



Material complementar

JAVASCRIPT

Trabalhando com eventos

www.treinaweb.com.br

SUMÁRIO

Tratando eventos	3
Definições importantes	3
Arquivo de testes	5
Eventos Legado	6
Eventos de formulário	6
Eventos de janela	6
Eventos de mouse	6
Eventos de teclado	7
Registrando rotinas de tratamento	8
Configurando propriedades de tratamento de evento	8
Configurando atributos de rotina de tratamento de eventos	10
addEventListener()	11
removeEventListener()	13
Chamada de rotina de tratamento de evento	14
Argumentos de rotina de tratamento de evento	14
Contexto de rotina de tratamento de evento	14
Escopo de rotina de tratamento de evento	14
Valor de retorno de rotina de tratamento	15
Ordem de chamada	15
Cancelamento de evento	15
Eventos de carga de documento	16
Eventos de mouse	20
Eventos de roda de mouse	23
Eventos de arrastar e soltar	27
Eventos de texto	32
Eventos de teclado	36

Tratando eventos

Se a intenção é adicionar funcionalidades interessantes em uma página da web, apenas ser capaz de inspecionar ou modificar o documento não é suficiente, você não acha? É preciso ser capaz também de detectar o que o usuário está fazendo, e responder a ele. É para isso que existem os manipuladores de eventos. Teclas pressionadas são eventos, cliques do mouse são eventos, mesmo o movimento do mouse pode ser visto como uma série de eventos.

Os programas JavaScript usam um modelo de programação dirigido por eventos assíncronos. Nesse estilo de programação, o navegador Web gera um evento em que algo interessante acontece no documento, no navegador ou em algum elemento ou objeto associado a ele. Por exemplo, o navegador Web gera um evento quando termina de carregar um documento, ou quando o usuário coloca o cursor do mouse sobre um hiperlink, ou ainda quando pressiona uma tecla no teclado.

Se um aplicativo JavaScript se interessa por determinado evento em especial, pode registrar uma ou mais funções para serem chamadas quando ocorrerem eventos desse tipo. Note que isso não é exclusividade de programação Web: todos os aplicativos com interfaces gráficas com o usuário são projetados dessa maneira – não fazem nada até que algo aconteça e, assim, respondem.

A maneira como o navegador trabalha com eventos é, fundamentalmente, muito simples. É possível registrar manipuladores para tipos específicos de eventos e para elementos específicos do documento. Sempre que um evento ocorre, o manipulador para esse evento, se houver, é chamado.

Para alguns eventos, como o pressionamento de teclas, saber apenas que o evento ocorreu não é o suficiente, pois pode-se querer saber qual tecla foi pressionada. Para armazenar essas informações, cada evento cria um objeto de evento que o manipulador pode verificar.

É importante perceber que, mesmo que os eventos possam ser disparados a qualquer momento, não há dois manipuladores que podem ser executados no mesmo momento. Se um código JavaScript ainda está em execução, o navegador aguardará até que ele termine antes de chamar o próximo manipulador. Isso também vale para o código que é disparado de outras maneiras, como com `setTimeout`.

Definições importantes

Para entender o que compõe os eventos, conheça algumas definições:

1. **Evento:** A palavra evento, por si só, não é um termo técnico que exija definição. Os eventos são ocorrências a respeito das quais o seu programa será notificado pelo navegador Web.
2. **Tipo de Evento:** É uma string que especifica o evento ocorrido. O tipo “mousemove”, por exemplo, significa que o usuário moveu o mouse e assim por diante. Como o tipo de um evento é apenas uma string, às vezes ele é chamado de nome do evento e, de fato, usa-se esse nome para identificar o tipo de evento específico sobre o qual se está falando.
3. **Alvo do Evento:** É o objeto no qual o evento ocorreu ou ao qual está associado. Ao falar de um evento deve-se especificar tanto o tipo como o alvo. Um evento load em um objeto Window, por exemplo. Os objetos Window, Document e Element são os alvos de eventos mais comuns.
4. **Rotina de tratamento do evento:** É uma função que manipula ou responde a um evento.
5. **Objeto do evento:** é um objeto associado a um evento em especial e contém detalhes sobre esse evento.

6. **Propagação do evento:** É o processo por meio do qual o navegador decide em quais objetos disparam rotinas de tratamento de eventos.
7. **Ações-padrão do evento:** Quando ocorre um evento click em um hiperlink, por exemplo, a ação padrão é o navegador seguir o link e carregar uma nova página. As rotinas de tratamento de evento podem impedir essa ação padrão, retornando um valor apropriado, chamando um método do objeto evento ou configurando uma propriedade desse objeto. Às vezes, isso se chama 'cancelar' o evento.

Conhecendo essas definições fica mais fácil aprofundar o tratamento de eventos em JavaScript.

Arquivo de testes

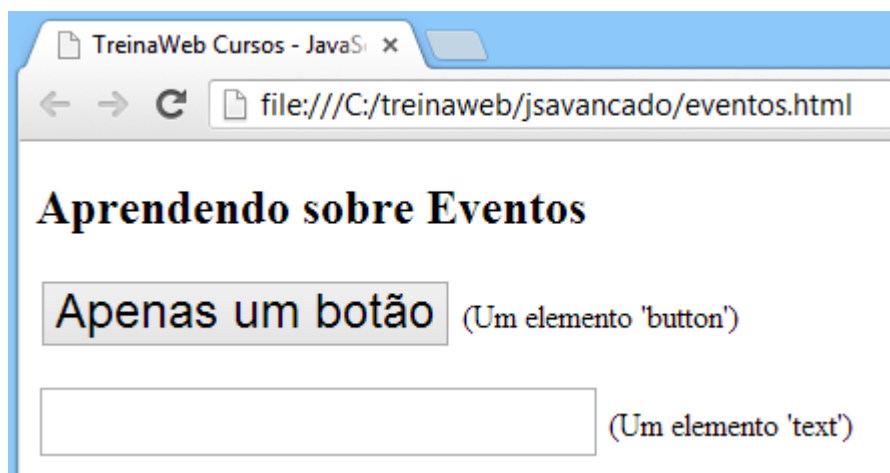
Para trabalhar com os eventos você precisa criar os arquivos em que os testes serão efetuados e poder executar os exemplos propostos.

Crie um arquivo com o nome **eventos.html** e insira o seguinte código:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <script type="text/javascript" src="eventos.js"></script>
    <title>TreinaWeb Cursos - JavaScript</title>
  </head>
  <body>
    <h2>Aprendendo sobre Eventos </h2>
    <p> <button type="button" id="botao" style="font-size: 150%;">Apenas um
botão</button> (Um elemento 'button') </p>
    <p> <input type="text" id="texto" style="font-size: 150%;"> (Um elemento 'text') </p>
  </body>
</html>
```

Salve o arquivo e crie um segundo arquivo com o nome **eventos.js**. No momento não será inserido nenhum código nele, mas será utilizado no decorrer dos estudos de eventos em JavaScript.

Abra o arquivo **eventos.html** no navegador. O resultado será:



Eventos Legado

Os eventos mais utilizados em aplicativos Web geralmente são os que existem há mais tempo e que são universalmente suportados: eventos para lidar com mouse, teclado, formulários HTML e objetos Window.

Este conteúdo, de eventos mais comuns de JavaScript, será tratado de forma bem prática e rápida. Serão utilizados alguns conhecimentos avançados da linguagem, mas também mostrado um pouco da base para facilitar o aprendizado de novos conhecimentos demonstrados no decorrer do curso.

Eventos podem ser divididos em quatro partes: Eventos de formulário, eventos de janela, eventos de mouse e eventos de teclado. Acompanhe, a seguir, breve descrição de cada um deles.

Eventos de formulário

Os formulário e hiperlinks foram os primeiros elementos de uma página Web a aceitar scripts, remontando aos primórdios da Web e de JavaScript.

Os elementos <form> disparam eventos submit quando o formulário é enviado, e eventos reset quando o formulário é redefinido. Elementos de formulário do tipo botão (inclusive botão de opção e caixas de seleção) disparam eventos click quando o usuário interage com eles, e os elementos de formulário que mantêm algum tipo de estado geralmente disparam eventos change quando o usuário muda o estado deles.

Relembrando: eventos primitivos não são aprofundados aqui por se tratar de pré-requisito para este curso. Porém, alguns conceitos serão lembrados e retomados para facilitar o aprendizado do conteúdo mais avançado.

Eventos de janela

Os eventos de janela representam ocorrências relacionadas à própria janela do navegador, ao invés de qualquer conteúdo específico do documento exibido dentro da janela.

Dentre os eventos de janela o mais importante é o evento load. Ele é disparado quando um documento e todos os seus recursos externos (como imagens, por exemplo) são totalmente carregados e exibidos para o usuário.

Eventos de mouse

Os eventos de mouse são gerados quando o usuário move o mouse e dá um clique em um documento. Por exemplo, o evento mousemove é disparado sempre que o usuário move ou arrasta o mouse.

Os eventos mousedown e mouseup são disparados quando o usuário pressiona e solta o botão do mouse.

Depois de uma sequência de movimentos mousedown e mouseup, o navegador dispara um evento click.

Se o usuário clicar em um botão do mouse duas vezes sucessivamente, o segundo evento click será seguido por um evento dblclick.

Esses são alguns dos eventos de mouse. Adiante será tratado um pouco mais sobre o assunto.

Eventos de teclado

Quando o navegador Web tem foco no teclado, gera eventos sempre que o usuário pressiona ou solta uma tecla no teclado. Entretanto, os atalhos de teclado que têm significado para o sistema operacional ou para o próprio navegador são frequentemente “consumidos” pelo sistema operacional, ou pelo navegador, e podem não ser visíveis para as rotinas de tratamento de eventos de JavaScript.

Os eventos de teclado são disparados no elemento do documento que tem o foco de teclado. Se nenhum elemento tiver foco, os eventos são disparados diretamente no documento.

Os eventos de teclado `keydown` e `keyup` são de baixo nível; ou seja, eles são disparados quando uma tecla é pressionada ou solta.

Quando um evento `keydown` gera um caractere imprimível, um evento `keypress` adicional é disparado depois de `keydown`, mas antes de `keyup`.

`keypress` é um evento de texto de nível mais alto, e seu objeto evento especifica o caractere gerado e não a tecla pressionada.

As rotinas de tratamento de evento de teclado recebem um objeto evento com um campo `keyCode` que especifica a tecla pressionada ou solta.

Além de `keyCode`, o objeto evento, de eventos de tecla, também tem `altKey`, `ctrlKey`, `metaKey` e `shiftKey`, que descrevem o estado das teclas modificadoras do teclado.

Registrando rotinas de tratamento

Existem duas maneiras básicas de registrar rotinas de tratamento de evento: A primeira, dos primórdios da Web, é configurar uma propriedade no objeto ou elemento do documento-alvo do evento. A segunda técnica, mais recente e mais geral, é passar a rotina de tratamento para um método ou elemento do objeto.

Para complicar as coisas, existem duas versões de cada técnica: Você pode configurar uma propriedade de tratamento de evento em código JavaScript; ou, para elementos do documento, configurar o atributo correspondente diretamente em HTML.

Para registro de rotina de tratamento, por meio de chamada de método, existe o método-padrão `addEventListener()` suportado por todos os navegadores, exceto o IE 8 e anteriores; e um método diferente, chamado `attachEvent()`, para todas as versões do IE anteriores ao IE9. Por ser um método específico para esse navegador e suas versões, não será abordado no curso.

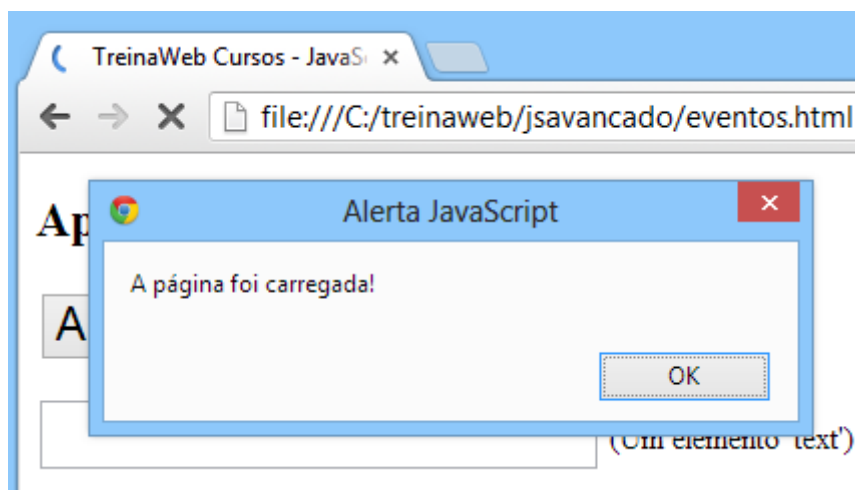
Configurando propriedades de tratamento de evento

O modo mais simples de registrar uma rotina de tratamento de evento é configurar uma propriedade do alvo do evento na função de tratamento de evento desejada. Por convenção, as propriedades de rotina de tratamento de eventos têm nomes que começam com o prefixo **on** seguido pelo nome de evento: `onclick`, `onchange`, `onload`, `onmouseover` etc. Esses nomes de propriedade diferenciam letras maiúsculas e minúsculas e são escritos com todas as letras minúsculas, mesmo quando o tipo de evento consistir em várias palavras.

Acompanhe o exemplo para entender melhor. Abra o arquivo **eventos.js** e insira este código:

```
// Configura a propriedade onload do objeto Window em uma função.  
// A função é a rotina de tratamento de evento: ela é chamada quando o  
// documento é carregado.  
window.onload = function() {  
  
    // Exibe um alerta  
    alert("A página foi carregada!");  
};
```

Salve o arquivo e abra a página **eventos.html** no navegador. O resultado será:



Essa técnica de registro de rotina de tratamento de evento funciona em todos os navegadores e para todos os tipos de evento normalmente usados. A desvantagem das propriedades de rotina de tratamento de evento é que são projetadas com a suposição de que os alvos de evento terão, no máximo, uma rotina de tratamento para cada tipo de evento. Ou seja, ele modifica ou sobrescreve qualquer rotina de tratamento registrada anteriormente.

Configurando atributos de rotina de tratamento de eventos

As propriedades de rotina de tratamento de evento de um elemento do documento também podem ser configuradas como atributos na marcação HTML correspondente. Se isso for feito, o valor do atributo deve ser uma string de código JavaScript. Esse código deve ser o corpo da função de tratamento de evento, e não uma declaração de função completa. Isto é, seu código de tratamento de evento HTML não deve ser circundado por chaves e nem prefixado com a palavra-chave function.

Isso tudo é apenas a definição técnica para os eventos colocados nos elementos do documento diretamente no código HTML. Observe:

```
<button type="button" id="botao" style="font-size: 150%;" onclick="suaFuncao();"
>Apenas um botão</button>
```

Se um atributo de rotina de tratamento de eventos HTML contém várias instruções JavaScript, você deve lembrar-se de separar essas instruções com pontos e vírgulas, ou decompor o valor do atributo em várias linhas como faria no JavaScript.

Alguns tipos de evento são direcionados ao navegador como um todo, ao invés de serem direcionados a qualquer elemento em especial do documento. Em JavaScript, as rotinas de tratamento para esses eventos são registradas no objeto Window. Em HTML, são colocadas na marcação <body>, embora o navegador as registre no objeto Window.

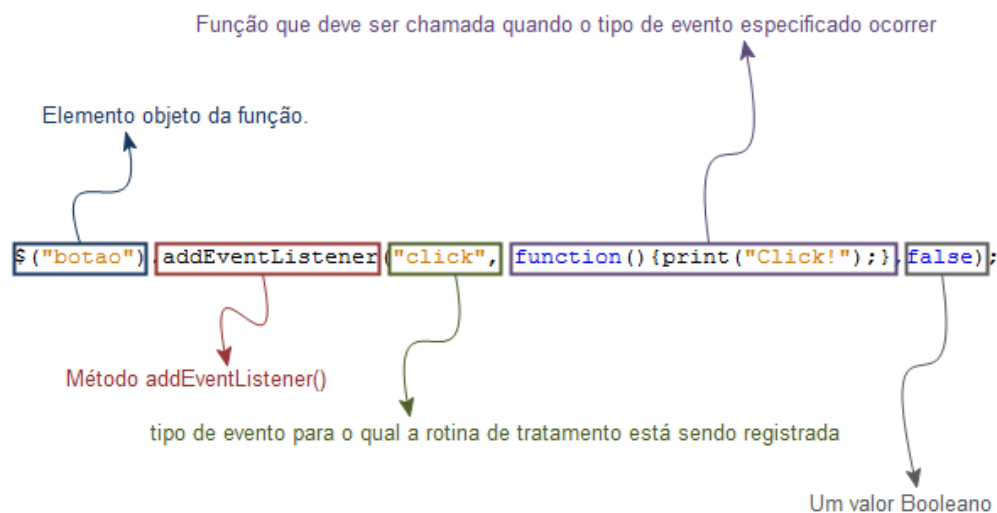
addEventListener()

No modelo de evento-padrão suportado por todos os navegadores, menos o IE8 e anteriores, qualquer objeto que possa ser alvo de evento – isso inclui os objetos Window e Document e todos os objetos Element do documento – define um método chamado `addEventListener()`, que pode ser usado para registrar uma rotina de tratamento de evento para esse alvo.

`addEventListener()` recebe três argumentos:

1. O primeiro é o tipo de evento para o qual a rotina de tratamento está sendo registrada. O tipo de evento (ou nome) é uma string e não deve incluir o prefixo `on`, utilizado para configurar propriedades de tratamento de evento;
2. O segundo argumento é a função que deve ser chamada quando o tipo de evento especificado ocorrer; e
3. O terceiro é um valor booleano. Normalmente você irá passar `false` para esse argumento. Se, ao invés disso, você passar `true`, sua função será registrada como uma rotina de tratamento de evento de captura e será chamada em uma fase diferente do envio do evento (a captura de eventos será abordada em outro momento).

Para uma representação visual do método, observe a imagem abaixo:



Agora que você visualizou a estrutura, crie um exemplo em que eventos são atribuídos a alguns elementos do documento. Abra o arquivo **eventos.js** e modifique o código para fique assim:

```
// Aguardar o documento ser carregado para atribuir os eventos
window.onload = function() {

    // Recuperando o elemento com id 'texto'
    var texto = document.getElementById("texto");

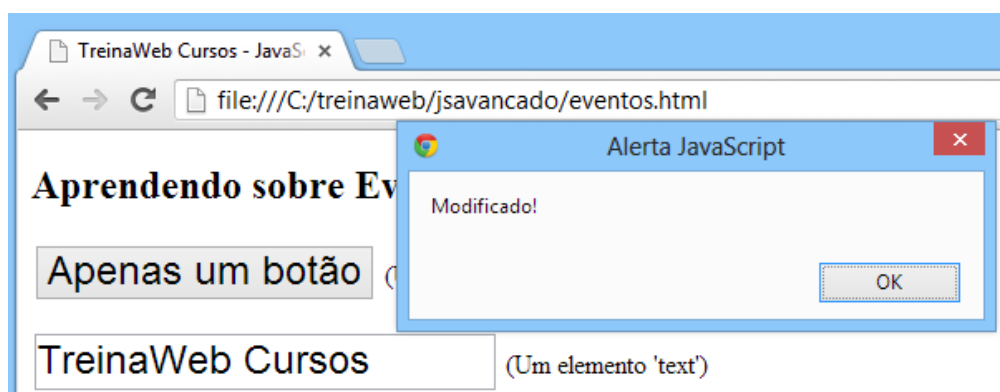
    // Atribuindo a ele um evento change
    texto.addEventListener("change", function() { alert("Modificado!");}, false);

    // Agora trabalhando com o elemento button
    var botao = document.getElementById("botao");
```

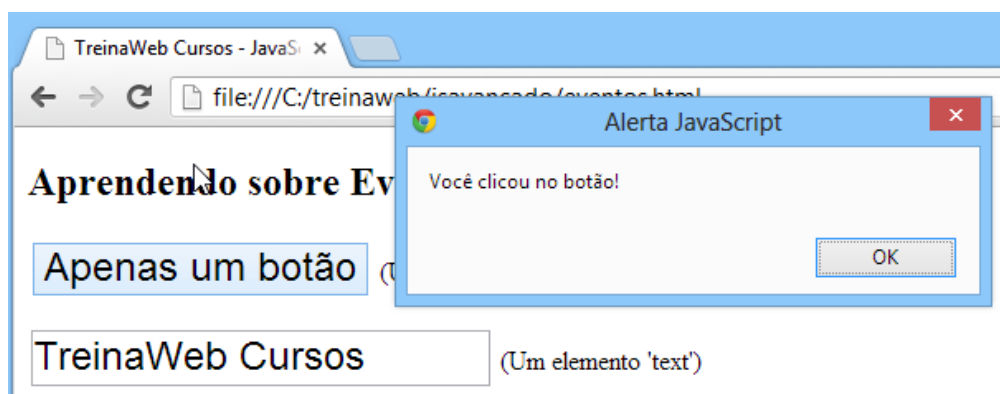
```
// Configurando uma propriedade de tratamento de evento
botao.onclick = function() { alert("Você clicou no botão!"); };

// Agora usando addEventListener()
botao.addEventListener("click", function() { alert("Outro alerta!"); }, false);
};
```

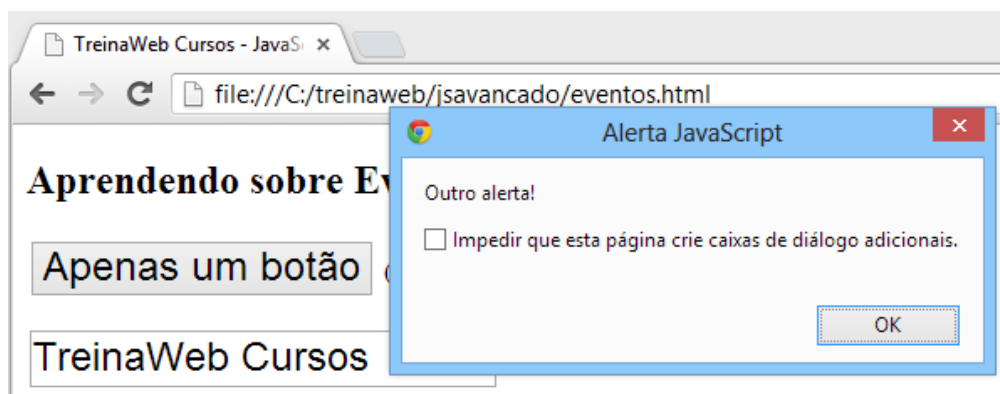
Salve o arquivo e atualize a página **eventos.html** no navegador. Na caixa de texto, digite **TreinaWeb Cursos** e depois clique em algum ponto na página que não seja o botão. A seguinte mensagem será exibida:



O evento **onchange** foi atribuído ao elemento. Irá acontecer algo curioso quando você clicar no botão, faça o teste:



Um alerta será exibido e assim que você clicar em **OK** outro alerta será exibido:



Chamar `addEventListener()` com “click”, como primeiro argumento, não afeta o valor da propriedade `onclick`. No código anterior, um clique no botão gerou duas caixas de diálogo `alert()`. Mais importante, você pode chamar `addEventListener()` várias vezes para registrar mais de uma função de tratamento para o mesmo tipo de evento no mesmo objeto.

Quando ocorre um evento em um objeto, todas as rotinas de tratamento registradas para esse tipo de evento são chamadas na ordem em que foram registradas. Chamar `addEventListener()` mais de uma vez, no mesmo objeto e com os mesmos argumentos, não tem efeito nenhum. A função de tratamento é registrada somente uma vez e a chamada repetida não altera a ordem em que as rotinas de tratamento são chamadas.

removeEventListener()

`removeEventListener()` é um método correlacionado com `addEventListener()`. Ele também espera três argumentos, mas remove a função de tratamento de evento de um objeto, ao invés de adicioná-la.

Às vezes, é interessante registrar uma rotina de tratamento de evento temporariamente e removê-la logo depois. Abaixo seguem informações sobre os três argumentos:

1. É uma string que especifica o nome do evento que o manipulador de eventos irá verificar.
2. Referência para a função de manipulador de eventos que irá remover.
3. É um valor booleano que especifica se o manipulador de eventos registrará ou não o evento de captura.

A lógica é a mesma utilizada para `addEventListener()`. Observe o exemplo:

```
texto.removeEventListener("change", function() { alert("Modificado!"); }, false);
```

O código acima irá remover a função que foi atribuída ao evento “change” no código anterior.

Chamada de rotina de tratamento de evento

Quando você tiver registrado uma rotina de tratamento de evento, o navegador Web irá chamá-lo automaticamente quando o tipo de evento determinado estiver no objeto especificado. As chamadas de rotina de tratamento de evento serão detalhadas a seguir. Acompanhe as explicações sobre os argumentos da rotina de tratamento, o contexto da chamada (o valor `this`), o escopo da chamada e o significado do valor de retorno de uma rotina de tratamento de evento. Infelizmente, para o IE8 e anteriores, alguns desses detalhes diferem dos outros navegadores.

Argumentos de rotina de tratamento de evento

As rotinas de tratamento de evento normalmente são chamadas tendo o objeto evento como único argumento. As propriedades do objeto evento fornecem detalhes sobre o evento. A propriedade `type`, por exemplo, especifica o tipo de evento ocorrido.

Contexto de rotina de tratamento de evento

Quando uma rotina de tratamento de evento é registrada pela configuração de uma propriedade, é como se fosse definido um novo método no elemento documento.

Portanto, não é de surpreender que as rotinas de tratamento de evento sejam chamadas como métodos do objeto no qual são definidas. Isto é, dentro do corpo de uma rotina de tratamento de evento, a palavra-chave `this` se refere ao alvo do evento.

As rotinas de tratamento são chamadas tendo o alvo como valor `this`, mesmo quando registradas com `addEventListener()`.

Escopo de rotina de tratamento de evento

Assim como toda função de JavaScript, as rotinas de evento têm escopo léxico. Elas são executadas no escopo em que são definidas, e não no escopo no qual são chamadas, e podem acessar qualquer variável local desse escopo.

Porém, as rotinas de tratamento de evento registradas como atributos HTML são um caso especial. Elas são convertidas em funções de nível superior que têm acesso às variáveis globais, mas não às variáveis locais. Por motivos históricos, elas são executadas com encadeamento de escopo modificado. As rotinas de tratamento de eventos, definidas por atributos HTML, podem usar as propriedades do objeto de alvo, do objeto contêiner `<form>` e do objeto `Document`, como se fossem variáveis locais.

Valor de retorno de rotina de tratamento

O valor de retorno de uma rotina de tratamento de evento, registrada pela configuração de uma propriedade de objeto ou de um atributo HTML, às vezes tem significado. Em geral, o valor de retorno `false` diz ao navegador que não deve executar a ação-padrão associada ao evento. A rotina de tratamento `onclick` de um botão `Submit` em um formulário, por exemplo, pode retornar `false` para evitar que o navegador envie o formulário. Da mesma forma, uma rotina de tratamento `onkeypress` em um campo de entrada pode filtrar a entrada de teclado retornando `false`, caso o usuário digite um caractere inadequado.

O valor de retorno da rotina de tratamento `onbeforeunload` do objeto `Window`, também tem significado. Esse evento é disparado quando o navegador está prestes a navegar para uma nova página. Se essa rotina de tratamento de evento retornar uma string, ela será exibida em uma caixa de diálogo modal, que pede para o usuário confirmar se deseja sair da página.

Ordem de chamada

Um elemento do documento, ou outro objeto, pode ter mais de uma rotina de tratamento de evento registrada para um tipo de evento em especial. Quando ocorre um evento apropriado, o navegador deve chamar todas as rotinas de tratamento, seguindo estas regras de ordem de chamada:

1. As rotinas de tratamento registradas pela configuração de uma propriedade do objeto ou atributo HTML, se houve, são sempre chamadas primeiro.
2. As rotinas de tratamento registradas com `addEventListener()` são chamadas na ordem em que foram registradas.
3. As rotinas de tratamento registradas com `attachEvent()` podem ser chamadas em qualquer ordem e seu código não deve depender de chamada sequencial.

Cancelamento de evento

Nos navegadores que suportam `addEventListener()` é possível cancelar a ação-padrão de um evento, chamando o método `preventDefault()` do objeto do evento. No IE8 e anteriores, no entanto, o mesmo efeito irá configurar como `false` a propriedade `returnValue` do objeto evento.

Cancelar a ação-padrão associada a um evento é apenas um tipo de cancelamento de evento, sendo possível também cancelar a propagação de eventos. Nos navegadores que suportam `addEventListener()`, o objeto evento possui o método `stopPropagation()`, que pode ser chamado para impedir a continuação da propagação do evento. Se existem rotinas de tratamento definidas no mesmo objeto, o restante das rotinas de tratamento ainda será chamado, mas rotina alguma de tratamento de evento em qualquer outro objeto será chamada após a chamada de `stopPropagation()`. Esse método pode ser chamado a qualquer momento durante a propagação de eventos.

Eventos de carga de documento

Você aprendeu sobre os fundamentos do tratamento de eventos JavaScript. Conheça, então, com mais detalhes, as categorias específicas de eventos. Comece pelos eventos de carga de documento.

A maioria dos aplicativos Web precisa de notificação do navegador Web para saber quando o documento foi carregado e quando está pronto para ser manipulado. O evento load no objeto Window tem esse objeto. O evento load não é disparado até que o documento e todas as suas imagens estejam totalmente carregados. O mais seguro é começar a executar seus scripts depois que o documento estiver totalmente analisado, mas antes que as imagens sejam baixadas. O tempo de inicialização dos aplicativos Web pode ser melhorado se você disparar seus scripts em eventos que não sejam load.

Acompanhe o exemplo a seguir para praticar a teoria acima. Abra o arquivo **eventos.html** e insira este código antes da tag `</body>`:

```


<br>



```

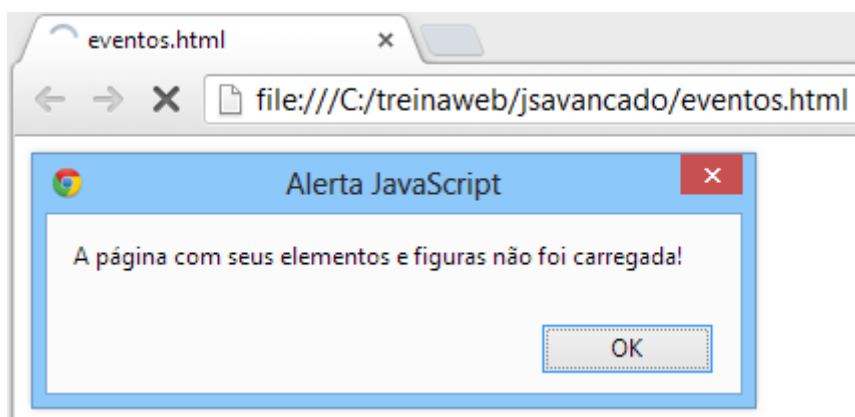
Salve o arquivo. Agora abra o arquivo **eventos.js** e modifique o código para que fique assim:

```
// Alerta antes da página ser carregada
alert("A página com seus elementos e figuras não foi carregada!");

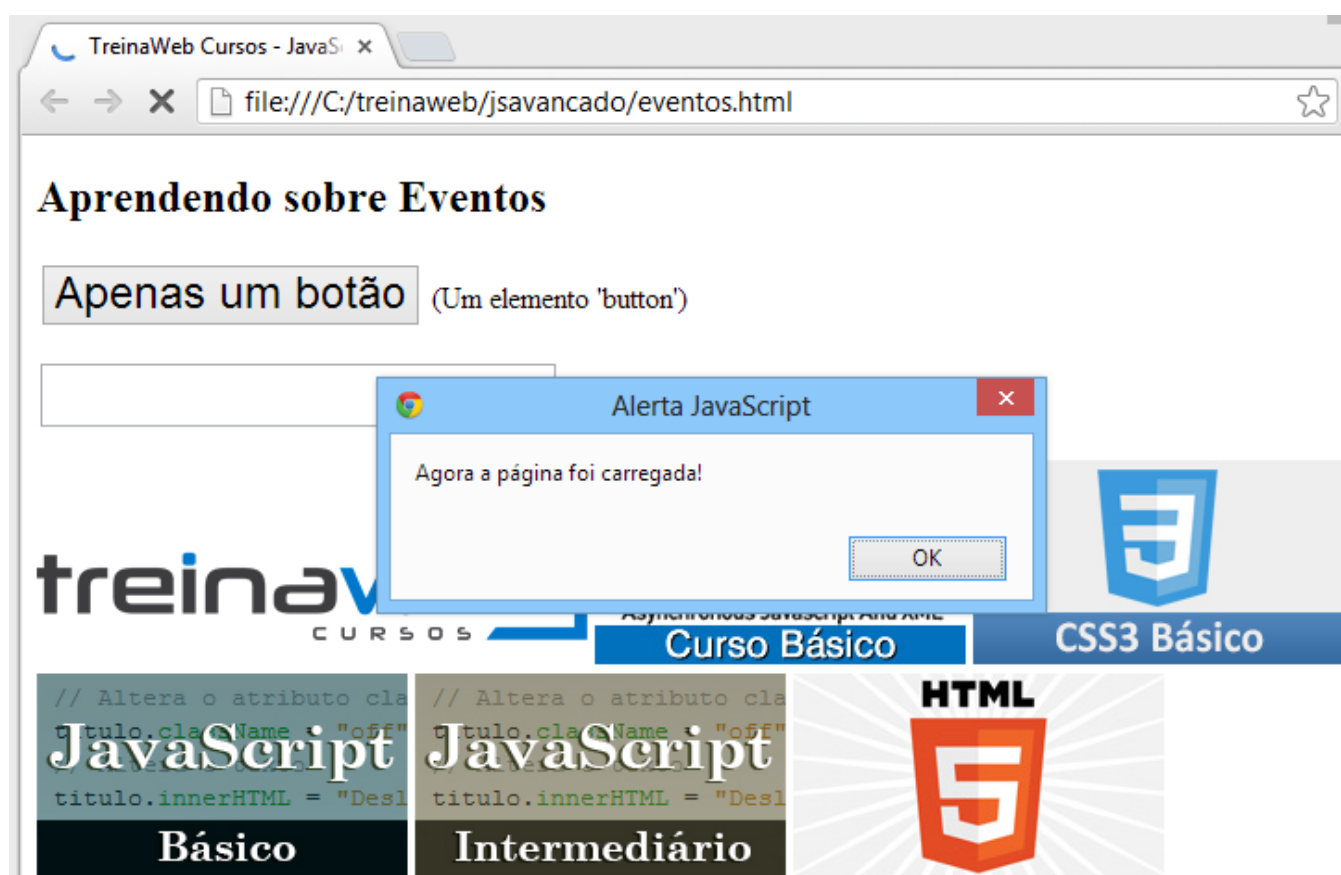
// Aguardar o documento ser carregado para atribuir os eventos
window.onload = function() {

    // Alerta informando que a página foi carregada
    alert("Agora a página foi carregada!");
};
```

Salve o arquivo e abra a página **eventos.html** no navegador. Será exibida a seguinte mensagem:



Após os elementos e as imagens serem carregadas, esta imagem será exibida:



Acompanhe, a seguir, o exemplo com a função `quandoPronto()`. As funções enviadas para `quandoPronto()` serão chamadas quando o documento estiver pronto para ser manipulado, ao contrário de `onload`. Modifique o arquivo **eventos.js** para que o código fique como este:

```
/*
 * Passa uma função quandoPronto() e ela será chamada (como um método do
 * documento) quando o documento for analisado e estiver pronto para manipulação.
 * As funções registradas são disparadas pelo primeiro evento DOMContentLoaded,
 * readystatechange ou load que ocorrer. Quando o documento estiver pronto e
 * todas as funções tiverem sido chamadas, qualquer função passada para quandoPronto()
 * será chamada imediatamente.
 */
```

```
*/

// Esta função retorna a função quandoPronto()
var quandoPronto = (function() {

    // Funções a serem executadas quando receberem um evento
    var funcs = [];

    // Troca para true quando a rotina de tratamento é disparada
    var pronto = false;

    // A rotina de tratamento de evento é chamada quando o documento está pronto
    function controladora(e) {

        // Se foi executada uma vez, apenas retorna
        if (pronto) return;

        // Se foi um evento readystatechange onde o estar mudou para algo que
        // não seja "complete", então ainda não está pronto
        if(e.type === "readystatechange" && document.readyState !== "complete"){
            return;
        }

        // Executa todas as funções registradas.
        // Observe a pesquisa a funcs.length a cada vez, no caso de a chamada
        // de uma dessas funções fazer com que mais funções sejam registradas.
        for(var i = 0; i < funcs.length; i++){
            funcs[i].call(document);
        }

        // Agora configura o flag ready como true e esquece as funções
        ready = true;
        funcs = null;
    }

    // Registra a rotina de tratamento de qualquer evento que possa receber
    if (document.addEventListener){
        document.addEventListener("DOMContentLoaded", controladora, false);
        document.addEventListener("readystatechange", controladora, false);
        window.addEventListener("load", controladora, false);
    }

    // Informa que a função não é suportada pelo navegador e retorna
    else{
        alert("Esta função não é suportada pelo navegador!");
        return;
    }

    // Retorna a função quandoPronto
    return function quandoPronto(f) {
        if(pronto) {

            // Se já está pronto, apenas executa
            f.call(document);
        }
        else{

            // Caso contrário, a enfileira para depois.
            funcs.push(f);
        }
    }
}
```

```
};  
} ( ) ;
```

Salve o arquivo. Esse exemplo não será utilizado agora, mas mais adiante, quando o seu funcionamento será explicado.

Eventos de mouse

Existem muitos eventos relacionados ao mouse. Os eventos click em links e botões Submit têm ações-padrão que podem ser evitadas. Você pode cancelar um evento contexto menu para impedir a exibição de um menu de contexto, mas alguns navegadores têm opções que tornam os menus de contexto não canceláveis. Relembrando: menus de contexto são exibidos quando se clica com o botão direito do mouse sobre o documento ou sobre algum elemento.

O objeto passado para as rotinas de tratamento de evento de mouse tem propriedades clientX e clientY que especificam as coordenadas do cursor do mouse em relação à janela contêiner. Para converter essa posição em coordenadas do documento, some os deslocamentos de rolagem da tabela.

Para você ter uma ideia de como funciona, abra o console do navegador e digite este código:

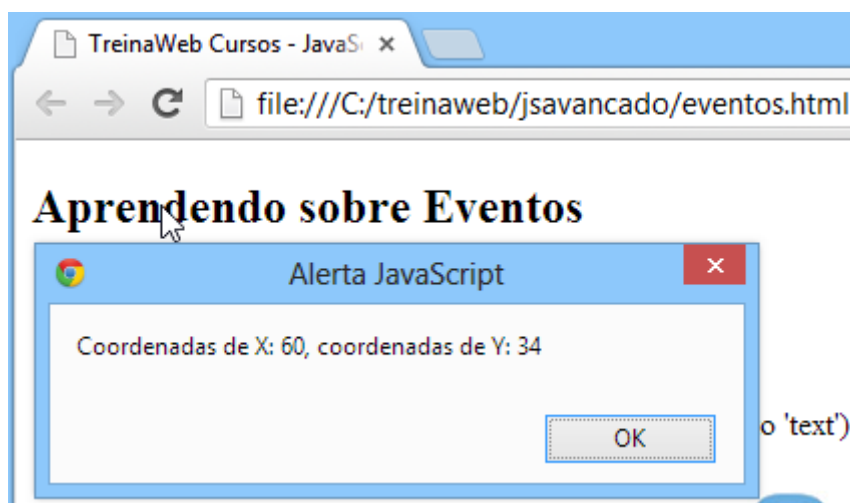
```
// Criando o objeto com o elemento que deseja manipular
var objeto = document.getElementsByTagName("h2");

// Função responsável por exibir as coordenadas
function coordenadas(event)
{
    // Salvando os valores das coordenadas
    var x = event.clientX;
    var y = event.clientY;

    // Exibindo as coordenadas do mouse onde houve o click
    alert("Coordenadas de X: " + x + ", coordenadas de Y: " + y);
}

// Adicionando o evento click ao elemento
objeto[0].addEventListener("click", coordenadas);
```

Execute o comando acima e clique em alguma parte do texto *Aprendendo sobre Eventos*, no navegador. O resultado será um alerta com as coordenadas do mouse durante o evento “click”:



As propriedades altKey, ctrlKey, metaKey e shiftKey especificam se várias teclas modificadoras do teclado estavam pressionadas quando o evento ocorreu: isso permite distinguir um clique normal de um clique com a tecla Shift pressionada, por exemplo.

Novamente, no console do navegador digite o seguinte código:

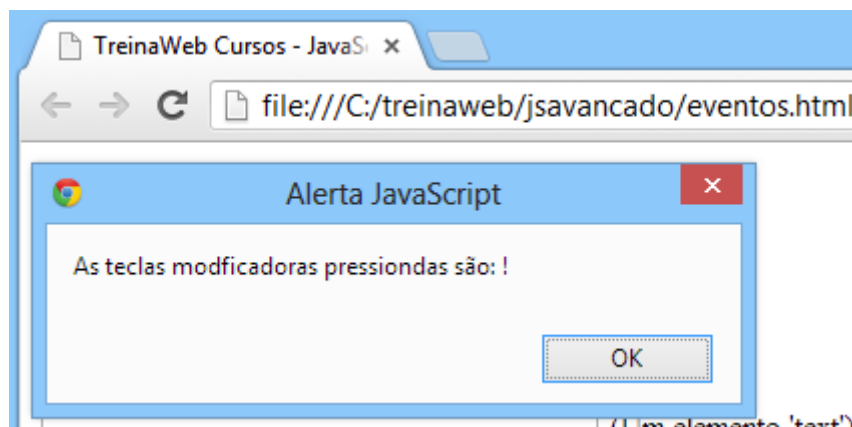
```
// Selecionando o elemento 'button' para adicionar o evento 'click'
var botao = document.getElementById("botao");

// Função responsável por verificar se as teclas modificadoras foram pressionadas
// durante o evento
function teclasModificadoras(event)
{
    // Verificando quais teclas modificadoras foram pressionadas
    var alt = event.altKey ? "alt " : "";
    var ctrl = event.ctrlKey ? "ctrl " : "";
    var shift = event.shiftKey ? "shift " : "";
    var meta = event.metaKey ? "meta " : "";

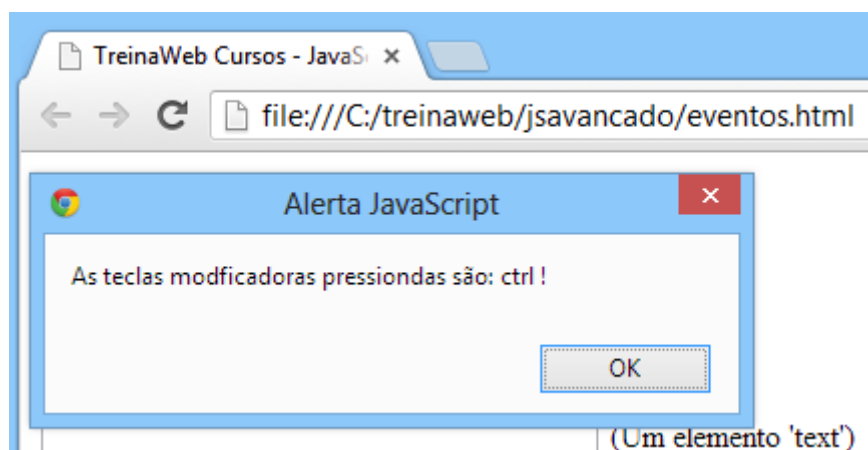
    // Exibindo as teclas modificadoras que foram pressionadas
    alert("As teclas modificadoras pressionadas são: "+alt+ctrl+shift+meta+"!");
}

// Atribuindo o evento ao elemento
botao.addEventListener("click", teclasModificadoras);
```

Execute o código e clique no botão **Apenas um botão**, na página que está no navegador. O resultado será:



Como nenhuma tecla modificadora foi pressionada, nenhuma é informada. Agora, antes de clicar no botão, pressione e segure a tecla **Ctrl**:



A tecla modificadora pressionada é exibida na mensagem.

Os principais eventos de mouse são:

1. **click**: Um evento de nível mais alto, disparado quando o usuário pressiona e solta um botão do mouse ou “ativa” um elemento de algum modo.
2. **contextmenu**: Um evento cancelável, disparado quando um menu de contexto está para ser exibido. Os navegadores atuais exibem menus de contexto em cliques do botão direito do mouse, de modo que esse evento também pode ser usado como o evento click.
3. **dblclick**: Disparado quando o usuário dá um click duplo com o mouse.
4. **mousedown**: Disparado quando o usuário solta um botão do mouse.
5. **mousemove**: Disparado quando o usuário move o mouse.
6. **mouseover**: Disparado quando o mouse entra em um elemento.
7. **mouseout**: Disparado quando o mouse sai de um elemento.

Eventos de roda de mouse

Todos os navegadores modernos suportam rodas de mouse e disparam eventos quando o usuário gira a roda do mouse. Muitas vezes os navegadores usam a roda do mouse para rolar o documento ou aproximar e afastar, mas é possível cancelar o evento `mousewheel` para enviar ações-padrão.

Existem vários problemas de interoperabilidade que afetam os eventos de roda do mouse, mas é possível escrever um código que funcione em todas as plataformas.

Os problemas envolvem o nome do evento (`mousewheel` para quase todos os navegadores, no Firefox é apenas `wheel`), o objeto passado a esses eventos e também as diferenças de hardware (algumas rodas, além de mover a página para frente e para trás também movem para a esquerda e para a direita).

Mas, para todos esses tipos de evento, o objeto evento é como um objeto de mouse: inclui as coordenadas do cursor do mouse e o estado das teclas modificadoras do teclado.

O objeto evento passado para uma rotina de tratamento de “roda de mouse” possui a propriedade `wheelDelta` que especifica quando o usuário girou a roda. Um “clique” de roda do mouse girando na direção contrária à direção do usuário, geralmente é um delta de 120; e um clique na sua direção é -120.

No Firefox, você pode usar o evento não padronizado `DOMMouseScroll`, no lugar de `mousewheel`, e usar a propriedade `detail` do objeto evento, ao invés de `wheelDelta`. Entretanto, a escala e o sinal da propriedade `detail` são diferentes de `wheelDelta`: multiplique `detail` por -40 para calcular o valor de `wheelDelta` equivalente.

Pratique o que aprendeu, executando este exemplo. Abra o arquivo **eventos.html** e insira este código logo abaixo de ‘<script type=“text/javascript” src=“eventos.js”></script>’:

```
<style>
  div#marcador {
    position: absolute;
    top: 50%;
    left: 50%;
    width: 100px;
    height: 100px;
    margin-left:-50px;
    margin-top:-50px;
    background:#f00;
    text-align: center;
    vertical-align:middle;
    line-height:100px;
    color:#fff;
  }
</style>
```

Abaixo da tag <body> insira o seguinte código:

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

Salve o arquivo e atualize a página **eventos.html** no navegador. Será exibido um quadrado de cor vermelha, no centro da tela. Mova a roda do mouse e veja que nada irá acontecer, ainda:



Agora abra o arquivo **eventos.js** e insira no fim do arquivo o código abaixo:

```
// Criando uma função para controlar o que deve ser feito ao se utilizar
// a roda do mouse
function handle(delta) {

    // Selecionado a posição 'top' do elemento que será manipulado
    // pela roda do mouse
    var top = document.getElementById('marcador').style.top;

    // Se o valor de top for vazio
    if(top === '') {

        // Declarando a variável que armazenará a posição Y do elemento
        var y;

        // Verifica se o navegador aceita a propriedade innerHeight
        if (self.innerHeight) {

            // Se aceitar, armazena o seu valor em y
            y = self.innerHeight;

        }

        // Se não aceitar, verifica se aceita clientHeight
        else if (document.documentElement && document.documentElement.clientHeight) {
```



```
// Se aceitar, armazena o valor em y
y = document.documentElement.clientHeight;
}

// Em último caso utilizar o corpo (body) da página como referência
else if (document.body) {

    // E armazena o valor em y
    y = document.body.clientHeight;
}

// Definindo a posição top do elemento na página
top = Math.round((y - 50)/2);
}

// Se o valor de top não for nulo
else {

    // Transforma o valor em inteiro e retira o texto 'px' dele
    top = parseInt(top.replace('px', ''));
}

// Se o valor enviado para a função for menor que 0 retira o valor
// 8 para a posição top do elemento. Caso contrário, atribui 8
top = delta < 0 ? top - 8 : top + 8;

// Seleciona o elemento e modifica o seu atributo top.
document.getElementById('marcador').style.top = top+'px';
}

// Função chamada quando o evento acontece
function wheel(event) {

    // Definindo uma variável delta e atribuindo o valor 0
    var delta = 0;

    // Efetuando teste para verificar se o navegador suporta o código
    if (!event) event = window.event;

    // Verificando se existe uma propriedade wheelDelta
    if (event.wheelDelta) {

        // Se existir, atribui o seu valor dividido por 120 para a variável
        // delta
        delta = event.wheelDelta/120;

        // Se o navegador for o Opera ou o Chrome modifica o valor de delta para
        // -delta
        if (window.opera) delta = -delta;
    }

    // Se não existir, verifica se a propriedade detail existe
    else if (event.detail) {

        // Se existir pega o valor da propriedade delta e divide por 3, além
        // de modificar o estado, utilizando o operador -
        delta = -event.detail/3;
    }
}
```

```
// Se houver valor em delta, então, chama a função handle passando o valor
// de delta
if (delta) handle(delta);

// Utiliza o método preventDefault() para cancelar a ação-padrão
// do evento, se ela estiver disponível para o navegador
if (event.preventDefault) event.preventDefault();

// Para garantir, caso o navegador não aceite preventDefault(), passe
// false para returnValue para cancelar o evento
event.returnValue = false;
}

// Verifica se o navegador suporta addEventListener. Se suportar, o utiliza
if (window.addEventListener)
    window.addEventListener('DOMMouseScroll', wheel, false);

// Para garantir configura o evento no documento.
window.onmousewheel = document.onmousewheel = wheel;
```

Salve o arquivo e, no navegador, atualize a página **eventos.html**. Role a roda do mouse e veja que, agora, o quadrado se movimentará:



Eventos de arrastar e soltar

Arrastar e soltar (ou DnD – sigla de Drag-and-drop, em inglês) é a interface com o usuário para transferir dados entre uma “origem de arrasto” e um “alvo de soltura”, que pode ser no mesmo aplicativo ou em aplicativos diferentes. DnD é uma interação homem/computador complexa e as APIs para implementar DnD são complexas:

1. Precisam estar vinculadas ao sistema operacional subjacente para que possam funcionar entre aplicativos não relacionados.
2. Devem conciliar as operações “mover”, “copiar” e “vincular” de transferência de dados, e deixar que a origem de arrasto e o alvo de soltura restrinjam o conjunto de operações permitidas para, então, permitir que o usuário faça uma escolha (normalmente modificadoras do teclado) no conjunto permitido.
3. Devem fornecer uma maneira para que a origem de arrasto especifique o ícone ou a imagem a ser arrastada.
4. Devem fornecer uma notificação baseada em evento, tanto para a origem do arrasto com para o alvo de soltura, do andamento da interação DnD.

A DnD é sempre baseada em eventos e a API JavaScript envolve dois conjuntos de eventos: um disparado na origem do arrasto e outro disparado no alvo de soltura. Todas as rotinas de tratamento de evento de DnD recebem um objeto evento parecido com um objeto evento de mouse, com a adição de uma propriedade `dataTransfer`. Essa propriedade refere-se a um objeto `DataTransfer` que define os métodos e as propriedades da API DnD.

Os eventos de origem de arrasto são relativamente simples. Qualquer elemento do documento que tenha o atributo HTML `draggable` é uma origem de arrasto:

```
elemento.draggable = true;
```

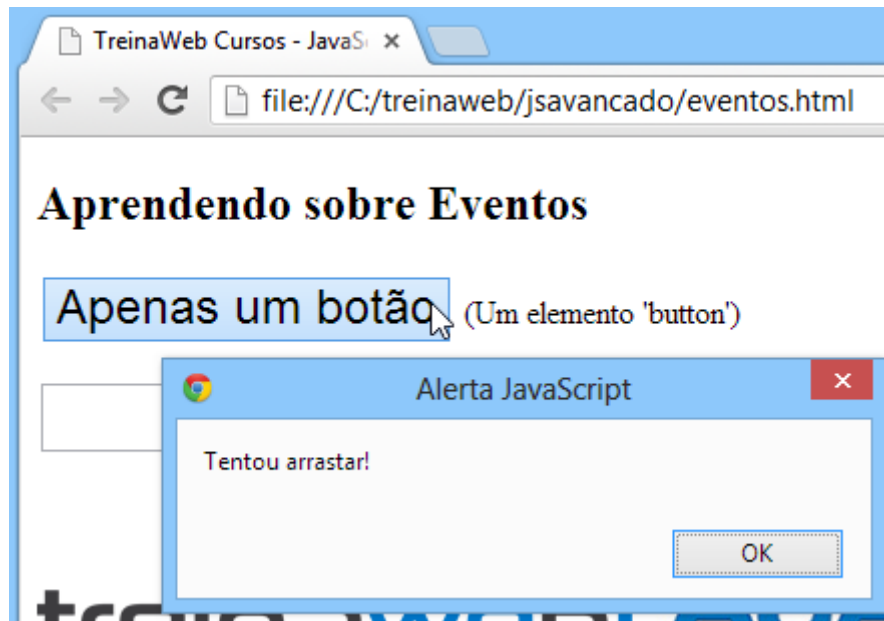
Quando o usuário inicia um arrastamento de mouse sobre uma origem de arrasto, o navegador não seleciona o conteúdo do elemento, ao invés disso ele dispara um evento `dragstart` no elemento. Para você ter uma ideia da teoria exposta, abra o console do navegador na página [eventos.html](#) e digite o seguinte código:

```
// Selecionando o elemento
var elemento = document.getElementById("botao");

// Informando que o elemento é passível do evento drag
elemento.draggable = true;

// Dando um alerta caso alguém tente arrastá-lo
elemento.ondragstart = function () { alert("Tentou arrastar!"); };
```

Depois de executar o código, clique no botão que está na página e tente arrastá-lo. A mensagem abaixo será exibida:



A rotina de tratamento para esse evento deve chamar `dataTransfer.setData()` para especificar os dados que a origem do arrasto está disponibilizando. A rotina de tratamento talvez queira configurar `dataTransfer.effectAllowed` para especificar quais das operações de transferência, “mover”, “copiar” e “vincular”, são suportadas; e talvez queira chamar `dataTransfer.setDragImage()` ou `dataTransfer.addElement()` para especificar uma imagem ou elemento do document a ser usado como representação visual do arrasto.

Enquanto o arrasto está em andamento, o navegador dispara eventos `drag` na origem de arrasto. Você pode receber esses eventos, se quiser atualizar a imagem de arrasto ou alterar os dados que estão sendo oferecidos, mas geralmente não é necessário registrar rotinas de tratamento de “arrasto”.

Quando ocorre uma soltura, o evento `dragend` é disparado. Se a origem de arrasto suporta uma operação “mover”, ele deve verificar `dataTransfer.dropEffect` para conferir se uma operação mover foi realmente realizada. Se foi, os dados foram transferidos para algum lugar e devem ser excluídos da origem do arrasto.

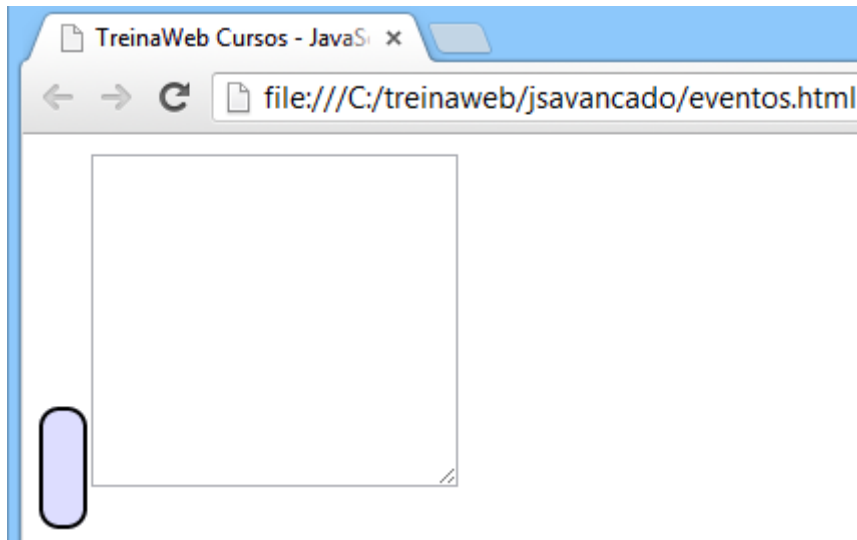
O evento `dragstart` é o único necessário para implementar origens de arrasto personalizadas simples. Crie um exemplo simples de arrastar. Abra o arquivo **eventos.html** e insira o código abaixo entre as tags `<style>` e `</style>`:

```
span#relogio{
  font: bold 24pt sans-serif;
  background: #ddf;
  padding: 10px;
  border: solid black 2px;
  border-radius: 10px;
}
```

Agora, logo abaixo da linha `<div id="marcador"></div>`, insira o seguinte código:

```
<span id="relogio"></span>
<textarea cols="40" rows="20"></textarea>
```

Salve o arquivo e o abra o navegador. Novos elementos surgirão na tela:



Abra o arquivo **eventos.js** e insira em seu final o seguinte código:

```
// Utilizando a função quando pronto para evitar que o script gere erros
// por ser executado na hora errada
quandoPronto(function() {

    // Armazenando o elemento com id 'relogio'
    var relógio = document.getElementById("relogio");

    // Uma imagem para arrastar
    var icon = new Image();

    // URL da imagem
    icon.src = "https://treinaweb-cursos.s3.amazonaws.com/prod/108/arquivos/curso-
ajax.jpg";

    // Exibe a hora a cada minuto
    function exibeHora() {

        // Obtém a hora atual
        var agora = new Date();

        // Atribuindo a hora e o minuto atual
        var hrs = agora.getHours(), mins = agora.getMinutes();

        if(mins < 10) mins = "0" + mins;

        // Exibe a hora atual no elemento
        relógio.innerHTML = hrs + ":" + mins;

        // Executa a função novamente em 1 minuto
        setTimeout(exibeHora, 6000);
    }

    // Executa a função exibe hora
    exibeHora();
```

```
// Torna possível arrastar o relógio
// Também é possível fazer isso com atributos HTML:
// <span draggable="true"> ...
relógio.draggable = true;

// Configura rotinas de tratamento de evento drag
relógio.ondragstart = function(event) {

    // Para compatibilidade com IE
    var event = event || window.event;

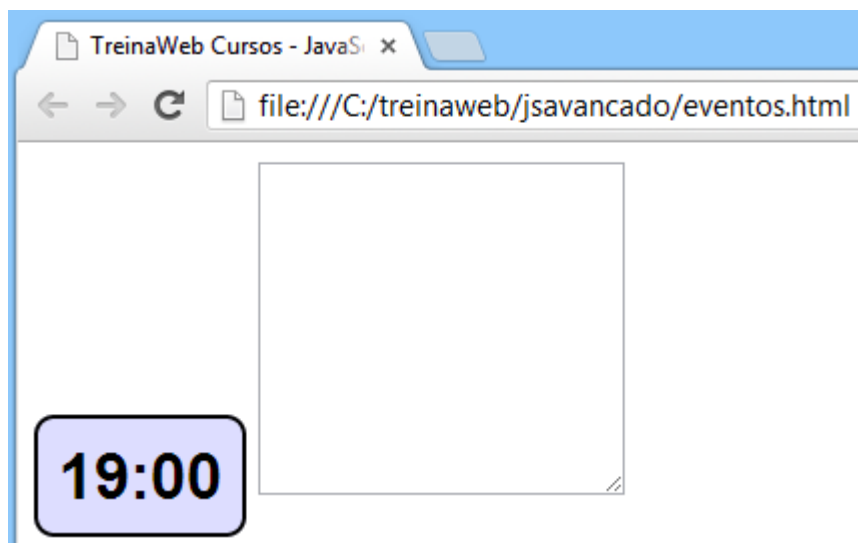
    // A propriedade dataTransfer é fundamental para a API de arrastar e soltar
    var dt = event.dataTransfer;

    // Informa ao navegador o que está sendo arrastado.
    // A construtora Date() usada como função retorna uma string de timestamp
    dt.setData("Text", Date() + "\n");

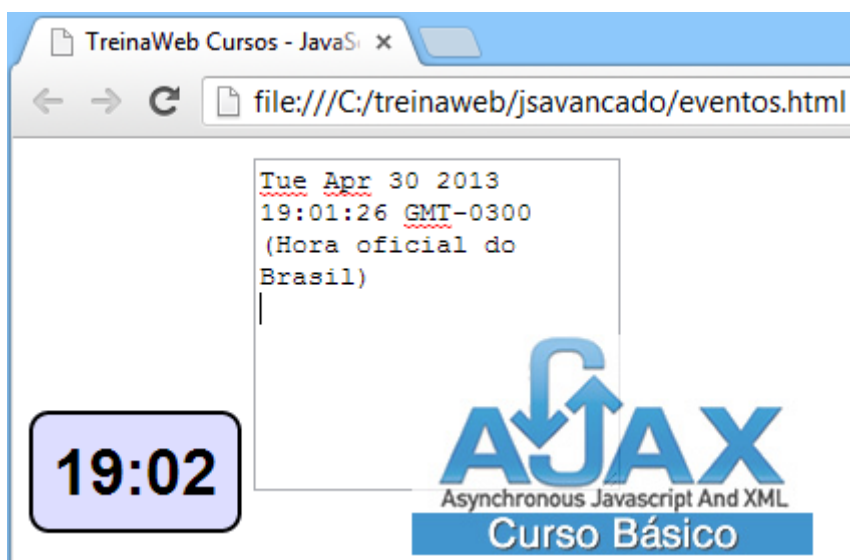
    // Diz ao navegador para arrastar o ícone a fim de representar o
    // timestamp em navegadores que o suportam. Sem essa linha, o
    // navegador pode usar uma imagem de texto do relógio como valor a
    // arrastar.
    if(dt.setDragImage) dt.setDragImage(icon, 0, 0);

};
});
```

Salve o arquivo e atualize a página **eventos.html** no navegador. O resultado será:



Clique no relógio e arraste-o sobre o textarea, uma imagem será exibida. Ao soltar sobre o textarea, o timestamp será inserido nele:



Os alvos de soltura são mais complicados do que as origens de arrasto. Qualquer elemento do documento pode ser um alvo de soltura. Não há necessidade de configurar um atributo HTML como acontece com as origens de arrasto, bastando definir receptores de evento apropriados.

São quatro os eventos disparados nos alvos de soltura, **mouse up**, **mouse move**, **mouse over**, e **mouse out**. Quando um objeto arrastado entra em um elemento do documento, o navegador dispara um evento **dragenter** nesse elemento. Seu alvo de soltura deve usar a propriedade **dataTransfer.types** para determinar se o objeto arrastado tem dados disponíveis em formato que ele possa entender. Também é possível verificar **dataTransfer.effectAllowed** para garantir que a origem de arrasto e seu alvo de soltura possam estar de acordo com uma das operações mover, copiar e vincular. Se essas verificações forem bem sucedidas, seu destino de soltura deve permitir que o usuário e o navegador saibam que ele está interessado em uma soltura. Você pode dar esse retorno ao usuário, mudando a borda ou a cor de fundo. Surpreendentemente, um alvo de soltura informa ao navegador que está interessado em uma soltura, cancelando o evento.

Eventos de texto

Antes de testar novos exemplos, 'limpe' um pouco o código. Os eventos de janela e mouse já foram praticados, então algumas coisas podem ser tiradas do código. Abra o arquivo **eventos.html** e modifique-o para que fique com este código:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>TreinaWeb Cursos - JavaScript Avançado</title>
    <script type="text/javascript" src="eventos.js"></script>
  </head>
  <body>
    <div id="marcador"></div>
    <h2>Aprendendo sobre Eventos </h2>
    <p> <button type="button" id="botao" style="font-size: 150%;">Apenas um
botão</button> (Um elemento 'button') </p>
    <p> <input type="text" id="texto" style="font-size: 150%;"> (Um elemento 'text') </p>
  </body>
</html>
```

Salve o arquivo. Bem melhor, não? Como alguns elementos foram apagados do documento, será preciso modificar o código JavaScript para que erros não sejam exibidos no console. Abra o arquivo **eventos.js** e comente; ou apague tudo o que envolva o último exemplo – tudo que estiver entre os códigos:

```
quandoPronto(function() {
  ....
});
```

Salve o arquivo e atualize a página **eventos.html**, no navegador. Abra o console e veja que nenhum erro será exibido:



Tudo pronto para prosseguir os estudos.

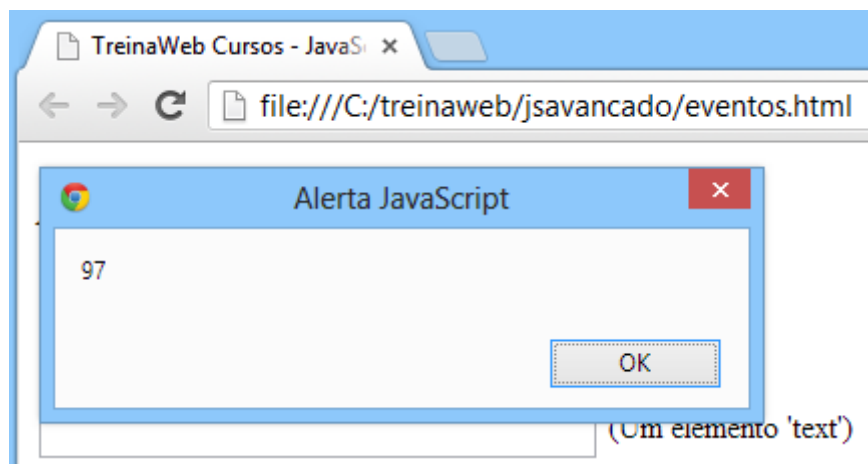
Os navegadores têm três eventos legados para entrada de teclado: Os eventos `keydown` e `keyup` (abordados no próximo tópico) e o evento `keypress`. O objeto evento, passado com eventos `keypress`, é um pouco confuso. Um evento `keypress` representa um único caractere de entrada. O objeto evento especifica esse caractere como uma posição de código Unicode numérica e deve ser usada a `String.fromCharCode()` para convertê-la em uma string.

Digite o código abaixo no console do navegador:

```
// Selecionando o elemento
var texto = document.getElementById("texto");

// Atribuindo o evento keypress
texto.addEventListener("keypress", function(event) {alert(event.keyCode);});
```

Execute-o e, na caixa de texto, digite a letra **a**. A seguinte mensagem será exibida:



Para exibir a letra ao invés de seu respectivo código, utilize o código abaixo:

```
// Selecionando o elemento
var texto = document.getElementById("texto");

// Converter o keyCode em letra
function converte(event) {

    // Retorna o código convertido
    alert(String.fromCharCode(event.keyCode));
}

// Atribuindo o evento keypress
texto.addEventListener("keypress", converte);
```

Atualize a página e execute o código acima no console. Veja que o retorno não será mais um número, mas a própria letra **a**.

Na maioria dos navegadores, a propriedade `keyCode` do objeto evento especifica a posição de código do caractere de entrada. Por motivos históricos, contudo, o Firefox usa a propriedade `charCode`. A maioria dos navegadores dispara eventos `keypress`

apenas quando é gerado um caractere imprimível. Contudo, o Firefox também dispara keypress para caracteres não imprimíveis. Para detectar esse caso, basta procurar um objeto evento com a propriedade charCode definida, mas configurada como 0.

O evento keypress pode ser cancelado para impedir a entrada do caractere. Isso significa que esse evento pode ser usado para filtrar a entrada e impedir que o usuário insira letras em campos destinados a dados numéricos, por exemplo.

Acompanhe este exemplo, que filtra a entrada de dados. Abra o arquivo **eventos.js** e insira o seguinte código:

```
// Permite somente números
function apenasNumeros(event)
{
    // Atribuindo o código da tecla digitada para a variável tecla
    var tecla = event.keyCode;

    // Verifica se o código da tecla digitada está entre o intervalo
    // que corresponde aos números
    if((tecla >= "97") && (tecla <= "122")){
        // Se estiver permite a exibição do número

        // Utilizando o método preventDefault() para cancelar a ação-padrão
        // do evento, se estiver disponível para o navegador
        if (event.preventDefault)event.preventDefault();

        // Para garantir, caso o navegador não aceite preventDefault() passe
        // false para returnValue para cancelar o evento
        event.returnValue = false;
    }
}

// Permite apenas letras
function apenasLetras(event)
{
    // Atribuindo o código da tecla digitada para a variável tecla
    var tecla = event.keyCode;

    // Verifica se o código da tecla digitada está entre o intervalo
    // que corresponde aos números
    if((tecla >= "48") && (tecla <= "57")){
        // Se estiver permite a exibição do número

        // Utilizando o método preventDefault() para cancelar a ação-padrão
        // do evento, se ela estiver disponível para o navegador
        if (event.preventDefault)event.preventDefault();

        // Para garantir, caso o navegador não aceite preventDefault() passe
        // false para returnValue para cancelar o evento
        event.returnValue = false;
    }
}
```

Salve o arquivo e atualize a página **eventos.html** no navegador. Depois digite o código abaixo no console do navegador para permitir apenas números na caixa de texto do documento:

```
// Selecionando o elemento
var texto = document.getElementById("texto");
```

```
// Atribuindo o evento keypress
texto.addEventListener("keypress", apenasNumeros);
```

Execute o código e tente digitar letras na caixa de texto. Veja que elas não serão exibidas. Mas, ao digitar números, eles serão exibidos.

O evento `keypress` é disparado antes que o texto, recentemente digitado, seja realmente inserido no elemento do documento em foco, sendo esse o motivo pelo qual as rotinas de tratamento para esses eventos podem cancelar o evento e impedir a inserção do texto.

Os navegadores também implementam um tipo de evento de entrada, que é disparado depois de o texto ser inserido em um elemento. O evento `input` é um deles. Esses eventos não podem ser cancelados e não especificam qual era o novo texto em seu objeto evento, mas fornecem notificações de que o conteúdo textual de um elemento mudou de algum modo. Para garantir que qualquer texto inserido em um campo de entrada apareça em letras maiúsculas, atualize a página **eventos.html** e digite o código abaixo no console:

```
// Selecionando o elemento
var texto = document.getElementById("texto");

// Atribuindo o evento input e definindo que todas as letras serão maiúsculas
texto.addEventListener("input", function() { this.value = this.value.toUpperCase(); });
```

Depois de executar o código acima, digite um texto na caixa de texto e veja que ele aparecerá em letras maiúsculas.

Eventos de teclado

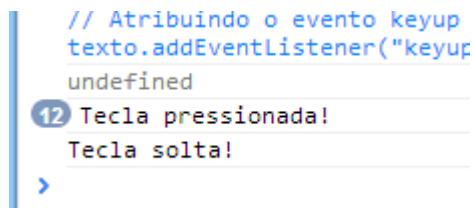
Os eventos `keydown` e `keyup` são disparados quando o usuário pressiona ou solta uma tecla no teclado. Eles são gerados por teclas modificadoras, teclas de função e teclas alfanuméricas. Se o usuário mantiver a tecla pressionada por tempo suficiente para que comece a repetir, ocorrerão vários eventos `keydown` antes que o evento `keyup` chegue. Atualize a página **eventos.html** e, no console, digite o código abaixo:

```
// Selecionando o elemento
var texto = document.getElementById("texto");

// Atribuindo o evento keydown
texto.addEventListener("keydown", function() { console.log("Tecla pressionada!"); });

// Atribuindo o evento keyup
texto.addEventListener("keyup", function() { console.log("Tecla solta!"); });
```

Execute o código e pressione, sem soltar, uma tecla qualquer na caixa de texto. No console, será exibido o seguinte:



```
// Atribuindo o evento keyup
texto.addEventListener("keyup",
undefined
12 Tecla pressionada!
Tecla solta!
>
```

O objeto evento, associado a esses eventos, possui a propriedade numérica `keyCode`, que especifica qual tecla foi pressionada. Para teclas que geram caracteres imprimíveis, `keyCode` geralmente é a codificação Unicode do caractere principal que aparece na tecla. As teclas de letra sempre geram valores de `keyCode` maiúsculos, independente do estado da tecla `Shift`, visto ser o que aparece na tecla física. Para o exemplo seguinte será emprestado um código usado anteriormente, e modificado apenas o evento. Atualize a página no navegador e, no console, digite este código:

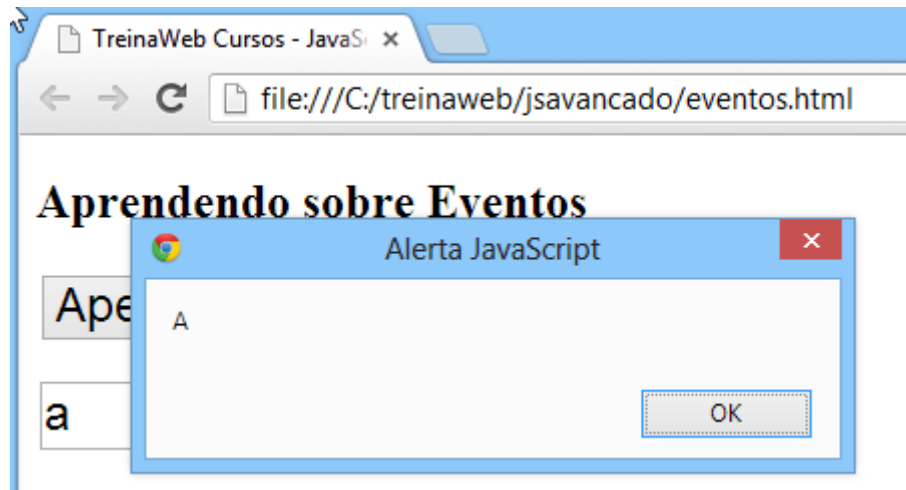
```
// Selecionando o elemento
var texto = document.getElementById("texto");

// Converter o keyCode em letra
function converte(event) {

    // Retorna o código convertido
    alert(String.fromCharCode(event.keyCode));
}

// Atribuindo o evento keydown
texto.addEventListener("keydown", converte);
```

Execute o código. Agora digite a letra **a**, sem pressionar a tecla **Shift**. O resultado será um **A** maiúsculo:



Da mesma forma, as teclas numéricas sempre geram valores `keyCode` para o dígito que aparece na tecla, mesmo que a tecla Shift seja pressionada para digitar caractere de pontuação.

Para teclas não imprimíveis, a propriedade `keyCode` terá outro valor. Os valores de `keyCode` nunca foram padronizados, mas é possível uma compatibilidade razoável entre os navegadores.

Assim como os objetos de evento de mouse, os objetos de evento de tecla têm as propriedades `altKey`, `ctrlKey`, `metaKey` e `shiftKey` que são configuradas como `true` se a tecla modificadora correspondente estiver pressionada quando o evento ocorrer. Outro exemplo será utilizado para os testes. Atualize a página e digite o código abaixo no navegador:

```
// Selecionando o elemento
var texto = document.getElementById("texto");

// Função responsável por verificar se as teclas modificadoras foram pressionadas
// durante o evento
function teclasModificadoras(event)
{
    // Verificando quais teclas modificadoras foram pressionadas
    var alt = event.altKey ? "alt " : "";
    var ctrl = event.ctrlKey ? "ctrl " : "";
    var shift = event.shiftKey ? "shift " : "";
    var meta = event.metaKey ? "meta " : "";

    // Exibindo as teclas modificadoras que foram pressionadas
    console.log("As teclas modificadoras pressionadas são: "+alt+ctrl+shift+meta+"!");
}

// Atribuindo o evento ao elemento,
texto.addEventListener("keydown", teclasModificadoras);
```

Execute o código acima e veja o que aparece no console ao pressionar uma tecla normal e depois a tecla **Ctrl**, por exemplo.

Os eventos `keydown` e `keyup`, e a propriedade `keyCode`, estão em uso há mais de uma década, mas nunca foram padronizados.