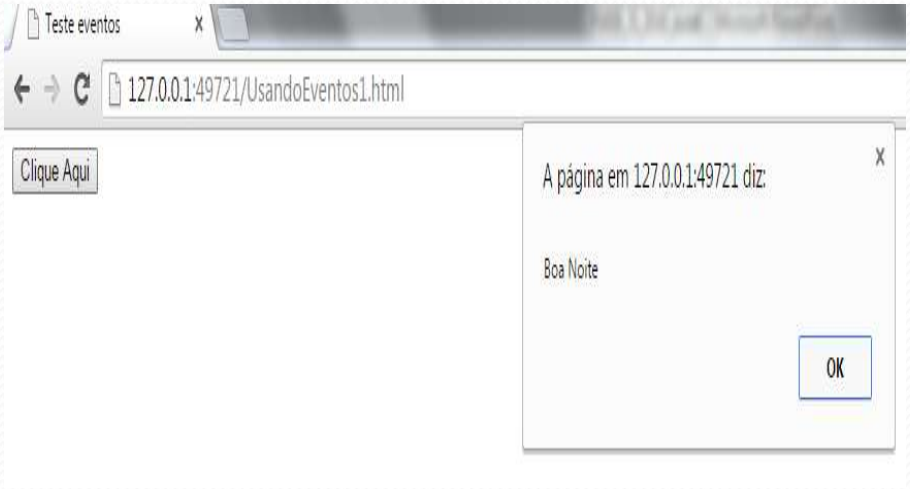
** Receber dados com prompt ou inputs

JavaScript – Eventos

- ✓ São fatos que ocorrem durante a execução do programa e podem ser detectados por um script;
- ✓ São muito usados em JavaScript e viabilizam a interatividade em uma página Web;
- ✓ O tratamento dos eventos pode ser a chamada de funções do script;
- ✓ Exemplos: clique do mouse, carregamento ou abandono de uma página web ou imagem, envio de um formulário html, uma tecla pressionada, seleção de texto, etc.

JavaScript – Eventos Exemplo: UsandoEventos1.html

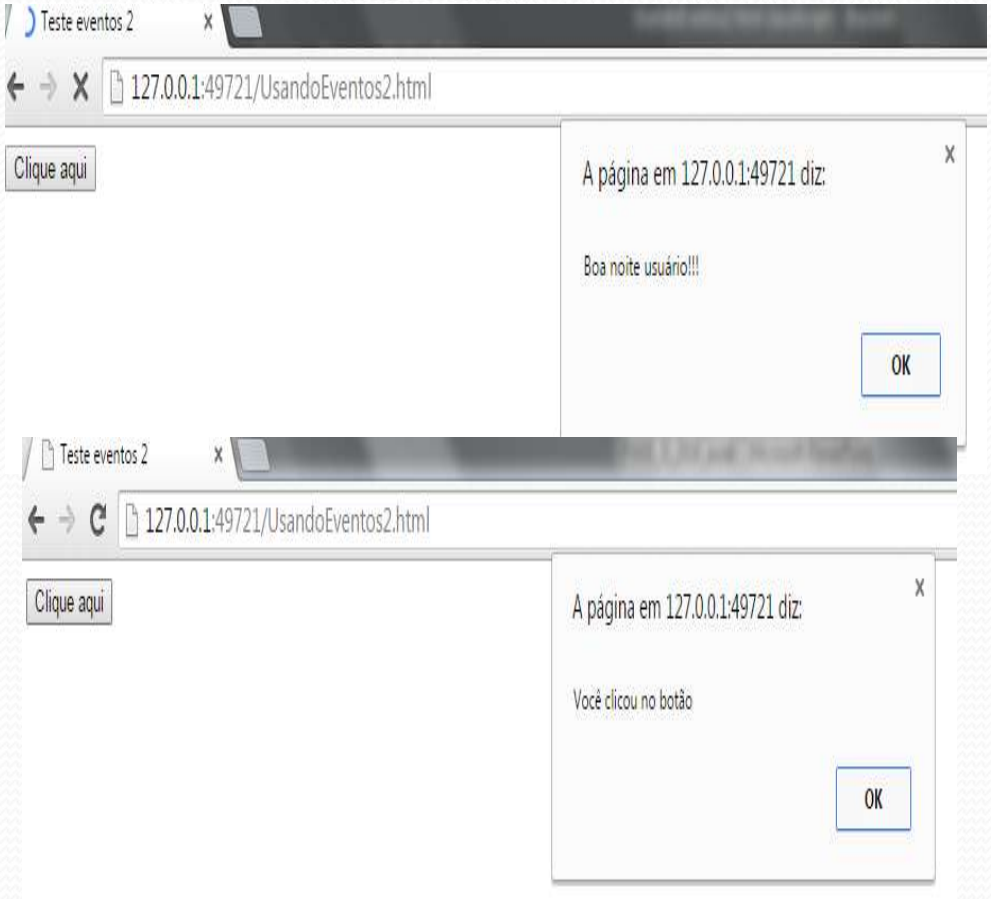
```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Teste eventos</title>
</head>
<body>
    <input type="button"
    value="Clique Aqui"
    onclick="alert('Boa Noite');">
    <!-- tag HTML com eventos -->

</body>
</html>
```



Associa evento onClick do botão a função alert

JavaScript–Eventos Ex.: UsandoEventos2.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Teste eventos 2</title>


<script>
function ler() {
alert("Boa noite usuário!!!");
}

function tocar() {
    alert("Você clicou no botão");
        }
</script>
</head>
    <body onload='ler();'>
<input  type="button"  value="Clique  aqui"
    onclick='tocar();' />
</body>
</html>
```



Quando a página for carregada é chamada a função ler

JavaScript – Tipo de Eventos – Quando ocorre???

onload - no carregamento da página (no body)

onunload - na descarga (saída) da página (body)

onsubmit - quando um botão tipo Submit recebe um click do mouse

onchange - quando o objeto perde o foco e houve mudança de conteúdo.

onblur – quando o objeto perde o foco, independentemente de ter havido mudança.

onfocus - quando o objeto recebe o foco.

onclick - quando o objeto recebe um click do mouse.

→ Quando o objeto perde o foco a sequência é: primeiro onchange e depois o onblur

JavaScript – Tipo de Eventos – Quando ocorre???

onmouseover - quando o ponteiro do mouse passa sobre o objeto (move sobre a imagem), pela primeira vez.

onselect - quando o objeto é selecionado.

ondblclick - quando o objeto recebe duplo clique do mouse.

onkeydown - quando uma tecla é pressionada (qualquer uma por exemplo ctrl, shift, f1, f2 etc).

onkeypress - quando uma tecla alfanumérica é pressionada .

onkeyup - quando uma tecla é liberada.

onmousedown - quando o botão do mouse é pressionado.

onmouseup - quando o botão do mouse é liberado.

onmousemove - enquanto o mouse se move dentro do elemento, atualizando a posição do ponteiro.

JavaScript – Tipo de Eventos – Quando ocorre???

onmouseout - quando o mouse é movido para fora da borda do objeto.

onhelp - quando o usuário pressiona a tecla F1.

oncontextMenu - quando o usuário dá um clique na área do documento para abrir menu de contexto

onabort - quando o usuário abortar o elemento antes de terminar o carregamento (o script, a imagem)

onerror - quando o arquivo de imagem não é encontrado ou está corrompido

onresize - quando o usuário redimensiona a página ou frames(quadros)

JavaScript – Eventos Ex.: UsandoEventos4.html

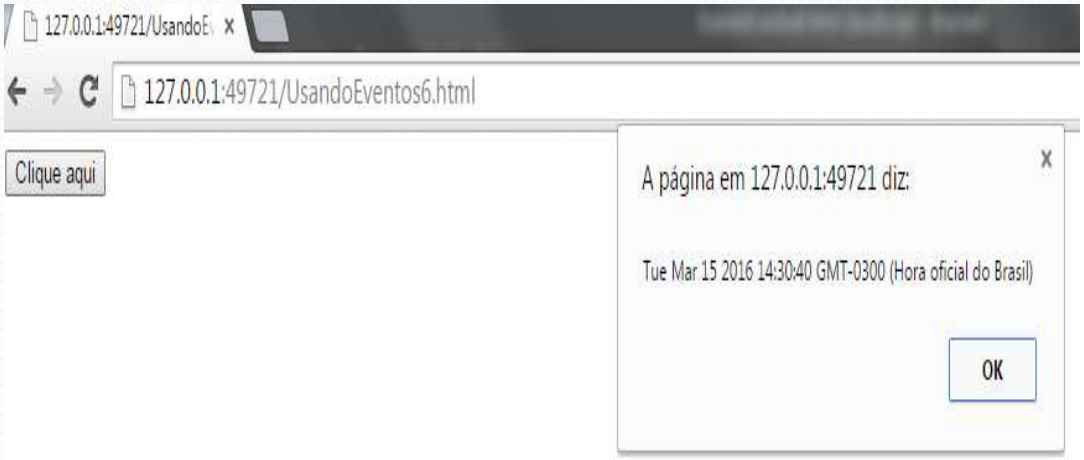
```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Teste eventos 4</title>
    <script>
        function qualTecla() {
            let tecla = String.fromCharCode(event.keyCode);
            alert( "Você clicou " + tecla);
        }
    </script>
</head>
<body>
        <img src="minhafoto.jpg" onerror="alert('Epa cadê a foto???');" />
        Nome : <input type="text" onkeypress="qualTecla();">
    </body>
</html>
```

JavaScript – Eventos Ex.: UsandoEventos5.html

```
<body
    <div onmousedown="mDown(this)" onmouseup="mUp(this)" onmousemove="mMove(this)"
    onmouseout="mOut(this)" style="background-
    color:#FF69B4;width:200px;height:200px;padding:40px;">Clique aqui e segure</div>
    <script>
        function mDown(obj) {
            obj.style.color = "#00FFFF";
            obj.innerHTML = "Solte o clique";}
        function mUp(obj) {
            obj.style.color = "#ffff00";
            obj.innerHTML = "Obrigado";}
        function mMove(obj) {
            obj.style.color = "#FF0000";
            obj.innerHTML = "Moveu para cima do Objeto";
        function mOut(obj) {
            obj.style.color = "#AA3000";
            obj.innerHTML = "Saiu da Borda do Objeto";}
   < /script>
</body>
```

JavaScript – Eventos Ex.: UsandoEventos6.html

```
<html>
<head>
   <meta charset="UTF-8"/>
</head>
<body>
   <button id="btn">Clique aqui</button>

   <script>
      function exibirMensagem()
      {
         let data = new Date();
         alert(data.toString());
      }


      let btn = document.getElementById("btn");

      btn.addEventListener("click", exibirMensagem);
   </script>
</body>
</html>
```



Adiciona o evento ao controle

DDAOV7
DDAOV8

Objeto Event

O *event* é um objeto especial que é enviado para um *handler* de evento à cada ocorrência. O *handler* de evento recebe esse objeto como um parâmetro. As propriedades do objeto *event* oferecem mais informações sobre o evento que ocorreu. As propriedade disponíveis são:

- `type` — Tipo de evento que ocorreu, como *mouseover*.
- `target` — Objeto de destino para o evento (como o documento ou um *link*).
- `which` — Valor numérico que especifica o botão do mouse que foi clicado para eventos de *mouse* ou a tecla que foi pressionada para eventos de teclado.
- `modifiers` — Lista de chaves de modificador que foram pressionadas durante um evento de teclado ou de *mouse* (como Alt, Ctrl e Shift).
- `data` — Lista de dados arrastados e soltos para eventos de arrastar e soltar.
- `x e y` — Posição x e y do mouse quando ocorreu o evento, medida a partir do canto superior esquerdo da página.
- `screenX` — Posição X do *mouse* , medida do canto superior esquerdo da tela.
  `screenY` — Posição Y do *mouse* , medida do canto superior esquerdo da tela.
- `keyCode` — Código ASCII da Tecla pressionada.

*O **x e y** uma propriedade que fornece as coordenadas horizontal e vertical em relação ao elemento DOM que ocorreu, por exemplo e o evento ocorreu em um botão, event.x seria a coordenada X do mouse em relação ao botão.*

screenx e screeny são coordenadas em relação à tela inteira.

Slide 114

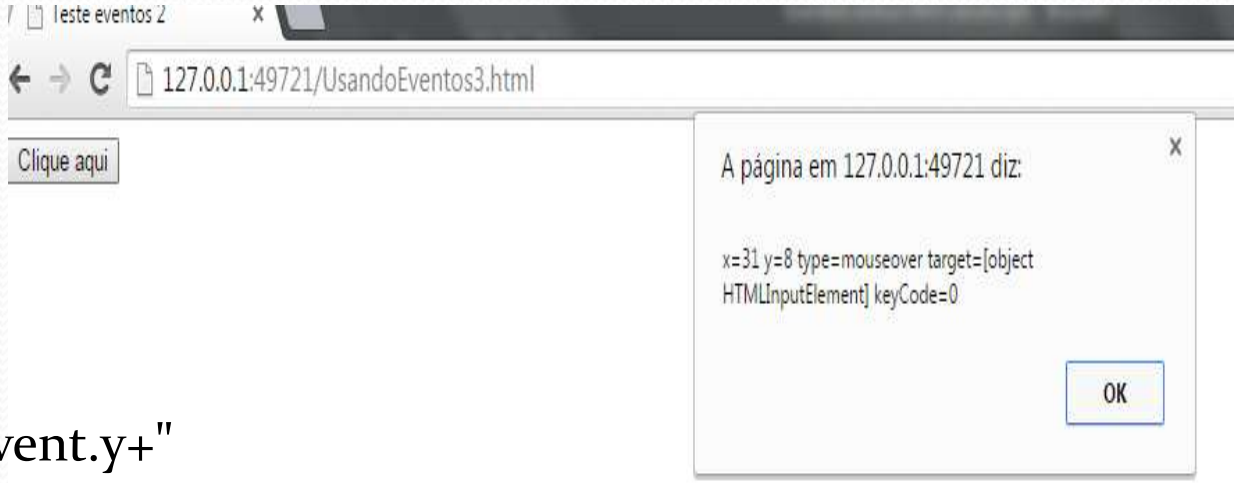
DDAOV7 A diferença entre event.x e event.screenX é o ponto de referência usado para calcular a coordenada X.
event.x é a coordenada X do mouse em relação ao elemento DOM onde o evento ocorreu. Por exemplo, se o evento ocorreu em um botão, event.x seria a coordenada X do mouse em relação ao botão.
event.screenX é a coordenada X do mouse em relação à tela inteira. Por exemplo, se o evento ocorreu em um botão, event.screenX seria a coordenada X do mouse em relação à tela inteira, independentemente da posição do botão na tela.
Em outras palavras, event.x é a coordenada X do mouse dentro do elemento DOM, enquanto event.screenX é a coordenada X do mouse na tela inteira.
DENILCE DE ALMEIDA OLIVEIRA VELOSO; 22/10/2023

DDAOV8 Se o evento não for relacionado ao mouse, o valor de event.x pode não ser confiável ou ter um significado diferente dependendo do tipo de evento.
DENILCE DE ALMEIDA OLIVEIRA VELOSO; 22/10/2023

Objeto Event

Exemplo: UsandoEventos3.html

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
     <title>Teste eventos 2</title>
      <script>
            function teste(){
              alert("x="+event.x+" y="+event.y+"
type="+event.type+" target="+event.target+"
keyCode="+event.keyCode + “Screenx="+event.Screenx+"
Screeny="+event.Screeny);
}
</script>
</head>

<body>
  <input type="button" value="Clique aqui" onClick='teste()' />
</body>
</html>
```



** no seu exemplo tem vários br’s para rolar a página

EXERCÍCIO - EVENTOS

Disponibilizar como Atividade13 no GITHUB.
→ Seunome/PWEB/Atividade13

Encontre na rede 3 figuras: janela aberta, janela fechada e janela quebrada.

Criar uma página que no carregamento mostra a janela fechada com um título "Abra a Janela". Se mover com o mouse sobre a imagem da janela fechada, mostra a janela aberta (abre a janela), se sair com o mouse da imagem mostra a janela fechada (fecha a janela). Se clicar sobre a imagem mostra a imagem da janela quebrada (quebra a janela).

Colocar um texto em um H1 informando "Janela aberta", "Janela Fechada" ou "Janela Quebrada"

Referências

CODEACADEMY. Cursos Gratuitos. https://www.codecademy.com/pt Acesso em: Jan. 2025.

CROCKFORD, Douglas. http://crockford.com/javascript/ Acesso: Mai.2021.

CSS. http://del.icio.us/carlosbazilio/{css+html} Acesso em: Jan.2015.

JS. Livro de JavaScript. https://github.com/getify/You-Dont-Know-JS Acesso em: Jan.2021.

JS1. http://www.significados.com.br/javascript/ Acesso em: Jan.2015

JS2. https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/JavaScript_Vis%C3%A3o_Geral Acesso em: Jan.2025

SILVA. Maurício Samy . JavaScript Guia do Programador. Editora Novatec.

JSOBJETOS. https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Trabalhando_com_Objetos Acesso em: Jan.2025.

MOZILLA. JavaScript.. https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript Acesso em: Set.2024.

OPERADORES. https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#Assignment_operators Acesso em: Jan.2025.

PEREIRA. Fábio M. Pereira. JavaScript Básico. DESENVOLVIMENTO DE SISTEMAS WEB – 2014.1 UNIVERSIDADE ESTADUAL DO SUDOESTE DA BAHIA CURSO DE CIÊNCIA DA COMPUTAÇÃO. 2014.

PRECEDENCIA. https://msdn.microsoft.com/pt-br/library/z3ks45k7(v=vs.94).aspx Acesso em: Jan.2025.

Referências

- ✓ W3SCHOOLS. Disponível em: http://www.w3schools.com/ Acesso em: Jan.2025.
- ✓ W3. Disponível em: http://www.w3.org/ Acesso em: Jan.2025.