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()**;

✓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>
```

Exemplo3:

```
var 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

JavaScript – 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:

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

JavaScript – TIPO DATE

- Obter data e hora

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

Exemplo:

```
var 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

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:

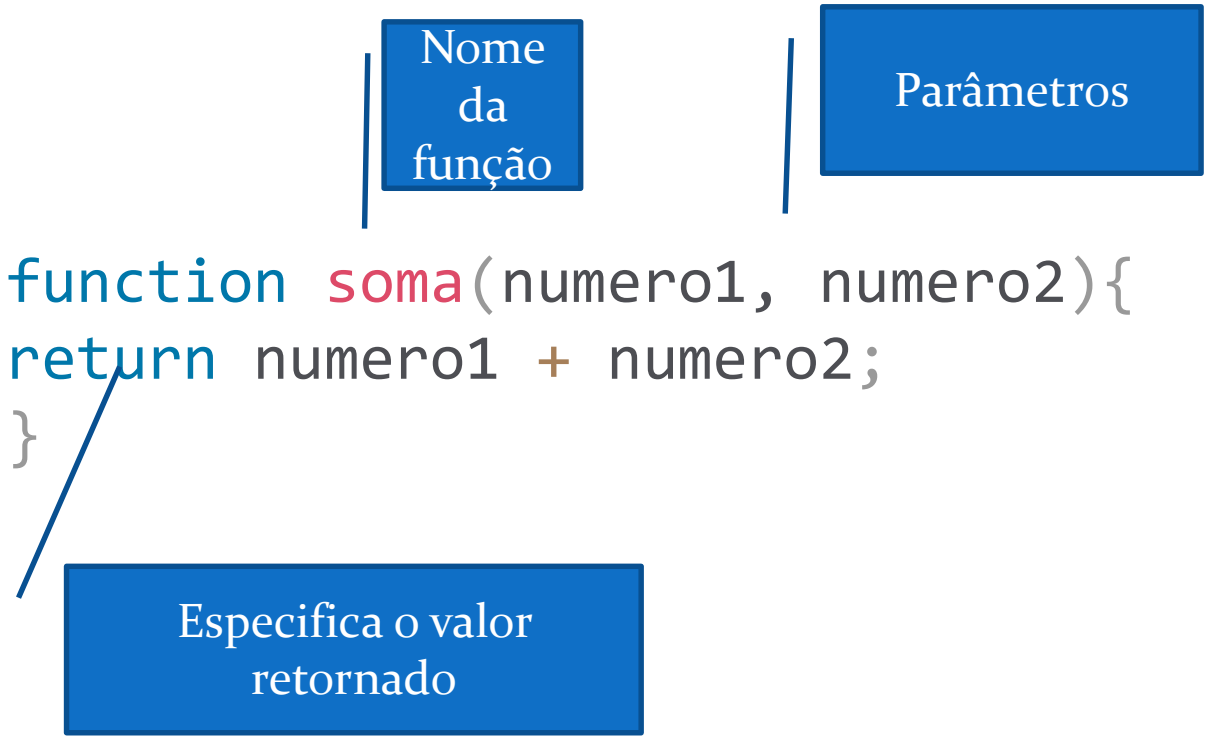
```
var dataCompleta = new Date();
var diaSemana = dataCompleta.getDay();

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.

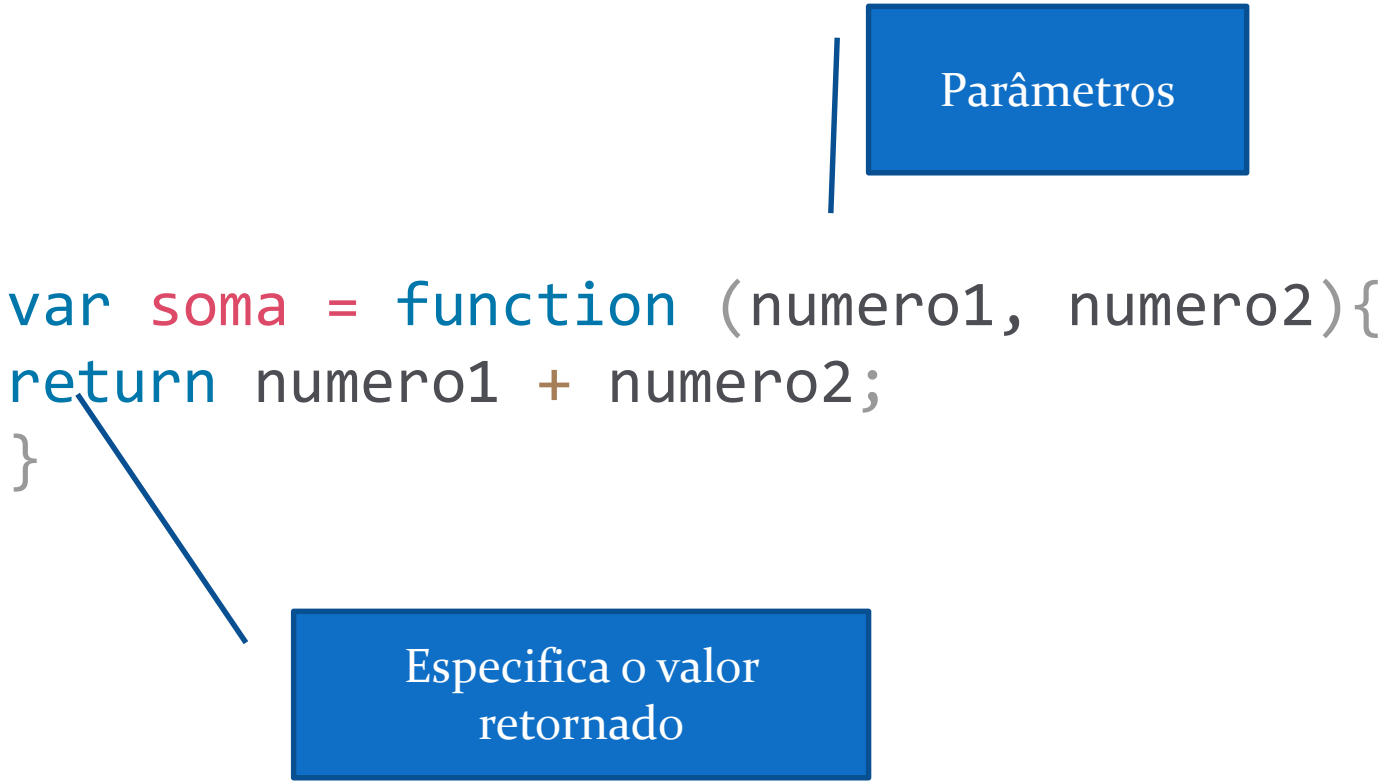


** Existem as funções intrínsecas: exemplos: parseFloat, eval, etc.

DV3
DDAOV1

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

DDAOV1

DENILCE DE ALMEIDA OLIVEIRA VELOSO; 29/09/2021

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..

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

// uma função pode retornar um alert
assim: return alert(“par”), para chamar
var 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:

```
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 – 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];
}

var 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

        var 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

```
<!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>
```

Criar **duas** funções:

Uma que receba três números como parâmetros e retorne o maior deles.

Usar um arquivo js externo.

Outra que receba três números e coloque em ordem crescente.

Disponibilizar como Atividade9 no GITHUB.

→ Seunome/PWEB/Atividade9

JavaScript – FOR

Executa um bloco de código um número de vezes.

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

JavaScript – FOR – Pense rápido

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

JavaScript – Instrução do ... while

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

Exemplo1:

```
var numero = 1;
do {
  document.write (numero+" ");
  numero++;
}
while (numero<=10)  → imprime de 1 até 10
```

Exemplo2:

```
<p id="teste"></p>
<script>
    var i = 0;
    do {
        document.getElementById("teste").innerHTML += 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:

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

JavaScript – Instruções break e continue

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:

```
var 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;

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:

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

Exemplo2:

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

JavaScript – ARRAYS

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

var notasMusicais = new Array(); *OU* var meuArray = [];

//os dois tem comprimeot(length)=0

var notasMusicais = new Array(7); var notasMusicais = [7]

(comprimento 7 e todos com valores undefined // comprimento 1 e posição 0=7)

→ var notasMusicais = (7), não significa nada

✓armazenam listas de dados;

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

var a = new Array(8); (comprimento 8, undefined)
var b = new Array(8, 9); (comprimento 2, valores 8 e 9 nas 1ª e 2ª posições)
var c = [8]; (comprimento 1 e dado 8)

JavaScript – ARRAYS

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

Ex. var myObject = {};
var bol = true;
var myArray=[1,bol,"Fatec",myObject];
→Posição de cada dado é fixa

Observação:

```
var 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";
  notasMusicais[7] = "xx";

  alert(notasMusicais); → do,ré,mi,fá,sol,lá,si,xx
```

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

JavaScript – ARRAYS - Exemplos:

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

//omitindo o operador New, literal
var idades = Array(4);
idades[0] = 10;   → atribuição dos valores
idades[1] = 20;
idades[2] = 30;
idades[3] = 40;

//omitindo o operador com dados, notacao literal
var cidades = Array("Sorocaba");

//tipos de dados diferentes
var misturados = [34, "doce", "azul", 11];

// array vazio
var nada = [];

alert("nomes: " + nomes + " comprimento:" + nomes.length);   →nomes:Clarice  Lispector,Carlos
                                                              Drumond,José de Alencar: Comprimento:3
alert("idades:" + idades + " comprimento:" + idades.length); → idades:10,20,30,40:Comprimento:4
alert("cidades:" + cidades); → cidades:Sorocaba
alert("misturados:" + misturados); → misturados:34,doce,azul,11
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:

Posicao 0

<u>Exemplo1:</u>

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

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

<u>Exemplo2:</u>

Posicao 0

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

** **Array Bidimensional: var novoArray = [[1,2,0],[3,4,0],[5,6,0]] → 3 linhas e 3 colunas**

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

```
var 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).

```
var 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)

```
var 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)

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

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

O método PUSH adiciona um elemento ao final de um array.

```
function incrementaFinal(_array)  {
    var newArray = [];
    newArray = _array.slice(0);  //extrai do início até o fim de array
    var lastNumber = newArray.pop(); //exclui o último no caso o 5
    newArray.push(lastNumber + 1); //5+1 → incluindo o 6
    return newArray;
}
var x= Array(1,2,3,4,5)
 alert(incrementaFinal(x)); → retorna 1,2,3,4,6
```

No exemplo do slide anterior se:

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

alert(incrementaFinal(x));

→ retorna o quê????

Qual é o resultado da variável total?

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

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

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

JavaScript – FOR – Exercício

Considerando o código js abaixo, supondo que a entrada fossem números de 1 a 9, informe qual será o resultado na tela.

```
var arValores1 = new Array(9);
var arValores2 = new Array(9);
var sprintValores1 = "";
var sprintValores2 = "";
var sVal;
for (x=0; x<=8; x++) {
   sVal = prompt("entre com número");
   arValores1[x] = eval(sVal);
   sprintValores1 += arValores1[x] + "\n";
   r = x%2;
   if (r==0) {
      arValores2[x] = arValores1[x] * 5;
      sprintValores2 += arValores2[x] + "\n";
    }
   else {
     arValores2[x] = arValores1[x] + 5;
    sprintValores2 += arValores2[x] + "\n";
    }
 }
 alert(sprintValores1 + "\n" + sprintValores2);
```

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

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

//Ou

var 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:

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

JavaScript – Criando Objetos - Propriedades

```
var 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]);
```

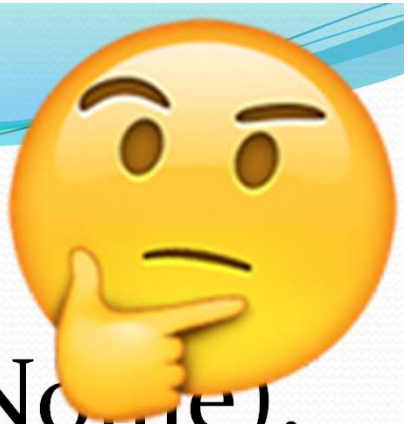
** ver UsandoObjetos1.html

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

JavaScript – Criando Objetos – Propriedades – dot notation

```
<script>
    var objeto = {};
    var nome_propriedade = 'nome';
    objeto[nome_propriedade] = 'Jose';
    alert(objeto.nome); // imprime Jose

    var objeto2 = {};
    objeto2["nome"] = "Joao";
    alert(objeto2.nome);  //imprime Joao
</script>
```

Exercício



Criar o Objeto:

Aluno01(propriedades: RA, Nome).

Atribua valores para as propriedades.

** Utilizar 3 formas diferentes

Disponibilizar como Atividade10 no GITHUB.

→ Seunome/PWEB/Atividade10

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

Supondo Json:

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

Para pegar os dados:

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

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

*** ver exemplo UtilizandoJSON.html*

JavaScript – Criando Objetos - Propriedades

→ Usando o for para imprimir as propriedades

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

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

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);
}

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

JavaScript – 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:

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

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

JavaScript – Objetos – Usando Função Construtora

Javascript não possui definição formal de classe. Pode 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;
  var ativo = true;  →VARIAVEL PRIVADA
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):

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

JavaScript – Objetos – Exemplo:UsandoFuncaoConstrutora.html

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

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

JavaScript – Objetos – Exemplo: UsandoPrototype.html

Um protótipo permite predefinir propriedades, incluindo métodos.

** ver UsandoObjetos7.html

```
var 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";

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

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

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 );
    };

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

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

JavaScript – Objetos – Métodos – Exemplo 1

✓Métodos são funções associadas a objetos.

** ver UsandoObjetos4.html

```
function Aluno() {
        var nome; // fica encapsulado
        var 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;
        }
```

JavaScript – Objetos – Métodos – Exemplo 1

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

        var objAluno = new Aluno();

        objAluno.setNome("Joaquim");
        objAluno.setRa("1234");
        objAluno.mostraDados();
```

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

JavaScript – Objetos – Métodos

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

      //criando a função externamente
                    function displayCar() {
                              var 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
                    };

                    var meuCarro2 = new car2("marca2", "modelo2", "ano2",
"proprietario2");
                    meuCarro2.displayCar();    →   Big Car ano2 marca2 modelo2
          </script>
</body>
</html>
```

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:

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

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

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

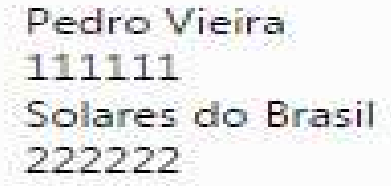
JavaScript – Objetos - Herança – Exemplo: ExemploUsandoHeranca.html (1)

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

function PessoaFisica() {→ "subclasse"
    var 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();
    var 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 - Objetos – Exercício

Disponibilizar como Atividade10 no GITHUB.
→ Seunome/PWEB/Atividade11

1. Crie 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.

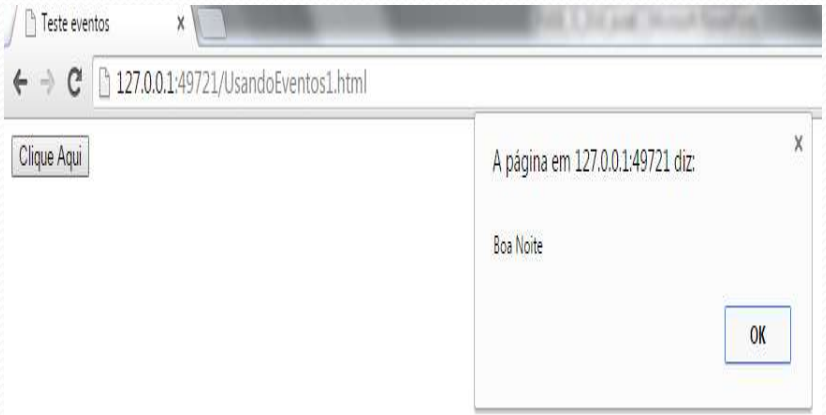
2. Crie uma nova função tipo Conta, com as propriedades *nome correntista, banco, numero da conta e saldo*. Crie utilizando herança duas novas funções: Corrente com Saldo Especial e Poupanca com Juros, Data Vencimento. Receber os dados via get e set. Criar um objeto de cada uma: Corrente e Poupanca e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

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

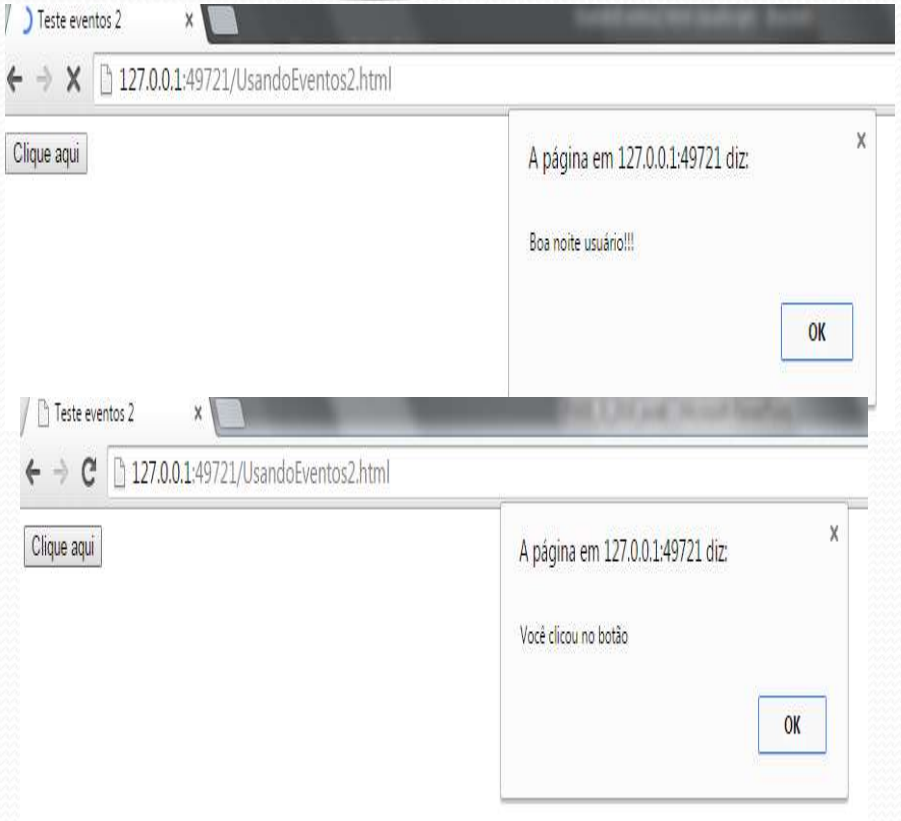
```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <title>Teste eventos</title>
</head>
<body>
    <input type="button"
    value="Clique Aqui"
    onclick="alert('Boa Noite');">
    <!-- tag HTML com eventos -->

</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, independente 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).

onselect - quando o objeto é selecionado.

ondblclick - quando o objeto recebe duplo clique do mouse

onkeydown - quando uma tecla é pressionada

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 - quando o mouse é movido sobre o objeto (enquanto está sobre o elemento div)

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

onstop - quando o usuário clica no Stop do navegador

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

onabort - quando o usuário abortar a página antes de terminar o carregamento

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

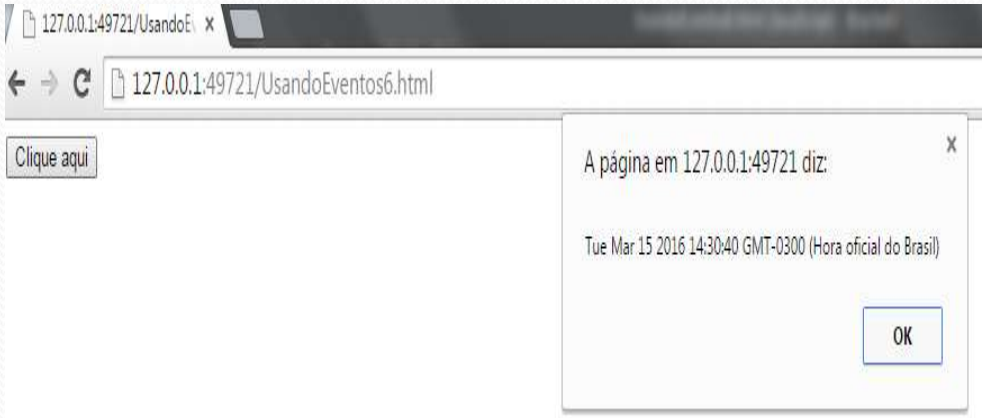
```
<!DOCTYPE html>
<html lang="pt-br">

<head>
	<meta charset="UTF-8">
	<title>Teste eventos 4</title>

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


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

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



Adiciona o evento ao controle

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:

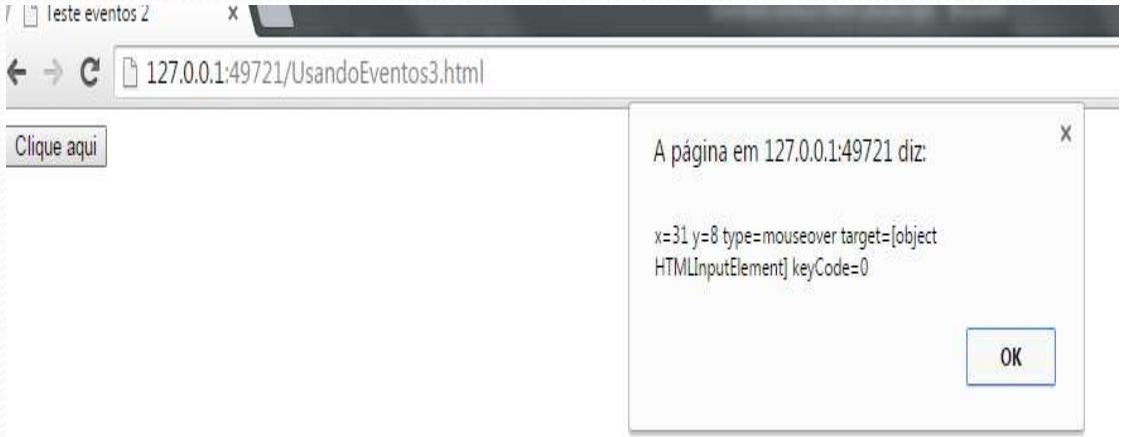
- `type` Tipo de evento que ocorreu, como *mouseover*.
- `target` Objeto de destino para o evento (como o documento ou um *link*).
- `which` Valor numérico que especifica o botão do mouse que foi clicado para eventos de *mouse* ou a tecla que foi pressionada para eventos de teclado.
- `modifiers` Lista 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 uma propriedade que fornece as coordenadas horizontais dentro da área do aplicativo do cliente em que o evento ocorreu (diferente das coordenadas dentro da página). Por exemplo, clicando no canto superior esquerdo da área do cliente sempre irá resultar em um evento de mouse com um valor x de 0, independentemente se a página foi rolada horizontalmente

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);
}
</script>
</head>

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

</html>
```



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

EXERCÍCIO - EVENTOS

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

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).

Referências

- ✓ CAELUM. Apostilas Cursos Gratuitas https://www.caelum.com.br/apostilas/. Acesso em: Jan. 2015.
- ✓ CODEACADEMY. Cursos Gratuitos. https://www.codecademy.com/pt Acesso em: Jan. 2015.
- ✓ DOM. Disponível em: http://www.w3schools.com/jsref/dom_obj_document.asp Acesso em: Jan.2015.
- ✓ EVENTOS. Disponível em: http://www.mspc.eng.br/info/jscriptContrEv.shtml Acesso em: Out.2016.
- ✓ HERANCA. Disponível em: http://www.devmedia.com.br/classes-no-javascript/23866. Acesso em: Jan.2015.
- ✓ HTMLCSS. Disponível em: http://del.icio.us/carlosbazilio/{css+html} Acesso Jan.2015.
- ✓ JAVASCRIPT_1. Disponível em: http://www.significados.com.br/javascript/ Acesso Jan.2015.
- ✓ JAVASCRIPT._2 Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/JavaScript_Vis%C3%A3o_Geral Acesso em: Jan.2015.
- ✓ OO. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Trabalhando_com_Objetos Acesso em:Jan.2015.
- ✓ OPERADORES. https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#Assignment_operators
- ✓ 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. Disponível em: https://msdn.microsoft.com/pt-br/library/z3ks45k7(v=vs.94).aspx Acesso em: Jan.2015.
- ✓ SILVA. Maurício Samy . JavaScript Guia do Programador. Editora Novatec.

Referências

- ✓ W3SCHOOLS. Disponível em: http://www.w3schools.com/ Acesso em: Jan.2016.
- ✓ W3. Disponível em: http://www.w3.org/ Acesso em: Jan.2016..