

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Mini-curso de introdução às tecnologias JavaScript, jQuery e Bootstrap



```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;
```

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

Antes de falarmos de JavaScript, vamos falar sobre formulários em HTML?

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;}
```

Instituto de Computação
UFMT 2017

Futxicaia da Tecnológica
Projeto de Extensão

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Formulários e HTML

- A especificação da linguagem HTML inclui uma série de elementos destinados a construção de formulários para páginas e sistemas web.
 - Esse elementos são utilizados, principalmente, para o envio de dados/informações/arquivos/etc por parte dos usuários.
 - Tornando a utilização de formulários imprescindível em sistemas web.

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;}
```

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0;  
    length = elems.length,  
    callbackExpect = !invert;
```

Tag <form>

- A tag <form> é utilizada para descrever e delimitar a existência de um formulário no documento HTML. Sintaxe básica:

```
<form>  
    ...  
</form>
```

- Importante: permite-se vários formulário por página, porém, não é permitido anilhá-los, geralmente, considera-se apenas o primeiro.

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

Atributos

- Principais atributos:

- **id**: String com identificação única do elemento
- **action**: URI que define para “onde” será enviada (submetida) as informações do formulário
- **method**: Define se os dados serão enviados incorporados ao corpo do formulário (POST) ou à URL (GET).

Mais informações e atributos sobre formulários:
<https://developer.mozilla.org/docs/Web/HTML/Element/form>

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0;  
    if ( !elems.length )  
        return matches;  
    callbackExpect = !invert;
```

Componentes de um formulário

- Elementos que são tradicionalmente inseridos entre as tags <form>...</form> para permitir que o usuário entre com dados, a citar:
 - input [text | password | radio | checkbox | reset | submit | hidden]
 - textarea
 - select
 - option

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

Componentes de um formulário

- Continuando:

- button
- label
- fieldset
 - legend

```
.ui-helper-hidden-accessible {  
border: 0px;  
clip: rect(0, 0, 0, 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0px;  
position: absolute;  
width: 1px;}
```

Importante: Assim como nos demais elementos HTML, também é possível aplicar aos elementos de formulário uma grande gama de estilos utilizando CSS

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

Exemplo

```
<form id="frm-add-aluno" action="dados.html" method="get">
  <label for="txt-rga">RGA:</label>
  <input type="text" id="txt-rga" name="txt-rga" />
  <br />
  <label for="txt-nome-completo">Nome Completo:</label>
  <input type="text" id="txt-nome-completo" name="txt-nome-completo" />
  <br />
  <label for="psswd-senha-aluno">Senha</label>
  <input type="password" id="psswd-senha-aluno" name="psswd-senha-aluno" />
  <br />
  <span>Curso:</span>
  <br />
  <input type="radio" id="rd-curso-si" name="rd-curso" />&ampnbsp;
  <label for="rd-curso-si">Sistemas de Informação</label>
  <br />
  <input type="radio" id="rd-curso-co" name="rd-curso" />&ampnbsp;
  <label for="rd-curso-co">Ciência da Computação</label>
<br/><button type="submit">Cadastrar</button>
</form>
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Exemplo

- Resultado:

RGA: João Silva

Nome Completo: 200511316020

Senha: •••••

Curso:

- Sistemas de Informação
- Ciência da Computação

Cadastrar

Importante: assim como os demais elementos HTML, os elementos de formulário também podem ser estilizados utilizando com CSS da forma que quisermos (fonte, borda, display, margem, etc)

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;}
```

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

HTML5

- Elementos e atributos relacionados a formulários introduzidos no HTML5:
 - *Input type:*
 - **color, date, datetime, number, range, etc.**
 - **Atributos:**
 - *autocomplete, autofocus, formaction, list, multiple, placeholder, required, contenteditable, min, max, step, etc.*

.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px; - **datalist, keygen, output**
margin: -1px;
overflow: hidden;
padding: 0;}

O que é JavaScript?

- JavaScript (JS) é uma linguagem de programação interpretada, baseada em *script* e comumente utilizada em navegadores web (*client-side*).
 - É uma linguagem multi-paradigma, adotando diferentes paradigmas, entre esses, alguns aspectos de orientação a objetos. É fracamente tipada com tipagem dinâmica.
 - Proposta inicialmente para o navegador Netscape em 1995 e adotada parcialmente pelo Internet Explorer em 1996.

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

História

- 1994 – Lançamento do (Mosaic) Netscape
 - Era necessária uma linguagem que auxiliasse desenvolvedores
- Nome original: Mocha
 - Criada em 10 dias (!) por Brendan Eich, em Maio/95
 - Escolhido por Marc Andressen, fundador da NetScape
- Setembro/95
 - Livescript
- Dez/95

• Javascript (jogada de marketing!)

• Lançada junto com o Netscape 2.0

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

História

- 1996
 - Jscript (Microsoft) – versão do Javascript para IE
 - Problemas de compatibilidade
- 96-97
 - ECMA (European Computer Manufacturers Association)
 - Padrão ECMAScript
- 98 – ECMA2 , 99 – ECMA3, ECMA4 – Projeto Abandonado, 2009 – **ECMA5**, 2011 – **ECMA5.1**, 2015 – ECMA6, 2016 – **ECMA7 (ECMAScript 2016)**

.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;}

Mais história...

- Caiu em descrédito com os desenvolvedores por alguns anos principalmente pela incompatibilidade entre navegadores.
 - Alguns desenvolvedores alegavam também que era uma linguagem “para leigos”, dando origem a scripts “confusos/bagunçados” e pouco otimizados.
 - Sua popularidade foi restaurada com o suporte a requisições assíncronas (AJAX) e surgimento de bibliotecas JavaScript, a citar: prototype e jQuery

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        len = elems.length,  
        callbackExpect = !invert;
```

Inserindo no Navegador

- Os scripts JavaScript são inseridos nos documentos HTML utilizando a tag `<script>`
- Um script JS pode ser definido diretamente entre a tag, ex:

```
<script> alert('Olá'); </script>
```

- Ou ser oriundo de um arquivo externo, usualmente com a extensão .js, ex:

```
.ui-helper-hidden-accessible {  
border: 0;  
<script src="meu-script.js"></script>  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

Variáveis

- Variáveis

- Não precisamos dizer o tipo, exemplo:
- **var a = 123;**
- E a tipagem é dinâmica (podemos alterar o tipo da variável em tempo de execução):
- **a = “agora sou uma string”;**

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;}
```

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0;  
    length = elems.length,  
    callbackExpect = !invert;
```

Funções

- Funções
 - Usa-se a palavra reservada `function` e não é necessário “tipar” o retorno e os parâmetros, ex:

```
function soma(a,b) {  
    var c = a+b;  
    return c;  
}
```

- Podemos definir uma função "*dentro de uma variável*", e depois instanciá-la:

```
var A = function () {  
    console.log("Sou uma classe?");  
}  
  
b = new A();  
c = new A();
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Prototype

- Como vimos, JS é uma linguagem multi-paradigma cuja a orientação a objetos implementada é baseada em prototipação. Isso significa que podemos fazer alterações nas "classes" durante a execução e todas as instâncias recebem essas alterações, exemplo:

```
a.prototype.ola = function () {
    alert("Olá Mundo!");
}

c.ola();
```

.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0; position: absolute;
width: 1px;}

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

Arrays

- Usualmente, declaramos arrays em JS de duas formas:

```
var arr = ["a", "b", "c", "d"];
```

ou

```
var arr = new Array("a", "b", "c", "d");
```

- Os valores do array podem ser acessados utilizando: arr[0], arr[1], etc *ou* ainda arr.1, arr.2, etc
- Sendo "uma variável um array", podemos utilizar métodos como:

`push()`, `pop()`, `shift()`, `splice()`, etc

- E propriedades como `.length`

Outros métodos e propriedades:

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

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Objetos

- Objetos podem possuir métodos e propriedades
 - A forma recomendada de declarar objetos é:
 - `var obj = {nome : “João”, idade : “21”, curso : “SI”};`
 - Os valores de um objeto pode ser acessados utilizando:
 - `obj.nome`, `obj.idade`, `obj.curso`
 - ou
 - `obj['nome']`, `obj['idade']`, `obj['curso']`
 - Objetos aceitam que as “propriedades sejam funções” (métodos)

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

Objetos

```
var aluno = {
  nome : “João”,
  idade : “21”,
  curso : “SI”,
  correr : function () {
    if (this.idade <= 30) {
      console.log(obj.nome+“ Corre muito!”);
    }
    else {
      console.log(obj.nome+“ Corre pouco!”);
    }
  }
};
```

.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 } 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;

Manipulação do DOM

- O que é o DOM ?
 - *Document Object Model*
 - É a árvore dos elementos (objetos) renderizados (exibidos) pelo navegador.
 - Após o carregamento de um documento HTML o navegador gera o objeto *document* que contém todos os elementos da página.
- A linguagem JavaScript fornece um conjunto de métodos para a manipulação do DOM.

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0;  
    if( !callback ) {  
        callback = elements.length ?  
            identity : emptyFunction;  
    } else if( typeof callback !== 'function' ) {  
        callback = emptyFunction;  
    }  
    if( invert ) {  
        callback = matchNone;  
    } else {  
        callback = matchAll;  
    }  
    callbackInverse = callbackInverse || createCallbackInverse( callback );  
    callback = createCallback( callback );  
    callbackInverse( elems, callback, invert );  
    return matches;  
};
```

document.getElementById()

- O método `getElementById()` é utilizado para “pegarmos” (referenciarmos) um elemento no `document` (DOM) pelo seu atributo `id`, `<p id='paragrafo'>...</p>`, ex:
 - *document.getElementById('paragrafo');*
 - Era um dos principais métodos para manipulação do DOM antes do surgimento e popularização das bibliotecas JavaScript.

.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;}

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0;  
    if( !callback ) {  
        callback = elements.length ?  
            identity : emptyFunction;  
    }  
    callback = callback.inverse ?  
        callbackInverse : callback;  
    callbackExpect = !invert;  
    for( ; i < elems.length; i++ ) {  
        matches.push( callback( elems[ i ], invert ) );  
    }  
    return matches;  
};
```

document.getElementById()

- Após “pegarmos” um elemento, podemos fazer “qualquer coisa” com ele, por exemplo:
 - Alterar estilo
 - Acessar / atualizar atributos
 - Remover do DOM
 - etc

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0;  
    callback = callback || elements.length;  
    callbackInverse = !invert;  
    if ( typeof invert === 'function' ) {  
        callback = invert;  
        invert = undefined;  
    }  
    for ( ; i < elems.length; i++ ) {  
        if ( invert ) {  
            if ( !callback( elems[i] ) )  
                matches.push( elems[i] );  
        } else {  
            if ( callback( elems[i] ) )  
                matches.push( elems[i] );  
        }  
    }  
    return matches;  
};
```

document.getElementById()

- Exemplo:

```
var elem = document.getElementById('paragrafo');
```



A partir desse instante, a variável **elem** faz referência ao elemento com id “paragrafo” que se encontra no DOM do documento HTML.

Assim, utilizando o **elem** é possível, por exemplo, retornar, ou alterar o texto do elemento ou mesmo retornar ou alterar a borda do elemento.



```
elem.textContent;  
elem.style.backgroundColor="#006600";
```

ui
border-radius: 10px;
border: 1px solid black;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
Instituto de Computação
UFMT 2017
position: absolute;
width: 1px;

Manipulação de Atributos

- A linguagem JavaScript também fornece métodos para a manipulação de atributos dos elementos, entre eles:
 - `getAttribute("atributo")`
 - Retorna o valor do “atributo”, exemplo:
`elem.getAttribute("id");`
`elem.getAttribute("type");`
`elem.getAttribute("placeholder");`
 - Alguns atributos devem ser acessados diretamente, sem utilizar o método `getAttribute`, exemplo: `elem.value`; ou ainda `elem.value="novo valor"`;

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0;  
    if( !callback ) {  
        callback = elem.length,  
        callbackExpect = !invert;  
    }  
    for( ; i < elem.length; i++ ) {  
        if( invert ? elem[i] : !elem[i] ) {  
            matches.push( elem[i] );  
        }  
    }  
    if( !callback ) {  
        return matches;  
    }  
    callback( matches );  
}
```

Manipulação de Atributos

- Outros métodos importantes na manipulação de atributos:
 - `setAttribute("atributo", "valor");`
 - Altera o `valor` do `atributo`, caso o elemento ainda não possua o `atributo`, o `atributo` é inserido com o `valor`.
 - `removeAttribute("atributo");`
 - Remove do elemento o `atributo`.

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Coleções de Objetos

- Conforme vimos, o método `document.getElementById()` faz referência a um único objeto.
 - Porém, também é possível manusear coleções de objetos do DOM, utilizando, por exemplo:
 - O método `getElementsByName("tag")`
 - Retorna todos os elementos de uma determinada `tag`
 - E a propriedade `.children`

• Retorna os nós filhos de um elemento no DOM

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Coleções de Objetos

- Seja o seguinte HTML:

```
<p id="paragrafo-1">Lorem ipsum dolor sit amet</p>
<p class="paragrafo">consectetur adipiscing elit</p>
<p class="paragrafo">Nam fringilla felis et efficitur</p>
<p id="paragrafo-2">Nunc lobortis in eros sed</p>
```

Poderíamos utilizar o seguinte JavaScript para referenciar todos esses elementos:

```
var paragrafos =
    document.getElementsByTagName("p");
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Coleções de Objetos

- Seja o seguinte HTML:

```
<ul id="Lista">
    <li>Lorem ipsum dolor sit amet</li>
    <li>consectetur adipiscing elit</li>
    <li>Nam fringilla felis et efficitur</li>
    <li>Nunc lobortis in eros sed</li>
</ul>
```

Para referenciar todos os itens da lista `#Lista` podemos utilizar:

```
var itens =
    document.getElementById("lista").children;
```

ou ainda:

```
var itens =
    document.getElementById("lista").getElementsByTagName("li");
```

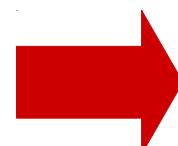
```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0;  
    length = elems.length,  
    callbackExpect = !invert;
```

Coleções de Objetos

- Em todos os casos apresentados, as variáveis *paragrafos* e *itens* são arrays e **não podem** ser “acessadas diretamente”, como fizemos antes:

```
paragrafos.textContent;
```

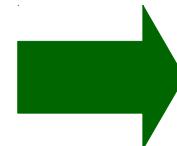
```
itens.style.backgroundColor="#006600";
```



ERRADO!

```
paragrafos[0].textContent;
```

```
itens[2].style.backgroundColor="#006600";
```



CORRETO!

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Coleções de Objetos

- Ou ainda, podemos iterar sobre o array:

```
for (i=0;i<paragrafos.length;i++) {
    console.log(paragrafos[i].textContent);
}
```

e

```
for (i=0;i<itens.length;i++) {
    console.log(itens[i].style.backgroundColor="#006600");
}
```

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Coleções de Objetos

- Outros métodos para nos referenciarmos a coleções de elementos do **DOM**:
 - ***getElementsByClassName('className')***: Retorna um array de objetos de acordo com o nome da classe passada como parâmetro
 - ***querySelector('seletor')***: Retorna o **primeiro** elemento correspondente ao seletor CSS passado como parâmetro.
 - ***querySelectorAll('seletor')***: Retorna um array com todos os elemento correspondente ao seletor CSS passado como parâmetro.

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

Eventos

- A linguagem JavaScript fornece um vasto conjunto de recursos para trabalharmos com **eventos**:
 - Os eventos geralmente estão relacionados a **interações do usuário**:
 - Ao clicar em um objeto, ao pressionar uma tecla do teclado, ao passar o mouse sobre um objeto, entre outros.
 - Ou ainda **relacionados ao documento HTML**:
 - Ao carregar página, ao sair da página, ao redimensionar a página, entre outros.

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0;  
    length = elems.length,  
    callbackExpect = !invert;
```

Eventos

- Alguns exemplos de eventos:
 - *onclick*: Disparado quando há um **click do mouse**
 - *onload*: Disparado quando a **página é carregada**
 - *onkeydown*: Disparado quando uma **tecla** do teclado é **pressionada**.
 - *onmouseover*: Disparado quando o **ponteiro do mouse passa sobre** um objeto.
 - *onfocus*: Disparado quando um **campo** de formulário **recebe o foco** (por exemplo o cursor é colocado em um campo de texto)
 - *onfocusout*: Disparado quando um **campo perde o foco**.

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

Eventos

- Alguns exemplos de eventos:
 - *oninput / onchange*: Disparado quando há uma **entrada / alteração de dados em um campo de formulário**.
 - *onsubmit*: Disparado quando um **formulário é submetido** (enviado).

Listas de eventos JavaScript:

http://www.w3schools.com/jsref/dom_obj_event.asp

<https://developer.mozilla.org/en-US/docs/Web/Events>

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        len = elems.length,  
        callbackExpect = !invert;
```

Manipulação de Eventos

- Podemos manipular eventos diretamente nos elementos HTML por meio de atributos:

```
<span id="span-1" onclick="alert('Fui clicado!');">  
    Lorem ipsum dolor sit amet  
</span>
```

```
<span id="span-2" onmouseover="alert('passaram o mouse!');">
```

Nunc lobortis in eros sed

```
</span>
```

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        elem = elems.length,
        callbackExpect = !invert;
```

Manipulação de Eventos

- Ou ainda, diretamente no JavaScript:

```
<script>
    document.getElementById("span-1")
        .addEventListener("click",function() { alert('fui clicado!'); });

```

```
</script>
```

```
<script>
    document.getElementById("span-2")
        .addEventListener("mouseover",function() { alert('passaram o mouse!'); });
</script>
```

```
.<span>-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;
```

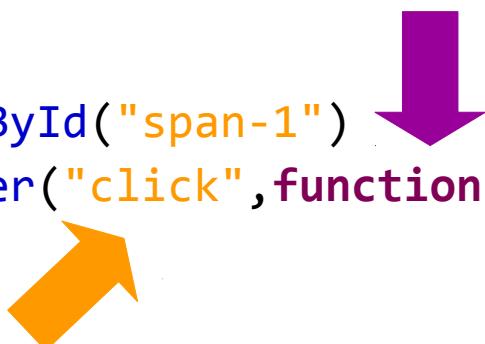
```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        len = elems.length,  
        callbackExpect = !invert;
```

Manipulação de Eventos

- Observe que:

Utiliza-se uma função anônima

```
document.getElementById("span-1")  
    .addEventListener("click",function() { alert('fui clicado!'); });
```



O prefixo **on** é descartado

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0; position: absolute;  
width: 1px;}
```

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        len = elems.length,  
        callbackExpect = !invert;
```

Manipulação de Eventos

- Para removermos eventos utilizamos o método `removeEventListener` exemplo:

```
document.getElementById("span-1")  
    .removeEventListener("click", funcaoClick);
```



Importante: não funciona para funções anônimas

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;}
```

**Instituto de Computação
UFMT 2017**

**Futxicaia da Tecnológica
Projeto de Extensão**

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

Por hora...

- JavaScript é uma linguagem completa, complexa e poderosa, vimos apenas o "topo do iceberg"
 - Temos muitas bibliotecas que valem a pena serem estudadas:
 - Underscorejs - <http://underscorejs.org/>
 - Backbonejs - <http://backbonejs.org/>
 - Requirejs - <http://requirejs.org/>
 - AngularJS (Angular 1.x) - <https://angularjs.org/>
 - **TypeScript ("nova versão do JavaScript em alto nível")**
 - Angular 2/4

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Bibliotecas JavaScript

- O que são as bibliotecas JS ?
- Por que desenvolver bibliotecas JS ?
- Quais as principais bibliotecas JS atualmente ?

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;}
```

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

O que são?

- **O que são?**

Conjunto de código em **JavaScript “Puro”** que fornecem soluções simples para a maioria das tarefas relacionadas a JS no dia-a-dia

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;}
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Por que desenvolvê-las ?

- **Por que desenvolvê-las ?**

- Para **facilitar a utilização do JS** na construção de *web sites* sistemas web “bonitos”, interativos, que utilizam AJAX, com maior interatividade e que aumentam a experiência do usuário:
 - Web 2.0

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;
```

Mas realmente facilitam?

- Desenvolver em JavaScript pode ser complicado, principalmente quando queremos que o código **funcione em todos os navegadores**
- E mesmo assim...
- A maioria do código JS desenvolvido é feito para trabalhar com um conjunto de tarefas específicas:
selecionar elementos, adicionar/modificar/esconder conteúdo/elementos, responder ações e requisições dos usuários (eventos) (McFarland, 2011)

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

Exemplo

- Escrever uma função JS para acessar os elementos que possuem uma determinada classe(class) e altere o conteúdo desses elementos

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;}
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Em JavaScript puro

```
1  function alterConteudoPelaClasse(classe_procurada,novo_conteudo) {
2      var elems = document.getElementsByTagName('*');
3      for (i in elems) {
4          if ((elems[i] != "") && (typeof elems[i] != "undefined")) {
5              var classes = elems[i].className;
6              if ((classes != "") && (typeof classes != "undefined")) {
7                  var classes_array = classes.split(" ");
8                  if (classes_array.length > 1) {
9                      for (j in classes_array) {
10                         if (classes_array[i]==classe_procurada) {
11                             elems[i].innerHTML = novo_conteudo;
12                             break;
13                         }
14                     }
15                 }
16             else {
17                 if (classes==classe_procurada)
18                     elems[i].innerHTML = novo_conteudo;
19             }
20         }
21     }
22 }
23 }
```

.ui
bor
cli
hei
mar
ove

Instituto de Computação
UFMT 2017

Futxicaia da Tecnológica
Projeto de Extensão

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Métodos Nativos

- Hoje em dia os principais navegadores já implementam métodos que facilitariam essa atividade:
 - `getElementsByName()`
 - Implementado no IE a partir da versão 9, e nos outros principais navegadores já a algum tempo.
 - `querySelector()`
 - `querySelectorAll()`

.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;}

Bibliotecas mais utilizadas

- Dos 1 milhão de websites mais acessados
 - **40.6%** não utilizam nenhuma biblioteca
 - Dos 59.4% restantes:
 - **53.4%** - **jQuery**
(Amazon.com, Wordpress.com, Microsoft.com, Tumblr.com, Pinterest.com, Imdb.com)
 - **5.0%** - **MooTools** (Babylon.com, 9gag.com, Joomla.org)
 - **3.7%** - **Prototype** (Apple.com)

Fonte: W³Techs (2012)

http://w3techs.com/technologies/overview/javascript_library/all

Bibliotecas mais utilizadas

- Dos 1 milhão de websites mais acessados:
 - **35.9%** não utilizam nenhuma biblioteca
 - Dos 64.1% restantes:
 - **60.4%** - **jQuery** (Amazon.com, Wordpress.com, Microsoft.com, Tumblr.com, Pinterest.com, Imdb.com)
 - **6.4%** - **Modernizr** (Hostgator.com, Nbcnews.com, Babylon.com)
 - **4.1%** - **MooTools** (Hp.com, Joomla.org)
 - **2.5%** **Prototype** (Buzzfeed.com, Steampowered.com)

– **Dojo: 0.1%** **AngularJS: 0.1%** **Ext JS: 0.1%**

Fonte: W3Techs (2014)

http://w3techs.com/technologies/overview/javascript_library/all

Bibliotecas mais utilizadas

- Dos 1 milhão de websites mais acessados:
 - **29.6%** não utilizam nenhuma biblioteca
 - Dos 70.4% restantes:
 - **67.4%** - **jQuery** (Yahoo.com, Amazon.com, Msn.com, Pinterest.com)
 - **10.0%** - **Bootstrap** (Americanexpress.com)
 - **9.6%** - **Modernizr** (Wordpress.com, Bbc.com, Forbes.com, Skype.com)
 - **3.8%** - **MooTools** (Hp.com)
 - **2.2%** **Prototype**

Fonte: W3Techs (2015)

http://w3techs.com/technologies/overview/javascript_library/all

Bibliotecas mais utilizadas

- Dos 1 milhão de websites mais acessados:
 - **26.4%** não utilizam nenhuma biblioteca
 - Dos 70.4% restantes:
 - **70.8%** - **jQuery** (Yahoo.com, Amazon.com, Msn.com, Pinterest.com)
 - **13.3%** - **Bootstrap**
 - **10.7%** - **Modernizr**
 - **3.4%** - **MooTools** (Hp.com)
 - **2.0%** - **Prototype**
 - **0.4%** - **Angular**

Fonte: W3Techs (2016)

http://w3techs.com/technologies/overview/javascript_library/all

Bibliotecas mais utilizadas

- Dos 1 milhão de websites mais acessados:
 - **24.5% não utilizam** nenhuma biblioteca
 - Dos 75.5% restantes:
 - **72.6%** - **jQuery** ([Yahoo.com](#), [Amazon.com](#), [Msn.com](#), [Pinterest.com](#))
 - **15.5%** - **Bootstrap***
 - **10.8%** - **Modernizr***
 - **2.9%** - **MooTools**
 - **1.7%** - **Prototype**
 - **0.4%** - **Angular**

Fonte: W³Techs (2017)

http://w3techs.com/technologies/overview/javascript_library/all

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        elemLength = elems.length,
        callbackExpect = !invert;
```

Exemplo utilizando bibliotecas:

- **jQuery**

```
function alteraConteudoPelaClasse(classe_procurada,novo_conteudo)
    $('. '+classe_procurada).html(novo_conteudo);
```

- **MooTools**

```
function alteraConteudoPelaClasse(classe_procurada,novo_conteudo)
    $$('. '+classe_procurada).set('html', novo_conteudo);
```

- **Prototype**

```
function alteraConteudoPelaClasse(classe_procurada,novo_conteudo) {
    $$('. '+classe_procurada).each(function(e) {
        .ui-helper-ei.update(novo_bconteudo);
        border: });
    clip:{ rect(0 0 0 0);
    height: 1px;
    margin: -1px;
    overflow: hidden;
    padding: 0;
    position: absolute;
    width: 1px;
    }
}
```

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

jQuery

- Atualmente é a biblioteca JS mais utilizada na construção de sites; Ampla aceitação boa documentação
- Além de seletores CSS3, tem suporte a:
 - Eventos, Efeitos, Manipulação de conteúdo, entre outros
- Possui “zilhões” de *plug-ins* por exemplo:
 - Validação e máscaras para formulários
 - Galerias de imagens, janelas modais, sliders, etc
- Possui versão para dispositivos móveis (jQuery Mobile)
- Possui um projeto que disponibiliza um conjunto de função de interface – jQuery UI



```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

Instalação

- Bastar baixar pelo site: <https://jquery.com/download/>
- E inserir arquivo *jquery.js* em nossas páginas/documentsos HTML
- Exemplo: `<script src="js/jquery.min.js"></script>`



O arquivo .min é a versão Minify ou compactada da biblioteca, com tamanho menor

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;}
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Instalação (CDN)

- Ou ainda, utilizar CDN (Content Delivery Network), como o Hosted Libraries da Google:

<https://developers.google.com/speed/libraries/>

```
<script
src="https://ajax.googleapis.com/ajax/Libs/jquery/2.2.0/jquery
.min.js"></script>
```

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;
```

**Instituto de Computação
UFMT 2017**

**Futxicaia da Tecnológica
Projeto de Extensão**

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

Sintaxe básica

- Apesar de outras formas de utilizar, a sintaxe básica a biblioteca *jQuery* é:

jQuery aceita praticamente todos os seletores do CSS3



$\$(seletores/obj).metodo(parametros);$



Método a ser aplicado em todos os elementos relacionados ao seletor



Alguns métodos, alteram/modificam algo nos elementos quando passamos parâmetros ou simplesmente retornam valores caso nenhum parâmetro seja passado

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;}
```

Instituto de Computação
UFMT 2017

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

Exemplos

- O método .css é utilizado para alterar estilos (style) do (conjunto de) elemento(s):

```
$( "li" ).css( 'background-color', '#edb8b8' );
```

```
$( "li.fundo" ).css( 'background-color', '#c0edb8' );
```

```
$( "li#item" ).css( 'background-color', '#f0ecbe' );
```

- Pode-se definir várias propriedades de uma única vez:

```
$( "li#item" ).css(
  { 'border-width' : '2px',
    'border-color' : '#000',
    'border-radius' : '4px' }
);
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Métodos Encadeados

- Vimos exemplos de chamadas de métodos únicos, porém, jQuery permite o “encadeamento” de métodos, exemplo:

```
$("#li#item")
  .css('background-color', '#ffebeb').hide(200).show(200)
  .css({'border-width': '4px', 'border-color': '#f00'});
```

Isso é possível porque a cada método retorna a instância do jQuery

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;}
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Callback

- A grande maioria dos métodos em jQuery permitem a definição de funções de *callback*, exemplo:

```
$('div#bloco').hide(800,
    function () {
        $('span#status').html('Escondido!');
        $this.show(800,
            function () {
                $('span#status').html('Mostrado');
            }
        );
    }
);
.ui-helper-hidden-accessible {
```

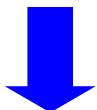
Callback functions: funções de *callback* são trechos de código associados a outro trecho de código que são “chamados” (executados) em determinados momentos propícios, em geral quando a execução do código principal é finalizada

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

Seletores

- jQuery aceita os seletores tradicionais de CSS, incluindo seletores do CSS1, CSS2, CSS3, exemplo:

Seletor CSS2



```
$( 'input[type=radio]:checked' ).hide(600);
```



Seletor CSS3

```
.ui-helper-hidden-accessible {  
border: 0;  
clip: rect(0 0 0 0);  
height: 1px;  
margin: -1px;  
overflow: hidden;  
padding: 0;  
position: absolute;  
width: 1px;}
```

Instituto de Computação
UFMT 2017

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

Seletores

- Também fornece métodos auxiliares para trabalharmos com a seleção de elementos ou “*filtering*”:

`.eq(índice)`

Reduz o conjunto de elementos a um índice específico

`.filter(seletor)`

Reduz o conjunto de elementos aos elementos que casam com o seletor passado.

`.first()`

Reduz o conjunto de elementos ao primeiro elemento.

.ui-helper-hidden-accessible
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Seletores

...continuando:

.has(seletor)

Reduz o conjunto de elementos a elementos que possuem filhos que casam com o seletor passado

.is(seletor|elemento|objeto)

Reduz o conjunto de elementos a elementos que casam com o seletor, elemento ou objeto passado. **Importante:** retorna um booleano (**true** ou **false**)

.not(seletor|elemento|objeto)

Remove os elementos que casam com o seletor, elemento ou objeto passado

```
.ui-helper-hidden-accessible {
    border: 0;
    clip: rect(0 0 0 0);
    height: 1px;
    margin: -1px;
    overflow: hidden;
    padding: 0;
    position: absolute;
    width: 1px;
```

Outros métodos de *filtering*:

<https://api.jquery.com/category/traversing/filtering/>

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Iteração

- Obviamente o **jQuery itera sobre um conjunto** de elementos quando por exemplo alteramos a borda de todos os `<p>` do nosso documento. Porém essa iteração ocorre **de forma implícita** para nós. Todavia, a biblioteca jQuery fornece um método para quando precisamos fazer iterações de forma explícita, o método **`.each()`**

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;}
```

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0,  
        length = elems.length,  
        callbackExpect = !invert;
```

.each()

- Exemplo:

É possível utilizar uma parâmetro para visualizarmos a posição na iteração

```
$("p").each(  
    function (index) {  
        console.log("Pos: "+index  
            +" id: "+$(this).attr("id")  
            +" conteúdo: "+$(this).html());  
    }  
);
```

Importante: O método `console.log()` é utilizado para imprimirmos informações no console do navegador, que geralmente pode ser aberto com a tecla F12.

Importante: `index` é apenas uma *alias*, podendo ser alterado para qualquer outro nome, exemplo, `i`, `in`, `indice`, etc.

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      length = elems.length,
      callbackExpect = !invert;
```

Eventos

- Conforme vimos, eventos são um importante aspecto da linguagem JavaScript e a biblioteca jQuery trabalha de forma eficiente com evento, conforme sintaxe básica:

```
$( "seletores" ).on( "evento", funcãoAserExecutada );
```

- Também é possível utilizar função anônima:

```
$( "seletores" ).on( "evento", function() {
  alert( "Alerta!" );
});
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Exemplos

```
$( "p.texto" ).on( "mouseover",
    function () {
        $(this).css( 'border', '1px solid #F00' );
    }
);

$( "input[type=button]" ).on( "click",
    function () {
        alert("Click!");
    }
);
```

```
.ui-helper-hidden-accessible {
border: 0;    $("seletores").on("evento",function() {
clip: rect(0 0 0 0); alert("Alerta!");
height: 1px;
margin: -1px; });
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        length = elems.length,
        callbackExpect = !invert;
```

Métodos Importantes

- A biblioteca jQuery também fornece outros métodos relacionados a importantes aspectos da manipulação do **DOM**:
 - Atributos
 - Classes
 - Visibilidade dos elementos

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        len = elems.length,
        callbackExpect = !invert;
```

Manipulação de Atributos

- jQuery fornece (entre outros) os seguintes métodos para manipularmos atributos:

.attr(atributo)

Retorna o valor de um determinado **atributo**, exemplo: **.attr("id")**;

.attr(atributo, valor)

Altera o valor de determinado **atributo** para **valor**, exemplo: **.attr("id", "novo_id")**;

.removeAttr(atributo)

Remove determinado **atributo** de um elemento, exemplo:
.removeAttr("checked");

.val()

Retorna o valor do atributo **value** do elemento

.val(valor)

Altera o atributo **value** de um elemento para um **valor**

```
grep: function( elems, callback, invert ) {
  var callbackInverse,
      matches = [],
      i = 0,
      elem = elems[ i ];
  if( !callback ) {
    callback = invert;
    invert = undefined;
  }
  for( ; i < elems.length; i++ ) {
    elem = elems[ i ];
    if( elem.nodeType === 1 && ( elem.className + " " ).indexOf( " " + invert + " " ) > -1 ) {
      matches.push( elem );
    }
  }
  return matches;
}
```

Manipulação de Classes (class)

- Em diversos momentos ao manipularmos o DOM, é necessário trabalharmos com o atributo **class**, para verificarmos por exemplo se um elemento possui determinada classe ou ainda alterar a classe dos elementos. Pensando nessa atividades rotineiras, jQuery há na jQuery os métodos que veremos a seguir:

```
.ui-helper-hidden-accessible {
  border: 0;
  clip: rect(0 0 0 0);
  height: 1px;
  margin: -1px;
  overflow: hidden;
  padding: 0;
  position: absolute;
  width: 1px;
```

```
grep: function( elems, callback, invert ) {  
    var callbackInverse,  
        matches = [],  
        i = 0;  
    if ( !elems.length )  
        return matches;  
    callbackInverse = !invert;  
    for ( ; i < elems.length ; i++ )  
        matches.push( callback( elems[ i ], invert ) );  
    return matches;  
};
```

Manipulação de Classes (class)

.addClass(classes)

Adiciona uma ou mais classes a um elemento ou conjunto de elementos, exemplo: `.addClass("classe-um classe-dois");`

.removeClass(classes)

Remove uma, várias, ou todas (caso seja utilizado sem nenhum parâmetro) as classes de um elemento ou conjunto de elementos

.hasClass(classe)

Verifica se o elemento (ou conjunto de) possui determinada classe retornando o booleano `true` caso possua e `false` caso não possua, exemplo: `.hasClass("destaque");`

.toggleClass(classe)

Altera uma ou mais classes de um elemento ou conjunto de elementos, ou seja, caso o elemento já possua a classe, ela é removida, caso contrário, ela é adicionada

Manipulação de Atributos

- **Importante:** o próprio método `.attr()` pode ser utilizado para a manipulação dos atributos `value` e `class`. Todavia, como esses são métodos bastante utilizados, a jQuery oferece métodos específicos para esses atributos.

Métodos relacionados a manipulação de atributos:

<https://api.jquery.com/category/attributes/>

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        g = elems.length,
        callbackInvert = !invert;
```

Visibilidade de Elementos

- Assim como a manipulação de classes e outros atributos, manipular a visibilidade de elementos (mostrar, esconder, alternar) também é uma atividade corriqueira a qual jQuery oferece novos métodos no intuito de facilitar a atividade e tornar e auxiliar na melhor experiência dos usuários. Veremos alguns desses métodos a seguir:

```
.ui-helper-hidden-accessible {
border: 0;
clip: rect(0 0 0 0);
height: 1px;
margin: -1px;
overflow: hidden;
padding: 0;
position: absolute;
width: 1px;
```

```
grep: function( elems, callback, invert ) {
    var callbackInverse,
        matches = [],
        i = 0,
        g = elems.length,
        callbackExpect = !invert;
```

Visibilidade de Elementos

.show(parâmetros)

Exibe um elemento ou um conjunto de elementos. Pode ser chamada sem passarmos nenhum parâmetro ou passando vários parâmetros como por exemplo o tempo em milisegundos, exemplo: `$("p").show(900);`

.hide(parâmetros)

Método análogo ao **.show()**, porém utilizado para esconder elementos.

.toggle(classe)

Caso o elemento esteja escondido ele será mostrado e caso esteja sendo mostrado, ele será escondido

```
.ui-helper-hidden-accessible {
    border: 0;
    clip: rect(0 0 0 0);
    height: 1px;
    margin: -1px;
    overflow: hidden;
    padding: 0;
    position: absolute;
    width: 1px;
```

Visibilidade de Elementos

`.hide()`, `show()` e `.toggle()` são métodos básicos da jQuery para exibirmos e escondermos elementos, todavia, há métodos com animações específicas, a citar:

- `.fadeIn()`
- `.fadeOut()`
- `.fadeToggle()`
- `.slideUp()`
- `.slideDown()`
- `.slideToggle()`

Outros métodos relacionados à aplicação de efeitos em elementos:

<https://api.jquery.com/category/effects/>