

Alteração de Atributos

■ *classList*

- exemplo de aplicação de classes pré definidas muito utilizado para controlar de forma dinâmica a aplicação de formatação (seletores de class CSS)
 - *contains(); add(); remove()*

```
<h1>Grocery List</h1>
<ul>
  <li id="one" class="vegetables">Onions</li>
  <li id="two" class="vegetables">Garlic</li>
  <li id="three" class="Veg1 Veg2 Veg3 Veg4">Cabage</li>
</ul>

<script>
  var element=document.querySelector("#three");
  if (element.classList.contains('promo'))
    {element.classList.remove('promo');}
  else
    {element.classList.add('promo');}
</script>
```

```
<style>
  li{list-style-type: none}
  .promo{
    background-color:lightblue;
    color:white;
  }
</style>
```

Grocery List

Onions
Garlic
Cabage

Alteração de Atributos

■ *className*

- Exemplo:

```
<style>
  div{background-color: lightblue;
    width:30%;
    text-align: center;}
  .cl1{font-size: 2em;}
  .cl2{color:white}
</style>
</head>
<body>
  <div class='cl1' id='d1'>className Example</div>
```

className Example

- propriedade permite definir o valor da class, neste caso substitui o valor anterior

```
<script>
  document.getElementById('d1').className='cl2';
```

className Example

■ *className*

- apesar de menos frequente é possível utilizar a propriedade *className* para adicionar valores ao atributo class sem substituir o anterior
 - passa por utilizar uma atribuição composta com o novo valor (string) iniciar por espaço

```
<script>  
    document.getElementById('d1').className += ' c12';
```

className Example

Element Object

Seleção de Elementos (traversing properties)

■ Traversing Properties

- `previousSibling`
- `nextSibling`
- `firstChild`
- `lastChild`
- `parentNode`
- ...

```
<script>
  var element, elBef, elNxt;
  element=document.getElementById('two');

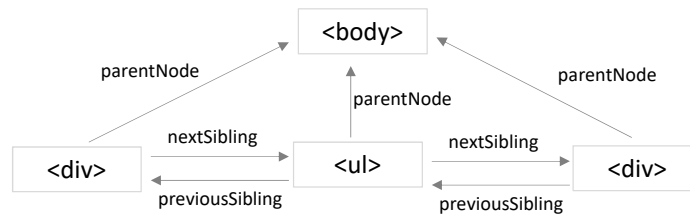
  elBef=element.previousSibling;
  elBef.className='promo';

  elNxt=element.nextSibling;
  elNxt.className='promo';
</script>
```



```
<!DOCTYPE html>
<html>

<head>
  <title>My page</title>
</head>
<body>
  <div>Header</div>
  <ul>
    <li>List</li>
  </ul>
  <div>Footer</div>
</body>
</html>
```



Window Object

HTML DOM Reference

The references describe the properties and methods of each object, along with examples.

| | | | | |
|---------------|-------------|----------|----------------|----------|
| Attributes | Console | Document | Element | Events |
| Event Objects | Geolocation | History | HTMLCollection | Location |
| Navigator | Screen | Style | <u>Window</u> | Storage |

window Object

- Representa uma janela aberta no browser

| Método | Descrição |
|--------------------------------|-----------------------------|
| <code>window.open()</code> | abre nova janela |
| <code>window.close()</code> | fecha a janela atual |
| <code>window.moveTo()</code> | movimenta a janela atual |
| <code>window.resizeTo()</code> | redimensiona a janela atual |
| <code>window.print()</code> | imprime a página atual |
| ... | |

- métodos adicionais do window object (sem referência explícita ao objeto)

| Método | Descrição |
|----------------------------|----------------------------------------------------------------------|
| <code>alert()</code> | gera uma caixa de alerta |
| <code>confirm()</code> | gera uma caixa de confirmação |
| <code>prompt()</code> | permite a inserção de valores por parte do utilizador |
| <code>setInterval()</code> | chama repetidamente uma função com base num intervalo constante (ms) |
| <code>setTimeout()</code> | chama uma função após um determinado tempo (ms) |
| ... | ... |

window Object

■ Propriedades:

| Propriedade | Descrição |
|---------------------------------|------------------------------------|
| <code>window.innerHeight</code> | altura da janela do browser em px |
| <code>window.innerWidth</code> | largura da janela do browser em px |

| | |
|-------------------------------|--------------------------------------------------|
| <u><code>window</code></u> | <i>current browser window (top level object)</i> |
| <u><code>history</code></u> | <i>pages in browser history</i> |
| <u><code>location</code></u> | <i>URL of current page</i> |
| <u><code>navigator</code></u> | <i>information about the browser</i> |
| <u><code>screen</code></u> | <i>device's display information</i> |

window Object

■ Funções de Temporização

■ Particularmente interessantes :

- Permitem controlar o tempo de chamada de uma outra função
 - Muito úteis por exemplo na implementação de *slideshows*, ...
- `setTimeout(nomeFunção, tempo(ms));`
 - chama a função após um período de tempo definido em ms
 - função pode ser definida diretamente através de uma *anonymous function*

```
var randomNumber;  
function alteraValor() {  
    randomNumber = Math.random();  
    document.getElementById('global').innerHTML = randomNumber;  
}  
  
setTimeout(alteraValor, 2000);  
</script>
```



2 seg

0.7736800073180348

- *setInterval (nomeFunção, tempo(ms));*
 - chama uma função de forma repetida a intervalos regulares definidos em ms
 - função pode ser definida diretamente através de uma *anonymous function*

```
<span id="global"></span> <br />

<script>
  var randomNumber;
  function alteraValor() {
    randomNumber = Math.random();
    document.getElementById('global').innerHTML = randomNumber;
  }
  setInterval(alteraValor, 2000);
</script>
```

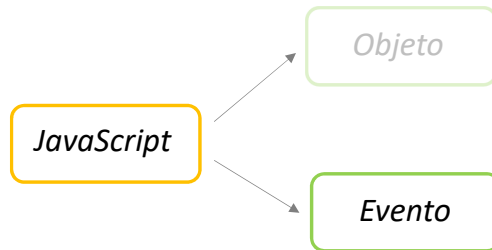
0.063333285407163203  2 seg 0.9507259053643793  2 seg 0.3472738463897258

- *clearInterval()*
 - Este método elimina um *timer* que tenha sido estabelecido com *setInterval()*. Para tal o *timer* deve estar armazenado numa variável que será o parâmetro do *clearInterval()*

```
var myVar = setInterval(function(){ myTimer() }, 1000);
...
function myStopFunction() {
  clearInterval(myVar);
}
```

Eventos

Eventos



| Evento | É disparado... |
|-----------|---------------------------------------------------------------------------|
| click | quando é pressionado e liberado o botão primário do mouse, trackpad, etc. |
| mousemove | sempre que o cursor do mouse se move. |
| mouseover | quando o cursor do mouse é movido para sobre algum elemento. |
| mouseout | quando o cursor do mouse se move para fora dos limites de um elemento. |
| dblclick | quando acontece um clique duplo com o mouse, trackpad, etc. |

Eventos

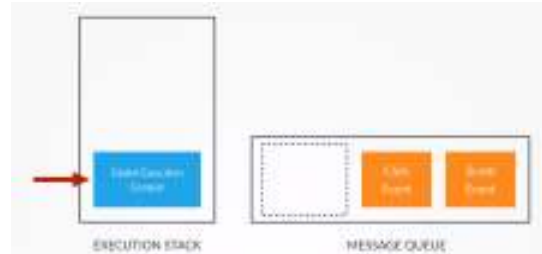
- É necessário um identificador (*event handler*) que permita iniciar (*trigger*) a execução do *script* quando esse evento ocorre
 - Existem diferentes tipos de eventos:
 - *User Interface Events*
 - *HTML elements Events*
 - *Mouse events*
 - *Keyboard events*
 - *Mutation events*
 - *HTML5 events*
 - ...

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

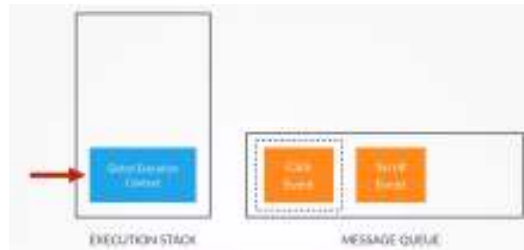
Processamento de Eventos



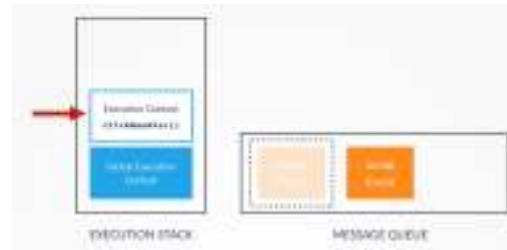
1. Enquanto funções estiverem a ser executadas, os eventos não podem ser considerados;



2. Uma vez libertados os contextos de execução (terminada a execução das funções), os eventos podem ser considerados



3. Quando um determinado evento ocorre é processado na *message queue*



4. É então criado o seu próprio contexto de execução (processamento/função associado à deteção do evento)

<https://www.udemy.com/the-complete-javascript-course/learn/v4/t/lecture/5869158?start=0>

Eventos

■ Event handling

1. Seleção do elemento HTML onde é definido o evento
 - Os eventos relacionados com o *Window Object* não se referem à DOM tree.
2. Definir o evento que faz o *trigger* da ação pretendida
 - Alguns eventos podem ser aplicados à maioria dos elementos
 - Exemplo: *mouseover*, ...
 - Outros eventos apenas podem ser definidos em elementos específicos
 - Exemplo: *submit*, ...
3. Definir o processamento
 - Código necessário para executar uma determinada ação

Eventos

■ Três formas distintas de declarar um evento:

1. HTML Event Handlers

- Usa diretamente atributos dos elementos HTML para responder a eventos no elemento onde foram declarados.
- Considerada **má prática** uma vez que associa de forma direta o HTML e o JavaScript
 - Deve existir uma separação clara na aplicação das duas tecnologias

```
<a onclick="hide()"> ... </a>
```

2. DOM Event Handlers

- Considerados melhores que a opção anterior uma vez que permitem uma separação clara entre *Javascript* e HTML
- A limitação principal é que permite associar apenas uma função por evento e não permite a passagem de valores à função

```
element.onevent=functionName;
```

3. DOM Event Listeners

- sintaxe diferente
- Permitem associar múltiplas funções ao mesmo evento e passagem de argumentos

```
element.addEventListener('event',functionName,[Boolean]);
```

Eventos

■ HTML Event Handler

- **Não permite** uma separação clara entre o HTML e o *Javascript*

```
<form method="post" action="#">
  <label for="username">username:</label>
  <input type="text" id="username" onblur="checkUsername()" />

  <div id="feedback"></div><br />

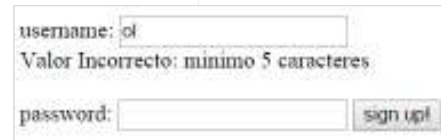
  <label for="password">password:</label>
  <input type="password" id="password" />

  <input type="submit" value="sign up!" />
</form>

<script>
  function checkUsername() {
    var elMsg = document.getElementById('feedback');
    var elUsername = document.getElementById('username');

    if (elUsername.value.length < 5) {
      elMsg.textContent = 'Valor Incorrecto: minimo 5 caracteres';
    } else { elMsg.textContent = '';}
  }
</script>
```

Quando o utilizador selecciona outro campo é disparado o evento



Testa o número de caracteres

Eventos

■ DOM Event handler

- Permite uma separação clara entre o HTML e o Javascript (o evento é definido no <script>)

element.ontevent=functionName

```
<form method="post" action="#">
  <label for="username">username:</label>
  <input type="text" id="username" />

  <div id="feedback"></div><br />

  <label for="password">password:</label>
  <input type="password" id="password" />

  <input type="submit" value="sign up!" />
</form>

<script>
  function checkUsername() {
    var elMsg = document.getElementById('feedback');
    var elUsername = document.getElementById('username');

    if (elUsername.value.length < 5) {
      elMsg.textContent = 'Valor Incorrecto: minimo 5 caracteres';
    } else { elMsg.textContent = ''; }
  }
  var elUser = document.getElementById("username");
  elUser.onblur = checkUsername;
</script>
```

Evento não é definido no HTML



Evento definido no JS (não permite a passagem de parâmetros, apenas o nome da função)

Eventos

■ Event Listener

- Podem invocar mais do que uma função mas não são suportados pelos browsers mais antigos

element.addEventListener('event',functionName,[Boolean])

```
<form method="post" action="#">
  <label for="username">username:</label>
  <input type="text" id="username" />

  <div id="feedback"></div><br />

  <label for="password">password:</label>
  <input type="password" id="password" />

  <input type="submit" value="sign up!" />
</form>

<script>
  function checkUsername() {
    var elMsg = document.getElementById('feedback');
    var elUsername = document.getElementById('username');

    if (elUsername.value.length < 5) {
      elMsg.textContent = 'Valor Incorrecto: minimo 5 caracteres';
    } else { elMsg.textContent = ''; }
  }

  var elUser = document.getElementById("username");
  elUser.addEventListener('blur', checkUsername, false);
</script>
```

Evento não é definido no HTML

* Ao contrário das duas situações anteriores o evento não é precedido de 'on'

** apenas o nome da função, os parêntesis são omitidos.

*** Quando existem eventos encadeados define a ordem dos eventos. Valor por defeito false (bubling)

■ Passagem de Valores a uma Função

■ *Event handlers & Listeners*

- Solução passa por definir uma **anonymous function** tal que permita receber argumentos
- A *anonymous function* pode conter várias chamadas a funções diferentes.

```
<script>
  function checkUsername(minlength) {
    var elMsg = document.getElementById('feedback');
    var elUsername = document.getElementById('username');

    if (elUsername.value.length < minlength) {
      elMsg.textContent = 'Valor Incorrecto: minimo 5 caracteres';
    } else { elMsg.textContent = ''; }
  }

  var elUser = document.getElementById("username");
  elUser.addEventListener('blur', function(){ checkUsername(5);} , false);
</script>
```

Com base numa **anonymous function** é passado o valor 5 à função *checkUsername*

Event Object

HTML DOM Reference

The references describe the properties and methods of each object, along with examples.

| | | | | |
|---------------|-------------|----------|----------------|---------------|
| Attributes | Console | Document | Element | <u>Events</u> |
| Event Objects | Geolocation | History | HTMLCollection | Location |
| Navigator | Screen | Style | Window | Storage |

Event Object

- Quando ocorre um evento, é criado automaticamente um **event object**
 - Guarda informação sobre o evento (tipo, elemento onde ocorreu o evento, ...)
 - É passado automaticamente a qualquer função que seja **event handler ou listener**
 - A função é definida geralmente com o parâmetro *e* (*event*)

| Property | Description |
|----------------------------|-----------------------------------------------------------------------------------------|
| bubbles | Returns whether or not a specific event is a bubbling event |
| cancelable | Returns whether or not an event can have its default action prevented |
| target | Returns the element that triggered the event |
| timeStamp | Returns the time (in milliseconds relative to the epoch) at which the event was created |
| type | Returns the name of the event |
| view | Returns a reference to the Window object where the event occurred |

| Method | Description |
|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| preventDefault() | Cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur |
| stopImmediatePropagation() | Prevents other listeners of the same event from being called |
| stopPropagation() | Prevents further propagation of an event during event flow |

Event Object

- O *event object* é passado de forma automática à função que é disparada pelo evento
 - tipicamente identifica-se por (e) (não é obrigatório!)
 - permite utilizar as propriedades e métodos do *event object* no interior da função

```
<form method="post" action="#">
<label for="username">username:</label>
<input type="text" id="username" />

<div id="feedback"></div><br />

<label for="password">password:</label>
<input type="password" id="password" />

<input type="submit" value="sign up!" />
</form>
```

A screenshot of a web form. It has a label 'username:' followed by a text input field. Below it is a label 'password:' followed by a password input field. To the right of the password field is a button labeled 'sign up!'.

Event Object

- Função executada **não possui** parâmetros de entrada

```
<div id="evObj"></div>

<script>
  function checkUsername(e) {
    var elTargetUser = e.target;
    document.getElementById('evObj').innerHTML = elTargetUser;
  }

  var elUser = document.getElementById("username");
  elUser.addEventListener('blur', checkUsername, false);
</script>
```

*Propriedades e
métodos disponíveis
no event (ex: target)*

A screenshot of the web form from the previous slide. The 'username' input field is now selected, and a tooltip is visible below it that says '[object HTMLInputElement]'. The 'password' field and 'sign up!' button are also visible.

*Evento disparado, o event
object é passado à
função*

Event Object

- Função executada **possui** parâmetros de entrada

```
<div id="evObj"></div>

<script>
  function checkUsername(e, minLength) {
    var elTarget = e.target;
    document.getElementById('evObj').innerHTML = elTarget;
  }
  var elUser = document.getElementById('username');
  elUser.addEventListener('blur', function (eObj){checkUsername(eObj, 5);}, false);
</script>
```

username:

password:

[object HTMLInputElement]

Tipos de Eventos

Tipos de Eventos

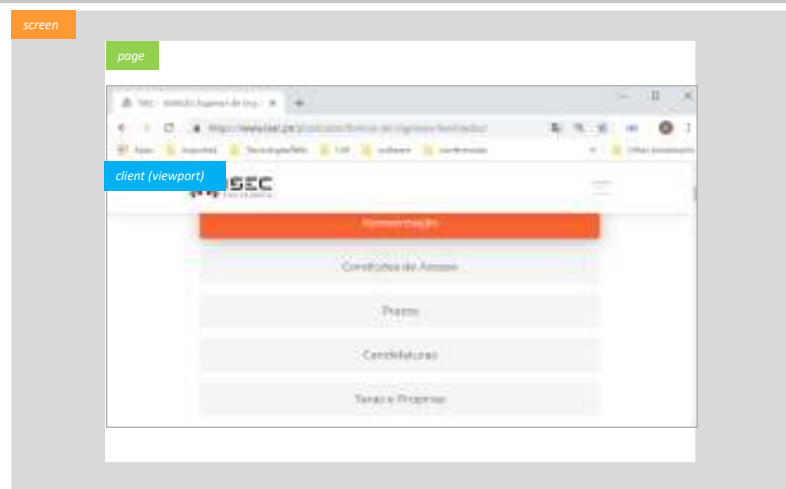
■ Mouse Events

- Dispara quando o:

| event | trigger |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Click</i> | utilizador clica no botão esquerdo do rato. Também dispara quando o utilizador pressiona a tecla <i>Enter</i> num elemento com <i>focus</i> |
| <i>Dbclick</i> | utilizador faz um <i>double click</i> no botão esquerdo do rato |
| <i>mousedown</i> | utilizador pressiona qualquer botão do rato |
| <i>mouseup</i> | utilizador liberta qualquer botão do rato (não pode ser iniciado por teclado) |
| <i>mouseover</i> | cursor do rato é sobreposto a um determinado elemento |
| <i>mouseout</i> | cursor do rato está sobre um elemento e se move para outro elemento (externo ao elemento inicial / não é filho do elemento inicial) |
| <i>mousemove</i> | cursor é movido em torno de um elemento (este evento é disparado de forma repetida) |

Mouse Events

Exemplo:
Propriedades específicas
mouse events



screen

screenX e **screenY** indicam a posição relativamente à totalidade do ecrã (top left corner)

page

pageX e **pageY** indicam a posição do cursor relativamente à página. O topo da página pode estar fora do viewport, ou seja mesmo com o cursor na mesma posição as coordenadas podem ser diferentes

client

clientX e **clientY** indicam a posição do cursor relativamente ao viewport. O scroll da página não afeta estas coordenadas

Mouse Events (propriedades específicas)

```
<ul id="posicao">
  <li id="sx"></li>
  <li id="sy"></li>
  <li id="px"></li>
  <li id="py"></li>
  <li id="cx"></li>
  <li id="cy"></li>
</ul>

<script>
  var sx = document.getElementById('sx');
  var sy = document.getElementById('sy');
  var px = document.getElementById('px');
  var py = document.getElementById('py');
  var cx = document.getElementById('cx');
  var cy = document.getElementById('cy');

  function mostraPosicao() {
    sx.innerHTML = event.screenX;
    sy.innerHTML = event.screenY;
    px.innerHTML = event.pageX;
    py.innerHTML = event.pageY;
    cx.innerHTML = event.clientX;
    cy.innerHTML = event.clientY;
  }

  var elEvento = document.getElementById('posicao');
  elEvento.addEventListener('mousemove', mostraPosicao, false);
</script>
```

Posição do cursor
do rato

- 109
- 153
- 87
- 54
- 87
- 54

Tipos de Eventos

■ User Interface Events

■ Dispara:

| event | trigger |
|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| load | quando é finalizado o download da página. Também pode ser aplicado a outros elementos em que é necessário o download, como <i>images</i> , <i>scripts</i> , <i>objects</i> |
| unload | antes de ser feito o unloading da página (deve funcionar como ajuda ao utilizador e não como incentivo para permanecer na página) |
| error | quando o browser encontra um erro JavaScript |
| resize | quando a janela do browser foi redimensionada |
| scroll | quando o user faz um scroll (down/up) da página |

User Interface Events

■ Exemplo: load event

```
<form method="post" action="#">
  <label for="username">username:</label>
  <input type="text" id="username" />

  <div id="feedback"></div><br />

  <label for="password">password:</label>
  <input type="password" id="password" />

  <input type="submit" value="sign up!" />
</form>
```

```
<script>
```

```
  function loadEvent() {
    var elFocus = document.getElementById('password');
    elFocus.focus();
  }
  window.addEventListener('load', loadEvent, false);
</script>
```

Evento 'load'

Tipos de Eventos

■ Keyboard Events

- Dispara quando o:

| event | trigger |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------|
| input | conteúdo de um elemento input ou textarea é alterado |
| keydown | utilizador pressiona qualquer tecla do teclado. Se o utilizador mantiver a tecla pressionada o evento dispara de forma repetida. |
| keypress | utilizador pressiona qualquer tecla que resultaria num caracter visível no ecrã. |
| keyup | utilizador liberta uma tecla do teclado. Este evento, ao contrário de keydown e keypress, é disparado após a representação do caracter. |

KeyboardEvent Object

| Property | Description |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| altKey | Returns whether the "ALT" key was pressed when the key event was triggered |
| ctrlKey | Returns whether the "CTRL" key was pressed when the key event was triggered |
| charCode | Returns the Unicode character code of the key that triggered the onkeypress event |
| key | Returns the key value of the key represented by the event |
| keyCode | Returns the Unicode character code of the key that triggered the onkeypress event, or the Unicode key code of the key that triggered the onkeydown or onkeyup event |
| location | Returns the location of a key on the keyboard or device |
| metaKey | Returns whether the "meta" key was pressed when the key event was triggered |
| shiftKey | Returns whether the "SHIFT" key was pressed when the key event was triggered |
| which | Returns the Unicode character code of the key that triggered the onkeypress event, or the Unicode key code of the key that triggered the onkeydown or onkeyup event |

Tipos de Eventos

```
<form action="#">
  <textarea rows="5" cols="20" id="message"></textarea>
</form>
<div id="charleft"></div>
<div id="lastkey"></div>
```

```
<script>
  function contaCaracteres(e) {

    var textEnter = document.getElementById('message').value;
    counter = (180 - (textEnter.length));

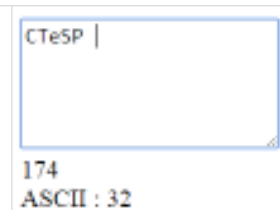
    var charDisplay = document.getElementById('charleft');
    charDisplay.textContent = counter;

    var lastkey = document.getElementById('lastkey');
    lastkey.textContent = 'ASCII : ' + e.keyCode;

  }

  var elTextarea = document.getElementById('message');
  elTextarea.addEventListener('keypress', contaCaracteres, false);
</script>
```

CTeSP |



174
ASCII : 32

- 2) atualiza o contador
- 3) mostra o nº de caracteres em falta
- 4) código ASCII ultima tecla
- 1) disparado repetidamente

Tipos de Eventos

■ Form Events

| event | Trigger |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>submit</i> | dispara no nó que representa o elemento <i>form</i> , geralmente utilizado para verificar os valores introduzidos antes de enviar ao servidor |
| <i>change</i> | dispara quando uma alteração é efetuada em alguns elementos do <i>form</i> (ex: seleção num menu <i>drop-down</i> , seleção de um <i>radio button</i> , seleção de uma <i>check box</i> , ...) |
| <i>input</i> | dispara quando são introduzidos valores nos elementos <i>input</i> e/ou <i>textarea</i> |
| <i>blur</i> | dispara quando se abandona um determinado elemento. Não é exclusivo dos <i>forms</i> (pode por exemplo ser utilizado com <i>links</i>). |
| <i>focus</i> | dispara quando se seleciona um determinado elemento. Não é exclusivo dos <i>forms</i> (pode por exemplo ser utilizado com <i>links</i>). |

Tipos de Eventos

■ Mutation Events

- Sempre que a estrutura da *DOM tree* é alterada (elementos adicionados ou removidos) é disparado um *mutation event*

| event | Trigger |
|------------------------------------|---------------------------------------------------------------------------------------|
| <i>DOMNodeInserted</i> | dispara quando um nó é inserido na DOM Tree |
| <i>DOMNodeRemoved</i> | dispara quando um nó é removido da DOM Tree |
| <i>DOMSubtreeModified</i> | dispara quando a estrutura da DOM Tree é alterada. |
| <i>DOMNodeInsertedIntoDocument</i> | dispara quando um nó é inserido na DOM Tree como descendente de outro nó já existente |
| <i>DOMNodeRemovedFromDocument</i> | dispara quando um nó é removido da DOM Tree como descendente de outro nó já existente |

Mutation Events

```
<h2>Lista de Compras <span id="counter">1</span></h2>
<ul id="list">
  <li>morangos</li>
</ul>
<div><a href="#">Novo list item</a></div>

<script>
  var elList = document.getElementById('list');
  var elLink = document.getElementsByTagName('a')[0];
  var elCounter = document.getElementById('counter');

  function adicionaElemento() {

    var elNewtext = document.createTextNode('café');
    var elNew = document.createElement('li');

    elNew.appendChild(elNewtext);
    elList.appendChild(elNew);

    function atualizaContador() {
      var elDim = document.getElementsByTagName('li').length;
      elCounter.innerHTML = elDim;
    }

    elLink.addEventListener('click', adicionaElemento, false);
    elList.addEventListener('DOMNodeInserted', atualizaContador, false);
  }
</script>
```

Lista de Compras 1

- morangos

[Novo list item](#)

Lista de Compras 2

- morangos
- café

[Novo list item](#)

é disparado a cada alteração

Tipos de Eventos

- **HTML5 events**
 - Eventos disponibilizados pela última versão do HTML

| Event | Trigger |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>DOMContentLoaded</i> | dispara no window object quando a DOM Tree está formada (efetuado download dos elementos HTML), ou seja o script é executado antes de ter sido efetuado o download de todos os outros elementos (css, imagens, ...). |
| <i>beforeunload</i> | dispara no window object antes de se efetuar o unload da página (deve constituir uma ajuda ao utilizador, por exemplo informar o utilizador de que as alterações a um formulário não foram gravadas). |