SI401

Programação para a Web

2º Semestre - 2024

Programação *Front-End*: *JavaScript*
Unidade 12

Prof. Guilherme Palermo Coelho

guilherme@ft.unicamp.br

Roteiro

- Objetos em *JavaScript*;
- *Strict Mode*;
- Referências.

OBJETOS EM *JAVASCRIPT*

Objetos em *JavaScript*

- Como visto anteriormente, *JavaScript* é uma linguagem ***orientada a objetos* (OO)**:
  - Da mesma forma que em outras linguagens OO, ***objetos*** em *JavaScript* são ***entidades*** que possuem ***atributos*** e ***métodos (propriedades)***:
    - ***Atributos***: utilizados para caracterizar um objeto;
      - São "***variáveis***" atribuídas ao objeto.
    - ***Métodos***: são ações que podem ser executadas em objetos;
      - São "***funções***" contidas em um objeto.

Objetos em *JavaScript*

- Em *JavaScript* praticamente tudo é interpretado como ***objeto***, inclusive os tipos primitivos de dados (exceto ***null*** e ***undefined***);
  - Há uma série de objetos predefinidos disponíveis para utilização;
  - O programador pode criar seus próprios objetos.
- Apesar de algumas novidades introduzidas com o padrão ECMAScript 2015, *JavaScript* ainda não possui todas as características observadas em outras linguagens orientadas a objetos.
  - **Ex.**: “Classes” são, na verdade, funções especiais.

Objetos em *JavaScript*

- **Atributos de Objetos**:
  - A forma mais simples de acesso aos ***métodos*** de um objeto é através da notação "***ponto***":
    - **"nome_objeto.nome_atributo"**;
  - **Ex.**: objeto "**myCar**" com três atributos ("**make**", "**model**" e "**year**"):

```
var myCar = new Object();
myCar.make = "Ford";
myCar.model = "Mustang";
myCar.year = 1969;
```

Objetos em *JavaScript*

- **Atributos de Objetos**:
  - Objetos em *JavaScript* também são chamados, em algumas situações, de ***arrays associativos***:
    - Cada atributo do objeto pode ser associado ao valor de uma ***string***;
    - O valor desta ***string*** (nome do atributo) pode ser usado para acessar o índice do ***array***;
    - **Ex.**:

```
myCar["make"] = "Ford";
myCar["model"] = "Mustang";
myCar["year"] = 1969;
```

Objetos em *JavaScript*

- **Atributo de Objetos**:
  - A utilização desta forma de acesso a atributos (notação "**colchete**") permite que o ***nome*** destes atributos seja qualquer ***string*** (com espaços em branco, início numérico, etc.):
    - Caso o identificador de um atributo **não** atenda às definições de *JavaScript* para nomes de variáveis, **não** será possível o acesso a esta propriedade pela **notação "ponto"**.
    - Esta interpretação de objetos como ***arrays associativos*** permite que se determine o nome de um atributo ***durante a execução*** do programa.

Objetos em *JavaScript*

- **Atributos de Objetos**:
  - **Ex.**:

Este tipo de acesso pode ser feito com variáveis que contenham qualquer valor que possa ser convertido para *string*.

```
var myObj = new Object(),
    str = "myString",
    rand = Math.random(),
    obj = new Object();

myObj.type                  = "Dot syntax";
myObj["date created"]       = "String with space";
myObj[str]                  = "String value";
myObj[rand]                 = "Random Number";
myObj[obj]                  = "Object";
myObj[""]                   = "Even an empty string";
```

Objetos em *JavaScript*

- **Criando objetos em *JavaScript***:
  - Existem formas diferentes de criar objetos em *JavaScript*:
    - Através de um ***instanciamento direto*** ou ***inicializadores de objeto***;
    - Através de ***funções construtoras***;
  - No ***instanciamento direto***, primeiramente é criada uma instância genérica de um objeto do tipo "**Object()**" e, em seguida, são atribuídas as propriedades a este novo objeto:

```
personObj=new Object();
personObj.firstname="John";
personObj.lastname="Doe";
personObj.age=50;
personObj.eyecolor="blue";
```

Objetos em *JavaScript*

- **Criando objetos em *JavaScript***:
  - Uma forma alternativa seria **já definir as propriedades** do novo objeto na sua própria criação:
    - Esta abordagem utiliza os chamados ***inicializadores de objeto***;

```
personObj={firstname:"John",lastname:"Doe",age:50,eyecolor:"blue"}
```

Objetos em *JavaScript*

- **Criando objetos em *JavaScript***:
  - No entanto, a forma mais comum (e mais parecida com outras linguagens) é através de ***funções construtoras***:
    - Estas funções definem o nome, a estrutura, as propriedades e os métodos de um objeto;
    - Uma instância do objeto é criada a partir do comando "**new**" seguido do nome da função construtora;

```
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;
}
```

```
var myFather=new person("John","Doe",50,"blue");
var myMother=new person("Sally","Rally",48,"green");
```

Objetos em *JavaScript*

- **Métodos**:
  - ***Métodos*** nada mais são que ***funções associadas*** a um objeto;
    - Em outras palavras, são ***propriedades*** de um objeto que são ***funções***;
  - Para adicionar um método a um objeto, deve-se:
    - Criar a função que será o método;
    - Atribuir esta função criada a uma determinada propriedade.

Objetos em *JavaScript*

- **Métodos**:

```
function person(firstname,lastname,age,eyecolor)
{
this.firstname=firstname;
this.lastname=lastname;
this.age=age;
this.eyecolor=eyecolor;

this.newlastname=newlastname;
}
```

```
function newlastname(new_lastname)
{
this.lastname=new_lastname;
}
```

```
myMother.newlastname("Doe");
```

Classes em *ECMAScript 2015*

```
class Veiculo {

        constructor(model, make, year) {// Construtor
                this._model = model;
                this._make = make;
                this._year = year;
        }

        get model() { // Getter para o atributo _model
                return this._model;
        }

        set model(model) { // Setter para o atributo _model
                this._model = model;
        }

}
```

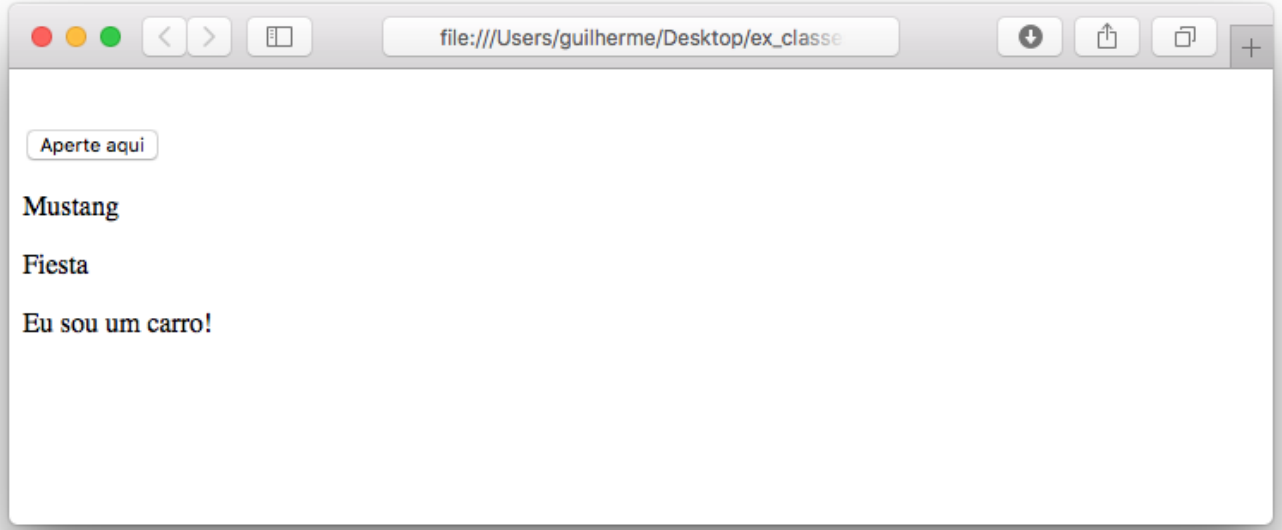
Classes em *ECMAScript 2015*

```
class Carro extends Veiculo{// Herança

        constructor(model, make, year) {
                super(model, make, year); //Construtor da superclasse
        }

        displayFabricante(){// Método
                return ”Fabricante: " + this._make;
        }

        static quemSouEu(){// Método Estático
                return “Eu sou um carro!”;
        }
}
```

Classes em *ECMAScript 2015*

```
<html lang="br">
<head>
     <script type="text/javascript" src="ex_classes.js"></script>
     <script>
          function testaClasse() {
               var myCar = new Carro("Mustang", "Ford", 1964);
               document.getElementById("modelAntes").innerHTML = myCar.model;
               myCar.model = "Fiesta";
               document.getElementById("modelDepois").innerHTML = myCar.model;
               document.getElementById("estatico").innerHTML = Carro.quemSouEu();
          }
     </script>
</head>
<body>
     <button type="button" onclick="testaClasse();">Aperte aqui</button>
     <p id="modelAntes"></p>
     <p id="modelDepois"></p>
     <p id="estatico"></p>
</body>
</html>
```

Classes em *ECMAScript 2015*



- **Mais informações**:

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Classes

Objetos em *JavaScript*

- **Enumerando todas as propriedades de um objeto**:
  - É possível percorrer todas as propriedades de um dado objeto com a instrução "***for...in***":
    - Sintaxe:

```
for (variavel in objeto)
{
    Código a ser executado;
}
```

Objetos em *JavaScript*

- **Enumerando todas as propriedades de um objeto**:

```
<html>
<body>
<script type="text/javascript">

var person={fname:"John",lname:"Doe",age:25};
var x;

for (x in person)
{
document.write(person[x] + " ");
}

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

John Doe 25

Objetos em *JavaScript*

- **Remoção de propriedades de um objeto**:
  - *JavaScript* também permite que uma determinada *propriedade* de um objeto seja ***removida***:
    - Para isso, usa-se a instrução ***delete***;

```
//Creates a new object, myobj, with two properties, a and b.
var myobj = new Object;
myobj.a = 5;
myobj.b = 12;

//Removes the a property, leaving myobj with only the b property.
delete myobj.a;
```

Objetos em *JavaScript*

- **Remoção de propriedades de um objeto**:
  - *Delete* também pode ser usada para eliminar variáveis globais que não tenham sido declaradas com a palavra-chave "**var**":

```
q = 17;
delete g;
```

  - ***Delete*** sempre retorna um valor *booleano*:
    - **True**: caso a remoção tenha ocorrido com sucesso;
    - **False**: caso contrário.

INSTRUÇÕES *TRY-CATCH* E *THROW*

Instruções *try-catch* e *throw*

- Como outras linguagens de programação, *JavaScript* também possui mecanismos para ***captura*** e ***tratamento*** adequado de erros:

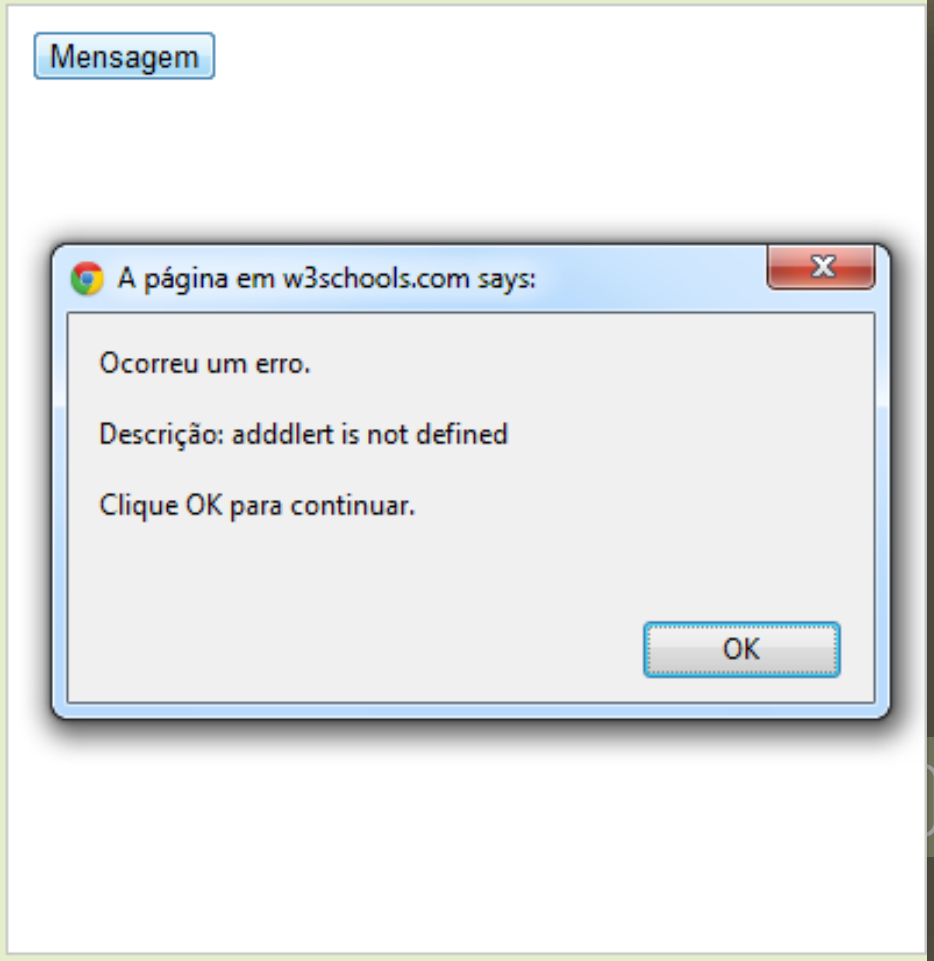
- A instrução *try-catch* permite que ***um bloco de código*** seja monitorado para detecção de erros:

  - Caso um erro seja detectado, o bloco de código associado a *catch* é executado.

```
try {
  //Código que deve ser executado
  }
catch(err) {
  //Código para tratamento de erros
  }
```

Instruções *try-catch* e *throw*

```
<html>
<head>
<script type="text/javascript">
var txt="";
function message()
{
try
  {
  adddlert("Olá!");
  }
catch(err)
  {
  txt="Ocorreu um erro.\n\n";
  txt+="Descrição: " + err.message + "\n\n";
  txt+="Clique OK para continuar.\n\n";
  alert(txt);
  }
}
</script>
</head>
<body>
<input type="button" value="Mensagem"
onclick="message()" />
</body>
</html>
```

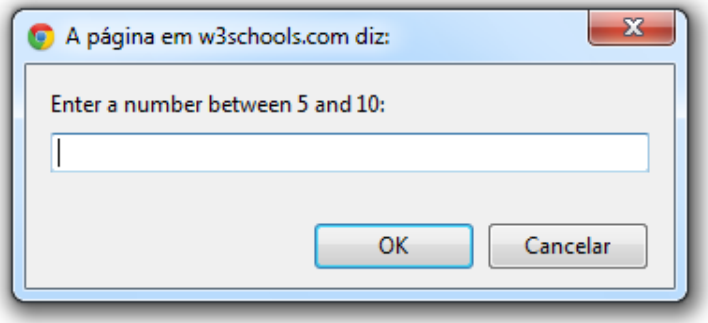


Instruções *try-catch* e *throw*

- Já a instrução *throw* é utilizada para ***criar uma exceção***:
  - Sintaxe: "**throw *excecao*;**"
    - ***excecao*** pode ser uma *string*, um *objeto*, valores *inteiros* ou *booleanos*;
    - Quando combinada com a instrução *try*...*catch*, a instrução *throw* permite um ***melhor controle*** do fluxo do programa e a ***geração de mensagens de erro*** mais precisas;

Instruções *try-catch* e *throw*

```
<html>
<body>
<script type="text/javascript">
var x=prompt("Enter a number between 5 and 10:","");
try
  {
  if(x>10) {
    throw "Err1";
    }
  else if(x<5) {
    throw "Err2";
    }
  else if(isNaN(x)) {
    throw "Err3";
    }
  }
catch(err)
  {
  if(err=="Err1") {
    document.write("Error! The value is too high.");
    }
  if(err=="Err2") {
    document.write("Error! The value is too low.");
    }
  if(err=="Err3") {
    document.write("Error! The value is not a number.");
    }
  }
</script>
</body>
</html>
```



Alguns Objetos Úteis

Alguns Objetos Úteis

- *JavaScript* define uma série de objetos que podem ser usados pelo programador.
- Dentre eles, podem ser destacados os objetos ***Array***, ***String***, ***Math*** e ***Date***;
  - Uma discussão sobre todos os objetos nativos na especificação da linguagem *JavaScript* pode ser encontrado em:

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects

http://w3schools.com/jsref/default.asp

Alguns Objetos Úteis

- **Array**:
  - *JavaScript* não possui um tipo dados explícito para arranjos de dados (vetores);
  - No entanto, pode-se utilizar o objeto pré-definido ***array*** para desempenhar esta tarefa;
    - Este objeto possui métodos e propriedades específicos para manipulação de arranjos:
      - Realização de tarefas tais como ***ordenação***, ***reversão***, ***concatenação***, etc.
      - Obtenção do comprimento do arranjo...

Alguns Objetos Úteis

- **Array**:
  - A criação de um ***array*** pode ser feita de três formas:
    - (1) Criando-se um ***array*** vazio e então atribuindo os elementos:

```
var myCars=new Array(); // regular array (add an optional integer
myCars[0]="Saab";       // argument to control array's size)
myCars[1]="Volvo";
myCars[2]="BMW";
```

    - (2) Forma condensada:

```
var myCars=new Array("Saab","Volvo","BMW"); // condensed array
```

    - (3) Forma literal (com inicializador de array):

```
var myCars=["Saab","Volvo","BMW"]; // literal array
```

Alguns Objetos Úteis

- **Array**:
  - Os elementos de um ***array*** podem ser objetos:
    - Inclusive ***outros arrays*** → ***arrays multidimensionais***;
    - Para acessar elementos de um ***array***, usa-se a notação:
      - "**nome_do_array[índice_elemento]**";
      - **Ex.**:

```
myCars[0]="Opel";
document.write(myCars[0]);
```

  - Mais informações http://w3schools.com/jsref/jsref_obj_array.asp

Alguns Objetos Úteis

- **String**:
  - É o objeto usado por *JavaScript* para armazenar trechos textuais;
  - Possui uma série de métodos e propriedades que facilitam a manipulação deste tipo de dados;
  - Sintaxe de criação de um objeto ***String***:

```
var txt = new String("conteudo_da_string");

ou

var txt = "conteúdo_da_string";
```

  - Descrição dos métodos e propriedades de ***String***:

http://w3schools.com/jsref/jsref_obj_string.asp

Alguns Objetos Úteis

- **Math**:
  - O objeto "**Math**" permite a realização de operações matemáticas:
    - Inclui vários métodos e constantes matemáticas;
    - Ex.:

```
var x=Math.PI;
var y=Math.sqrt(16);
```

  - **Atenção**: **Math** não possui construtor!
    - Métodos e atributos são estáticos;
  - Descrição das funções e constantes matemáticas providas por **Math**:

http://w3schools.com/jsref/jsref_obj_math.asp

Alguns Objetos Úteis

- **Date**:
  - Objetos do tipo "**Date**" são utilizados para manipular datas e horários;
  - Existem quatro formas para criação de um objeto deste tipo:

```
new Date() // current date and time
new Date(milliseconds) //milliseconds since 1970/01/01
new Date(dateString)
new Date(year, month, day, hours, minutes, seconds, milliseconds)
```

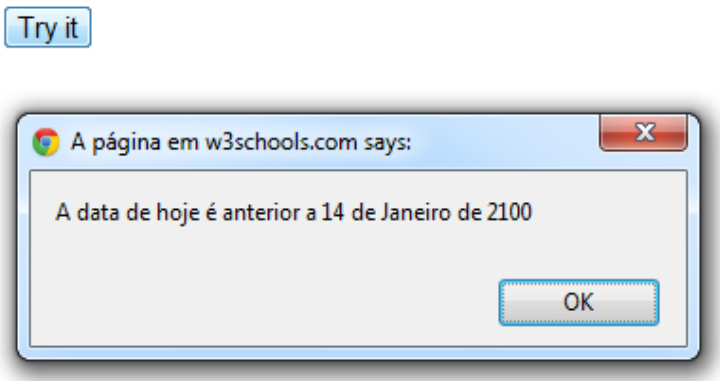
  - No primeiro caso, a variável que receber o novo objeto terá a ***data*** e o ***horário*** atuais do sistema;
  - No segundo caso, deve-se passar como parâmetro o número de milissegundos contados a partir de 01/01/1970;

Alguns Objetos Úteis

- **Date**:
  - No terceiro, uma ***string*** com a data desejada:
    - Ex.: "*Wed Mar 28 2012 16:20:34 GMT-0300 (E. South America Standard Time)*";
  - No último caso, passa-se cada campo da *data* e *horário* desejado (valores numéricos):
    - **Atenção**: o número indicativo do mês começa a ser contado em 0;
  - ***Date*** provê uma série de métodos para manipulação de datas e horários, tanto no fuso-horário local do navegador quanto no fuso-horário GMT;
  - Mais informações: http://w3schools.com/jsref/jsref_obj_date.asp

Alguns Objetos Úteis

```
<html>
<head>
<script type="text/javascript">
function myFunction()
{
   var x=new Date();
   x.setFullYear(2100,0,14);
   var today = new Date();

   if (x>today)
   {
     alert("A data de hoje é anterior a 14 de Janeiro
de 2100");
   }
   else
   {
     alert("A data de hoje é posterior a 14 de Janeiro
de 2100");
   }
}
</script>
</head>
<body>

<button onclick="myFunction()">Try it</button>

</body>
</html>
```



É possível comparar diretamente dois objetos Date!

JavaScript Strict Mode

JavaScript *Strict Mode*

- A partir da versão 1.8.5 (ECMAScript 5) de JavaScript, foi introduzida a opção de execução de *scripts* em modo "estrito" (*strict*);
- **Objetivo**: introduzir uma maior rigidez de sintaxe;
  - Torna alguns "erros silenciosos" em erros lançados (throw *SyntaxError*);
    - **Ex**.: declarar dois parâmetros com o mesmo nome em uma função;
  - Impõe algumas limitações a ações que geralmente estavam associadas a erros ou a más práticas de programação;
    - **Ex**.: atribuir um valor a uma variável não declarada (erro de digitação?)

JavaScript *Strict Mode*

- A partir da versão 1.8.5 (ECMAScript 5) de JavaScript, foi introduzida a opção de execução de *scripts* em modo "estrito" (*strict*);
- **Objetivo**: introduzir uma maior rigidez de sintaxe;
  - Abre caminho para futuras versões de JavaScript;
    - Estabelece uma lista de palavras reservadas que serão utilizadas em futuras versões da linguagem;
      - Ex.: *interface*, *package*, *private*, *protected*, *public*, etc.
- Mais informações:
  - https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Strict_mode
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode/Transitioning_to_strict_mode

JavaScript *Strict Mode*

- Sintaxe:
  - "use strict";
- Deve ser declarado:
  - No início do *script* (escopo global);
  - No início de uma função (escopo local);
- **Ex.:**

```
"use strict";
myFunction();

function myFunction() {
    y = 3.14;   // This will also cause an error because y is not declared
}
```

Exercício Sugerido 3

Exercício Sugerido 3

- Crie um documento HTML que contenha um formulário para preenchimento de 10 valores numéricos e um botão. Quando este botão for pressionado, um *script* deve ser chamado e os seguintes passos executados:
  - Um objeto chamado **meuVetor** deve ser criado, sendo que um dos atributos deste objeto deve ser um vetor composto pelos 10 valores preenchidos pelo usuário;
  - Este objeto **meuVetor** deve ser passado a um método *bubbleSort*() que implemente este algoritmo de ordenação para os elementos do vetor;
  - O método *bubbleSort*() deve alterar **meuVetor** (manter ordenação);
  - Os valores ordenados do vetor devem ser impressos na tela, seguidos da **data** e **hora** em que a ordenação foi feita.

Referências

Referências

- [1] W3Schools JavaScript Tutorial:

  http://www.w3schools.com/js/default.asp

- [2] Standard ECMA-262: ECMAScript Language Specification

  http://www.ecma-international.org/publications/standards/Ecma-262.htm

- [3] Mozilla Developer Network – JavaScript:

  http://developer.mozilla.org/pt-BR/JavaScript