Apostila de JavaScript

Profº Ivan Souza

Versão: 1.0

Setembro de 2014

Introdução ao código JavaScript

JavaScript é uma linguagem de programação interpretada. Foi originalmente implementada como parte dos navegadores web para que scripts pudessem ser executados do lado do cliente e interagissem com o usuário sem a necessidade deste script passar pelo servidor, controlando o navegador, realizando comunicação assíncrona e alterando o conteúdo do documento exibido.

É atualmente a principal linguagem para programação client-side em navegadores web. Começa também a ser bastante utilizada do lado do servidor através de ambientes como o node.js. Foi concebida para ser uma linguagem script com orientação a objetos baseada em protótipos, tipagem fraca e dinâmica e funções de primeira classe. Possui suporte à programação funcional e apresenta recursos como fechamentos e funções de alta ordem comumente indisponíveis em linguagens populares como Java e C++.

História

JavaScript foi originalmente desenvolvido por Brendan Eich da Netscape sob o nome de Mocha, posteriormente teve seu nome mudado para LiveScript e por fim JavaScript. LiveScript foi o nome oficial da linguagem quando foi lançada pela primeira vez na versão beta do navegador Netscape 2.0 em setembro de 1995, mas teve seu nome mudado em um anúncio conjunto com a Sun Microsystems em dezembro de 1995 quando foi implementado no navegador Netscape versão 2.0B3.

A mudança de nome de LiveScript para JavaScript coincidiu com a época em que a Netscape adicionou suporte à tecnologia Java em seu navegador (Applets). A escolha final do nome causou confusão dando a impressão de que a linguagem foi baseada em java, sendo que tal escolha foi caracterizada por muitos como uma estratégia de marketing da Netscape para aproveitar a popularidade do recém-lançado Java.

JavaScript rapidamente adquiriu ampla aceitação como linguagem de script client-side de páginas web. Como consequência, a Microsoft desenvolveu um dialeto compatível com a linguagem de nome JScript para evitar problemas de marca registrada. JScript adicionou novos métodos para consertar métodos do Javascript relacionados a data que apresentavam problemas[carece de fontes]. JScript foi incluído no Internet Explorer 3.0, liberado em Agosto de 1996. Javascript e Jscript são tão similares que os dois termos são comumente usados de forma intercambiável. A Microsoft entretanto declara muitas características nas quais JScript não conforma com a especificação ECMA.

Em novembro de 1996 a Netscape anunciou que tinha submetido JavaScript para Ecma internacional como candidato a padrão industrial e o trabalho subsequente resultou na versão padronizada chamada ECMAScript.

JavaScript tem se transformado na linguagem de programação mais popular da web. Inicialmente, no entanto, muitos profissionais denegriram a linguagem pois ela tinha como alvo principal o público leigo. Com o advento do Ajax, JavaScript teve sua popularidade de volta e recebeu mais atenção profissional. O resultado foi a proliferação de frameworks e bibliotecas, práticas de programação melhoradas e o aumento no uso do JavaScript fora do ambiente de navegadores, bem como o uso de plataformas de JavaScript server-side.4

Em janeiro de 2009 o projeto CommonJS foi fundado com o objetivo de especificar uma biblioteca padrão para desenvolvimento JavaScript fora do navegador

Quando desenvolvemos páginas web, utilizamos como base a linguagem de marcação de hipertexto (HTML), que forma toda a estrutura do documento a partir de tags, trechos de código que são identificados e interpretados pelo navegador para montar algum elemento.

Aliando essa estrutura às CSS (Cascading Style Sheets), conseguimos estilizar os elementos da página, garantindo uma formatação visual mais completa e de fácil manutenção. Temos assim uma página bem estruturada, com vários elementos visuais (e não visuais) como formulários e tabelas bem formatados, com bordas, cores de plano de fundo, imagens, etc.

Porém, não vamos muito além disso, utilizando apenas HTML e CSS se obtemos apenas documentos ESTÁTICOS, sem interação com o usuário, como caixas de diálogo e validação de entrada de dados.

Para preencher essa lacuna, foi desenvolvida na década de 1990 uma linguagem de script chamada inicialmente de Mocha, posteriormente renomeada para LiveScript e finalmente lançada com o nome de **JAVASCRIPT**.

O nome **JavaScript** foi inicialmente motivo de confusão devido a semelhança com o nome da linguagem Java, mas estas duas tecnologias não estão diretamente relacionadas. Ou seja, JavaScript não é uma implementação especial da Java ou é baseado nesta última.

A JavaScript foi inicialmente implementada e lançada no browser Netscape 2.0, e hoje é a principal linguagem de script do lado cliente, suportada por todos os principais browsers.
Essa é chamada uma "linguagem do lado cliente" por que sua execução se dá diretamente no browser, independente de servidor. Assim, Javascript não é uma linguagem para acesso a banco de dados, mas para tratamento e dinamização dos elementos da página.

Por que frameworks são importantes?

Em primeiro lugar vamos entender o porquê do uso de frameworks ter crescido e ganhado tantos adeptos nos últimos anos de desenvolvimento web.

Quando queremos criar sites responsivos, de fácil manutenção, entre outras funcionalidades, é muito difícil e trabalhoso de se fazer manualmente e é aí que entram os frameworks, facilitando o trabalho do desenvolvedor.

E é exatamente por isso que os desenvolvedores web gostaram tanto de utilizar frameworks.

Em outro artigo eu listei 6 dos principais frameworks para desenvolvimento front-end, nesse artigo irei listar algumas das bibliotecas e dos frameworks mais famosos de javascript.

jQuery - http://jquery.com/

jQuery Framework

Esse, sem dúvida nenhuma é o mais importante de todos, acredito ter sido o pioneiro e mais utilizado em todo o mundo.

O jQuery é um excelente framework pois ele é rápido e leve, podendo ser criadas animações, manipulações de eventos, entre outras. O Ajax também é uma API muito fácil de usar e que é compatível com a grande parte dos navegadores. Sem dúvidas o jQuery definitivamente mudou o jeito de desenvolver na internet e mudou para melhor.

Muito utilizado por grandes empresas e projetos como Wordpress, Wikipédia, etc.

jQuery UI - http://jqueryui.com/

jQuery UI

O jQuery UI é uma variação no jQuery, ou seja, foi criado em cima da biblioteca jQuery.

É um conjunto de interações de interface dos usuários, efeitos, widgets e temas.

Se você precisa criar um site com uma grande interação com o usuário ou simplesmente inserir um plugin de data(date picker) em algum formulário de contato, por exemplo. jQuery UI é a sua escolha perfeita.

jQuery Mobile - http://jquerymobile.com/

jQuery Mobile

Também feito com base no jQuery, o jQuery mobile é a escolha perfeita para aplicações feitas para dispositivos móveis. Seu código é também baseado em HTML5 e possui u design bem flexível e facilmente personalizável.

Javascript MVC - http://javascriptmvc.com/

Javascript MVC

O JavaScriptMVC é um framework para o lado do cliente(cliente-side) no desenvolvimento em Javascript. É uma das melhores maneiras de se criar com qualidade aplicações de fácil manutenção em um espaço de tempo muito curto.

Framework com muitos recursos de suporte para geração de códigos, testes e gerenciamento de dependências.

Backbone.js - http://documentcloud.github.com/backbone/

Backbone.js

Esse framework dá estrutura para aplicações web por meio de modelos com valores-chave de eventos personalizados. Conta também com uma rica API com diversas funções por meio de sua interface RESTful JSON. Ao trabalhar em uma aplicação web que envolve um monte de JavaScript, uma das primeiras coisas que você aprende é parar de colocar todos os seus dados para o DOM. É muito fácil ver aplicações JavaScript que acabam como pilhas com diversos seletores de jQuery e callbacks. Quando falamos de aplicações ricas cliente-side, é muito mais útil uma abordagem um pouco mais estruturada. Com Backbone.js, você representa seus dados como modelos, que podem ser criados, validados, destruídos, e salvos no servidor. Nesse link é possível ver alguns dos exemplos em que são utilizados esse framework.

Exemplos de backbone.js - http://documentcloud.github.com/backbone/#examples

Knockout - http://knockoutjs.com/

Knockout Framework

Knockout é uma biblioteca de JavaScript que o ajuda a criar telas ricas e responsivas. Toda vez que você tem seções de interface do usuário que a atualização dinâmica (por exemplo, mudar dependendo das ações do usuário ou quando um muda a fonte de dados externas), podem ser implementadas de uma maneira mais simples e fácil com o Knockout.

Prototype - http://prototypejs.org/

Prototype JS

O prototype js leva a complexidade de todo desenvolvimento web para fora do lado do cliente. Foi criado para solucionar alguns problemas do mundo real e melhora a interface do Ajax e do DOM.

MooTools - http://mootools.net/

MooTools Framework

Como o próprio slogan já diz, se trata de um framework javascript bem compacto, modular e orientado a objetos projetado para usuários intermediários e avançados de Javascript.

Com ele é possível escrever facilmente códigos robustos, flexíveis e que funcionem em qualquer navegador moderno, além de ter uma documentação muito bem estruturada e explicativa.

Dojo - http://dojotoolkit.org/

Dojo Framework

O Dojo pode ser baixado a partir do site oficial em diversas versões. Cada uma dessas versões pode ter determinadas partes da biblioteca do Dojo num arquivo Javascript, e permitir que você carregue outras partes da biblioteca dinamicamente usando o método de importação do Dojo. O Ajax é a versão mais popular do Dojo, e essa versão inclui suporte para operações assíncronas (para chamadas de AJAX), efeitos visuais, manipulações de eventos e as bibliotecas base do Dojo.

Script.aculo.us - http://script.aculo.us/

Script.aculo.us Framework

Criar efeitos visuais em Javascript ficaram bem mais fáceis com o esse framework. O script.aculo.us é uma biblioteca que permite a criação de efeitos de forma simples, sem precisar ser um desenvolvedor com conhecimentos avançados em Javascript.

Kendo UI - http://www.kendoui.com

Kendo UI Framework

O Kendo UI além de ser um framework javascript, também é HTML5. Com ele é possível criar sites modernos e robustos, além de aplicativos mobile.

Inserindo o código Javascript na página HTML

Os códigos **Javascript** podem ser inseridos na página HTML de duas formas:

Interno no documento: para inserir o código direto na estrutura do HTML, utilizamos as tags <script></script>, conforme mostra a Exemplo 1.

Exemplo 1: Inserindo código Javascript interno no HTML

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
      //código Javascript
    </script>
  </head>
  <body>
  </body>
</html>
```

Externo ao documento: o código Javacript também pode ser mantido em um arquivo separado do HTML. Para isso, deve-se referenciar tal arquivo na página, como vemos na Exemplo a seguir.

Exemplo 2: Referenciando código Javacript em arquivo externo

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript" src="meuArquivo.js"></script>
  </head>
  <body>
  </body>
</html>
```

O arquivo deve ser salvo com a extensão .JS e sua estrutura é a mesma utilizada quando o código é posto internamente no documento.

Cabe aqui uma observação importante: a tag <script> requer a tag de fechamento separada, não podendo ser fechada em si própria como <script type=.. />.

Sintaxe da linguagem Javascript

Javascript foi criada com base na ECMAScript e sua sintaxe é bastante semelhante a linguagens de alto nível muito utilizadas como C e Java, como veremos a seguir.

Usando variáveis

Essa linguagem possui tipagem dinâmica, ou seja, não é necessário definir o tipo das variáveis ao declará-las, para isso basta usar a palavra reservada “var”. A seguir temos alguns exemplos de utilização de variáveis.

Exemplo 3: Utilizando variáveis

```
var nome;
nome = “Fulano de Tal”;
var idade = 30;
idade = 30 + 20 - 10*5;
```

Trabalhando com funções

Javascript fornece também suporte a funções, aliado às facilidades da tipagem dinâmica, o que torna a definição de métodos simples e prática. Para criar funções, utilizamos a palavra reservada “function”.

As funções podem receber parâmetros e retornar valores, mas o tipo de retorno e o tipo dos parâmetros não precisa ser previamente definido. A seguir temos exemplos de funções com e sem parâmetros e retorno.

Exemplo 4: Função em Javascript sem parâmetro e sem retorno

```
function exibirMensagem()
{
alert(“Olá, seja bem vindo(a)!”);
}
```

Observação: a função alert será apresentada posteriormente, mas serve para exibir uma mensagem popup para o usuário.

Exemplo 5: Função em Javascript com parâmetro e com retorno

```
function somar(A, B)
{
return A + B;
}
```

Para definir o valor de retorno da função, deve-se utilizar a palavra reservada “return” seguida do valor ou expressão do resultado.

Estruturas de controle de fluxo

Assim como a maioria das linguagens de alto nível, **Javascript** possui estruturas condicionais e de repetição para controle de fluxo. A seguir temos a sintaxe e um exemplo de uso de tais estruturas.

Exemplo 6: Sintaxe da estrutura IF-ELSE

```
if(condição 1)
{
   //ação se condição 1 verdadeira
}
else if (condição 2)
{
   //ação se condição 2 verdadeira
}
else
{
   //ação se nenhuma das condições for verdadeira
}
```

O bloco ELSE pode ser omitido, caso apenas uma condição precise ser avaliada. A Exemplo 7 mostra um exemplo onde uma variável chamada idade é avaliada e, dependendo do seu valor, uma mensagem é exibida na tela.

Exemplo 7: Exemplo de uso da estrutura IF-ELSE

```
if(idade > 18)
{
alert(“É maior de idade”).
}
else
{
alert(“É menor de idade”);
}
```

Exemplo 8: Sintaxe da estrutura SWITCH

```
switch(variável)
{
case valor1:
//ações caso valor1
break;
case valor2:
//ações caso valor2
break;
case valor3:
//ações caso valor3
break;
default:
//ações caso nenhum dos valores
break
}
```

O comando switch verifica o valor de uma variável e, para cada uma das opções, executa um conjunto de ações. Se nenhum dos valores for verificado, os comandos do bloco default são executados.

O bloco default, porém, pode ser omitido caso não exista uma ação padrão a ser executada se nenhuma das opções for observada.

Exemplo 9: Exemplo de uso da estrutura SWITCH

```
switch(dia)
{
case 1:
alert(“Hoje é domingo”);
break;
case 2:
alert(“Hoje é segunda”);
break;
case 3:
alert(“Hoje é terça”);
break;
default:
alert(“Hoje não é nem domingo, nem segunda, nem terça”);
break
}
```

Exemplo 10: Sintaxe da estrutura WHILE

```
while(condição)
{
//ações
}
```

A estrutura de repetição while é usada para executar um conjunto de ações enquanto uma condição for verdadeira. Quando a condição se tornar falsa, o bloco é finalizado. A seguir temos um exemplo que exibe uma mensagem para o usuário enquanto uma variável for menor que 5.

Exemplo 11: Sintaxe da estrutura WHILE

```
var contador = 0;
while(contador < 5)
{
alert(“Olá”);
contador = contador + 1;
}
```

Uma outra estrutura muito semelhante é a DO-WHILE, que executa um bloco de ações até que uma condição seja falsa. Porém, essa condição é avaliada após a execução dos comandos, fazendo com que estes sejam executados pelo menos uma vez.

Exemplo 12: Sintaxe da estrutura DO-WHILE

```
do
{
   //ações
}
while(condição)
```

O mesmo exemplo da Exemplo 11, usando o DO-WHILE poderia ser representado como na Exemplo a seguir.

Exemplo 13: Exemplo de uso da estrutura DO-WHILE

```
var contador = 0;
do
{
alert(“Olá”);
contador = contador + 1;
}
while(contador < 5)
```

Por último, temos o comando FOR, que usa um contador para executar um bloco de ações uma determinada quantidade de vezes.

Exemplo 14: Sintaxe da estrutura FOR

```
for(inicialização; condição; complemento)
{
   //ações
}
```

O entendimento dessa estrutura fica mais fácil se observarmos um exemplo prático. Na Exemplo 15, uma variável “contador” é inicializada com o valor zero, e enquanto for menor que 10, o laço for deve ser executado.

Exemplo 15: Exemplo da estrutura FOR

```
var contador;
for(contador = 0; contador < 10; contador++)
{
   alert(contador);
}
```

Exibindo mensagens para o usuário com Javascript: Alert

A linguagem **Javascript** possui um número imenso de funções nativas que precisariam de vários artigos para serem explicadas detalhadamente. Aqui, serão mostradas apenas algumas das mais utilizadas por quem está iniciando os estudos.

A primeira função apresentada é a alert, que serve para exibir uma mensagem em uma janela popup. Essa função recebe apenas um parâmetro, que é o texto a ser exibido.

Exemplo 16: Exemplo da função alert

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <script type="text/javascript">
      alert("Olá, seja bem vindo ao Linha de Código.")
    </script>
  </head>
  <body>
  </body>
</html>
```

Salvando o conteúdo da Exemplo acima como um arquivo de extensão HTML, ao abri-lo no browser teríamos o como resultado uma mensagem como a mostrada na Figura 1.

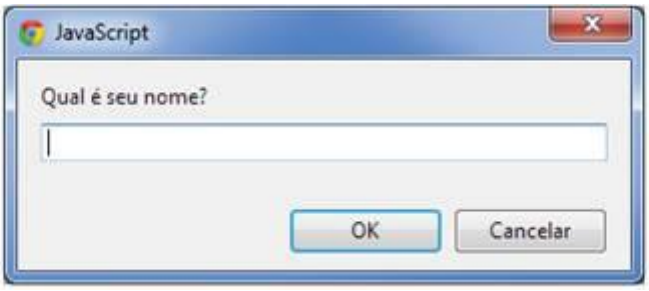
Exemplo de uso da função alert

Em seguida, temos a função prompt, que também abre uma janela popup, mas com uma caixa de texto para coletar informações do usuário. O retorno dessa função é o texto digitado.

Exemplo 16: Exemplo da função alert

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <script type="text/javascript">
      var nome;
      nome = prompt("Qual é seu nome?");
      alert("Olá, " + nome);
    </script>
  </head>
  <body>
  </body>
</html>
```

Nesse caso, o valor retornado pela função prompt foi atribuído a uma variável chamada "nome", que é utilizada em seguida na função alert para saudar o usuário.



Exemplo de uso da função prompt

Manipulando eventos

A linguagem **Javascript** também permite trabalhar com eventos dos elementos HTML como botões e caixas de texto. Eventos são disparados quando alguma ação é executada, como o clique e num botão ou a digitação de valores em um input.

No **código Javascript** pode-se atribuir valores aos eventos como se fossem propriedades, desde que o valor atribuído seja um código a ser executado. Por exemplo, a Exemplo a seguir mostra o código de uma página com um botão que, ao ser clicado, exibe uma mensagem (alert).

Exemplo 17: Exemplo de tratamento de evento

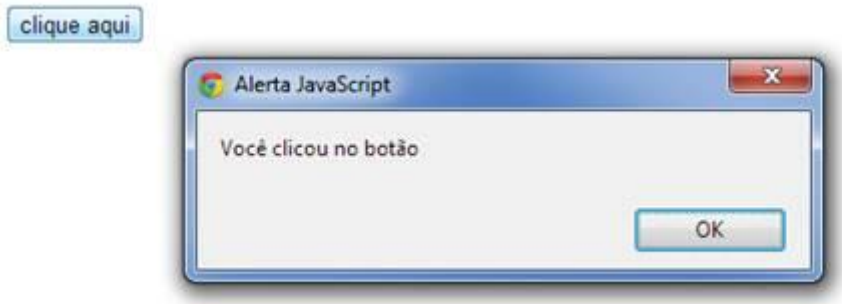
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
  </head>
  <body>
<button onclick="alert('Você clicou no botão');">clique aqui</button>
  </body>
</html>
```

Vale notar que nesse caso as tags <script> não foram necessárias, pois o código encontra-se dentro do próprio button. Outro ponto a ser observado é que foram usadas aspas duplas para definir o código e aspas simples no interior do mesmo, quando precisamos escrever um texto dentro de um evento.

Caso o código a ser executado no evento seja muito extenso, é possível criar uma função para isso e associá-la ao evento em questão. Basta informar, no lugar do código, o nome da função (com parâmetros, caso existam). Na Exemplo 18 temos um exemplo desse tipo.

Exemplo 18: Exemplo de tratamento de evento usando função

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <script type="text/javascript">
      function clique_botao()
      {
        alert("Você clicou no botão");
      }
    </script>
  </head>
  <body>
    <button onclick="clique_botao();">clique aqui</button>
  </body>
</html>
```



Tratamento de evento onclick do botão

O código executado na função, para fins didáticos, é o mesmo que o da Exemplo anterior, mas é fácil perceber que outras linhas poderiam ser inseridas para complementar o evento.

Agora foi preciso usar as tags <script>, pois o código encontra-se separado do elemento que o utilizará.

Conclusão

O **código Javascript** nos permite tornar uma página mais dinâmica, respondendo a interações do usuário. A sintaxe, bastante parecida com outras linguagens e muito intuitiva, facilita o aprendizado e utilização.

Antes de encerrar, cabe uma observação importante: alguns browsers bloqueiam a execução de scripts **Javascript**, pois estes podem ser utilizados de forma indevida para comprometer informações do usuário. Portanto, caso você tenha dificuldade para testar algum dos exemplos aqui apresentados ou outros códigos que venha a desenvolver, verifique as configurações do seu browser.

Concluímos aqui este artigo, cujo objetivo foi fazer uma breve introdução à linguagem de script do lado cliente mais utilizada atualmente: Javascript. Para garantir domínio dessa tecnologia, é preciso que o leitor busque outras fontes de informação como a documentação contida no site W3Schools.

Eventos em Javascript: Tratando eventos

Introdução

Eventos são procedimentos executados em consequência a uma ação. Por exemplo, quando o usuário clica em um botão, é disparado um evento deste elemento chamado “click”. Quando se pressiona uma tecla sobre outro elemento, é disparado um evento chamado “keydown” e quando a tecla é solta, o evento “keyup” é disparado.

Várias ações, não só na interface gráfica, fazem com que eventos sejam disparados. Então, sabendo que algo é feito quando certas ações são executadas, pode ser interessante e necessário ter controle sobre esta situação. Nesse ponto entra o TRATAMENTO DE EVENTOS, que consiste em definir o que será feito quando um determinado evento for disparado.

Algumas linguagens que possuem ambientes de desenvolvimento RAD (Rapid Application Development) permitem acessar esses eventos de forma bastante prática, através da interface gráfica do IDE. Porém, às vezes é preciso controlar esses eventos apenas pelo código, sem auxílio de ambiente gráfico, o que é muito comum para a linguagem Javascript. Por isso é importante conhecer a sintaxe da linguagem para esse tipo de ação.

O tratamento de eventos geralmente é feito a partir de funções nas quais se implementa o código que deve ser executado quando o evento ocorrer. Essas funções são chamadas de “event handlers” (tratadores de evento).

A seguir veremos como efetuar o tratamento de eventos de elementos como botões e inputs HTML a partir de um script Javascript, garantindo uma maior interação com o usuário.

Propriedades de eventos

Na linguagem HTML, os objetos possuem propriedades que dão acessos aos seus eventos. Essas propriedades têm o nome iniciando com “on”, seguido do nome do evento. Por exemplo, a propriedade “onclick” dá acesso ao evento click de um elemento.

O*bservação: apenas a nível de curiosidade, o “on” no nome dessas propriedades indica que elas permitem definir algo que vai ocorrer quando o evento ocorrer. Na tradução para o português temos que “on” significa “em/no/na”. Ou seja, algo que vai ocorrer “NO CLIQUE” do botão, ou “NO PRESSIONAMENTO DA TECLA” em um input.*

Um infográfico que apresenta algumas dessas propriedades pode ser encontrado em Atributos de tratamento de eventos da HTML5 - Infográfico.

Tratando eventos diretamente nas propriedades

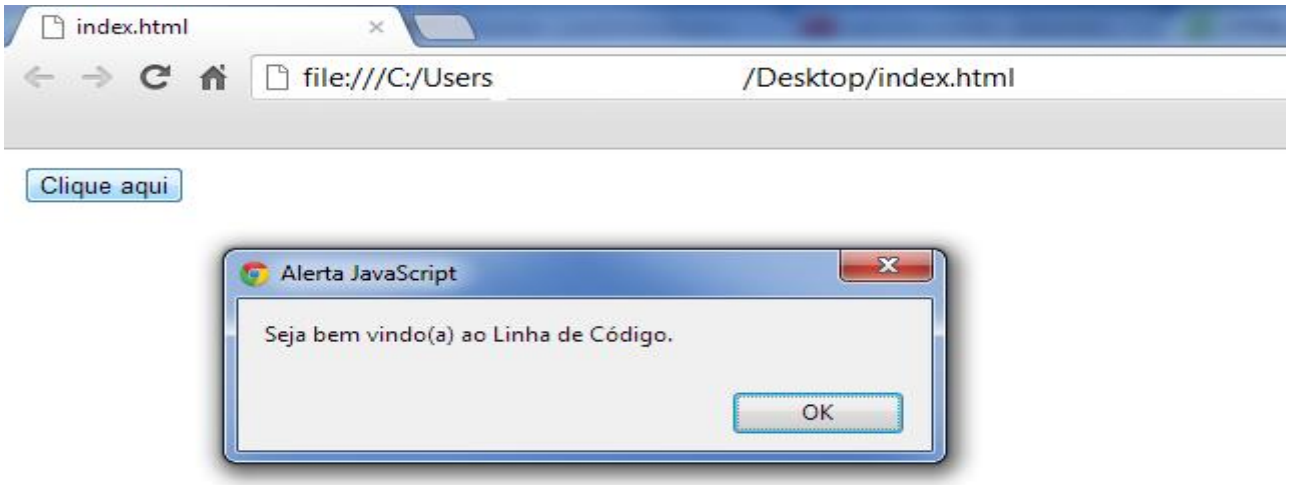
A linguagem Javascript permite que o código a ser executado em um evento seja informado diretamente na propriedade que lhe dá acesso. Essa forma é utilizada quando se tem poucas instruções a serem executadas e quando as expressões são curtas e de fácil compreensão.

Na Exemplo a seguir adicionamos um código à propriedade onclick de um botão para que seja exibida uma mensagem quando o usuário clicar sobre ele.

Exemplo 1: Tratando clique do botão na propriedade

```
<!doctype html>
<html>
<head>
   <meta charset="UTF-8"/>
</head>
<body>
   <button id="btn" onclick="alert('Seja bem vindo(a) ao Linha de Código.')">Clique aqui</button>
</body>
</html>
```

É importante notar que foram utilizadas aspas duplas para definir o valor da propriedade e aspas simples internamente. Essa é a sintaxe da HTML para essas situações, sempre que se necessitar usar aspas dentro do código, essas devem ser simples, pois as duplas são usadas externamente.



Código executado direto na propriedade do evento

Como vimos, apenas uma instrução foi executada e por isso foi simples e eficiente colocar o código direto na tag. Porém, caso existissem outras expressões a serem executadas, com cálculos e outros processamentos, se tornaria inviável escrever tudo no interior da propriedade. Para esses casos, veremos que é possível criar uma função (event handler) e chamá-la a partir da propriedade de evento.

Definindo event handlers

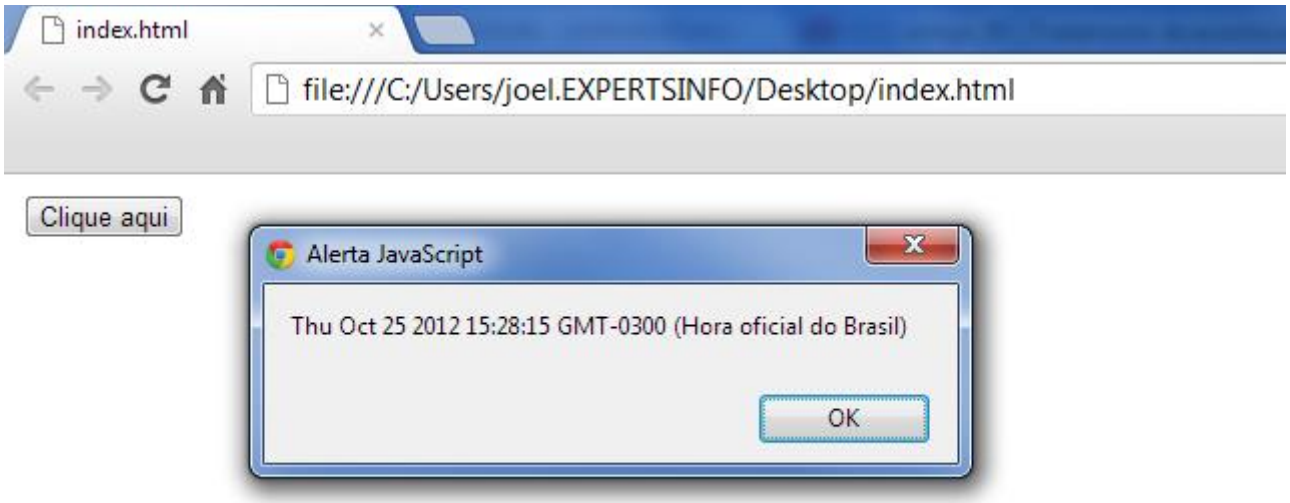
Os event handlers, como já foi dito, são funções que contém o código a ser executado na ocorrência de um evento. Em Javascript, podemos criar uma função utilizando a sintaxe padrão e fazer a chamada a essa função na propriedade de evento, informando seu nome e possíveis parâmetros no lugar onde se colocaria o código diretamente (caso da Exemplo 1).

Na Exemplo 2 criamos uma função chamada “exibirMensagem” que apresenta uma caixa de diálogo (alert) com a data atual. Nessa função foram usadas apenas duas linhas de código, mas fica claro que caso fosse preciso, várias outras instruções poderiam ser executadas.

Exemplo 2: Tratando o evento com uma função

```
<!doctype html>
<html>
<head>
   <meta charset="UTF-8"/>
   <script type="text/javascript">
   function exibirMensagem()
   {
      var data = new Date();
      alert(data.toString());
   }
   </script>
</head>
<body>
   <button id="btn" onclick="exibirMensagem();">Clique aqui</button>
</body>
</html>
```

Ao clicar no botão, a função é chamada e o resultado é o que se vê na Figura 2.



Código executando dentro de uma função

Tratando eventos com a função addEventListener

A função “addEventListener”, nativa da linguagem Javascript, permite ligar um evento de um objeto a uma função que fará seu tratamento. Usando essa função, não é preciso definir a propriedade de evento do objeto diretamente, pois isso será feito via código, dinamicamente. No exemplo a seguir, temos a mesma função exibir mensagem, mas dessa vez a ligamos ao evento click do botão usando a addEventListener.

Exemplo 3: Usando a função addEventListener

```
<!doctype html>
<html>
<head>
   <meta charset="UTF-8"/>
</head>
<body>
   <button id="btn">Clique aqui</button>

   <script type="text/javascript">
      function exibirMensagem()
      {
         var data = new Date();
         alert(data.toString());
      }

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

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

Nesse caso foi preciso usar a função “getElementById” para selecionar o objeto cujo evento seria tratado, no caso, o botão.

O primeiro parâmetro da addEventListener é o nome do evento que será tratado, nesse caso, “click”. O segundo é o nome da função que será executada.

Essa forma dá mais flexibilidade ao código, pois deixa o código HTML independente do Javascript. Todas as modificações necessárias são feitas no script, sem precisar alterar a tag.

Como inverter links ou textos com Javascript

Olá pessoal, vejo muita gente reclamar que quando vão fazer download em algum site ou blog, os organizadores colocam o endereço de url invertidos, obrigando os usuários a clicarem em seus parceiros, comprar créditos em celular, etc, o que é muito chato para o usuário.

Pensando nisso eu resolvi escrever esse artigo e ensinar os leitores do Linha de Código a criarem um sistema em javascript que inverte o ou qualquer coisa que colocarmos para inverter. Veremos que se trata de um sistema bem simples e muito, muito útil.

Chega de conversar e vamos logo ao que interessa não é mesmo?

Como todo projeto web, precisaremos criar em primeiro lugar o arquivo .html, que será a página do site.

Nota: No nosso exemplo vou utilizar o Sublime Text2 como editor html, css, javascript, etc, mas você pode ficar a vontade para usar qualquer editor html que queira, caso queira usar também o sublime text2, vou colocar o link de download dele aqui(http://www.sublimetext.com/2) , o editor é muito bom e eu recomendo. Agora vamos a criação do nosso documento, a estrutura padrão de qualquer documento html pode ser vista na Exemplo 1.

Exemplo 1: Estrutura padrão html

```
<html>
<head>
   <title></title>
</head>
<body>

</body>
</html>
```

Suponhamos aqui que todos já conheçam a estrutura padrão de uma página html e não iremos explicar o que cada uma das tag's fazem, vamos então começar a montar nossa página.

Exemplo 2: Estrutura da página html do exemplo

```
<html>
<head>
   <title>Invertendo links e textos - Linha de Código</title>
</head>
<body>
<h1>Inverter link ou palavra</h1>
<form>
<input type="text" name="txt" value="" />
<input type="button" id="btn" value="Inverter" onclick="inverter()" />
</form>
</body>
</html>
```

Agora nosso projeto começa a criar vida, inserimos um título na página e um lugar onde iremos inserir a url/texto para inverter e um botão para fazer a inversão.

Nossa página deve estar com essa aparência:

Nós temos basicamente a estrutura toda pronta, mas está faltando o principal, que é a ação de inverter o link, iremos pegar o que for inserido no campo de texto, inverter ele e exibir invertido para o usuário.

Para fazer essa inversão iremos usar o 3 funções nativas do javascript, primeiro vamos usar a função split() para separar a palavra em letras, após isso iremos usar a reverse() para inverter a ordem das letras e por fim usaremos a join() para juntar tudo novamente e exibir o resultado ao usuário.

É um código bem simples de se realizar e veremos como é simples.

Como sabemos podemos escrever um código javascript dentro do próprio código ou usar um arquivo externo. No nosso exemplo, para facilitar o estudo iremos escrever dentro do próprio código, mas ele precisará ficar dentro de uma tag script e essa tag terá de ser inserida dentro da tag head.

A tag script deve ser representada da seguinte forma:

Exemplo 2: Representação da tag script

```
<script type="text/javascript">
    /* Aqui o seu código javascript */
    </script>
```

Caso queira usar um arquivo externo, fique à vontade, mas você precisará do código da Exemplo 3 para linkar esses dois arquivos.

Exemplo 3: Linkando um arquivo javascript externo

```
<script type="text/javascript" src="link do seu arquivo javascript"></script>
```

Agora vamos colocar o código usado para inverter o link dentro de nossa tag script, confira Exemplo 4.

Exemplo 4: Código Javascript

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

    function inverter(){
        var inverter = document.getElementsByName("txt");
        valor = inverter.item(0).value.toString().split("");
        normal = valor.reverse().join("");
        document.body.innerHTML += normal + "<br />";
    }

</script>
```

Agora, ao colocar o link ou a palavra que queremos inverter no box de texto e clicar em inverter, ela será exibida embaixo invertido.

Explicando o código javascript, nós criamos uma função chamada inverter, essa será a função que será chamada quando clicarmos no botão, por isso o evento onclick="inverter()" no botão.

Pegamos o que foi inserido no input text e separamos em letras com o split(), após isso invertemos e juntamos com o reverse() e join() respectivamente.

O resultado obtido deve ser como o apresentado abaixo.

Agora qualquer coisa que você colocar no input text vai ser invertido, seja ele link ou texto. Abaixo vou colocar o código fonte completo do artigo, seu código deverá ficar dessa forma.

Exemplo 5: Código fonte do artigo

```
<html>
<head>
   <title>Invertendo links e textos - Linha de Código</title>
   <script type="text/javascript">

      function inverter(){
         var inverter = document.getElementsByName("txt");
         valor = inverter.item(0).value.toString().split("");
         normal = valor.reverse().join("");
         document.body.innerHTML += normal + "<br />";
      }

   </script>
</head>
<body>
<h1>Inverter link ou palavra</h1>
<form>
<input type="text" name="txt" value="" />
<input type="button" id="btn" value="Inverter" onclick="inverter()" />
</form>
</body>
</html>
```

Conhecendo o HTML5 Notifications API

HTML5 Notifications API

A API de Notificações permite exibir notificações ao usuário para eventos como novos e-mails, tweets ou eventos de calendário e também nas interações do usuário, independentemente de qual aba esteja aberta.

É importante ressaltar que essa funcionalidade não é nem um pouco cross-browser, ou seja, não funciona em todos os navegadores, somente naqueles que são adeptos ao webkit, como Google Chrome e o Firefox também aderiu à funcionalidade.

Mas afinal, o que é essa tal de notificação de fato? É simples, quem utiliza o GTalk em um desses navegadores que citei acima, já deve ter visto que sempre que alguém manda uma mensagem pelo chat e você não está com a aba do gmail ativa, você recebe uma notificação na tela dizendo que houve uma interação no chat, isso é HTML5 Notifications API.

Notificação do Gmail

Verificando compatibilidade do browser

Como vimos, nem todos os browsers(navegadores) suportam o uso da API de Notificação. Para isso, usamos um código pra verificar essa compatibilidade.

Exemplo 1: Verificando compatibilidade

```
// Verificando suporte a tecnologia
if (window.webkitNotifications) {
console.log('Seu browser suporta Notifications');
}
else {
console.log('Seu browser não suporta Notifications =(');
}
```

Estrutura da Notificação

Uma notificação é composta por um título (title), um corpo de texto (body), e uma imagem representativa(normalmente onde coloca-se a logo do site). Ao criá-la, você deve especificar esses atributos para que seja visualizada corretamente.

Para implementar as notificações é preciso que o usuário aceite o seu uso, essa ação deve ser feita pelo próprio usuário, na Exemplo 2 veremos como pedir essa autorização.

Basicamente essa autorização deverá conter um botão onde o usuário irá clicar onde para obter a autorização das notificações.

Exemplo 2: Pedindo autorização dos usuários - HTML

```
<input type="button" id="click-me" />
```

Exemplo 3: Código javascript

```
var Notifications = {
    requestPermission: function(callback) {
        window.webkitNotifications.requestPermission(callback);
    }
};

$(function() {
    $('#click-me').click(function() {
        Notifications.requestPermission(function() {
            alert('Permissão concedida');
        })
    });
});
```

Dessa forma, quando o usuário clicar no botão, será executada a ação de autorização das notificações.

Agora já temos quase tudo pronto, só está faltando enviar a notificação para o usuário, mas primeiro temos sempre que verificar se a permissão foi dada pelo usuário, na Exemplo 4 vemos como enviar essa notificação para o usuário.

Exemplo 4: Enviando notificação ao usuário

```
var Notifications = {
    requestPermission: function(callback) {
        window.webkitNotifications.requestPermission(callback);
    },

    showNotification: function(){
        // Verificando se a permissão já foi concedida
         if (window.webkitNotifications.checkPermission() > 0) {
                // Se não houver, volta a pedir permissão
                Notifications.requestPermission(function() {
                    Notifications.showNotification();
                });
         }
         else {
                // Se a permissão já foi concedida, cria a notificação e envia a mesma.
                var notification = window.webkitNotifications.createNotification("http://userserve-
ak.last.fm/serve/64s/318711.jpg", "Olá!", "Esta é a mensagem");
                notification.show();
         }
    }
};
```

Como podemos ver, o código é bem simples de se utilizar e de grande utilidade nos projetos. Agora, iremos ter um adicional, iremos ver uma lib de notificações, ou seja, um arquivo já pronto com a notificação, bastando apenas o usuário chamar essa notificação.

Para chamar a notificação é assim:

Exemplo 5: Chamando uma notificação em uma lib

```
Notifications.show("http://userserve-ak.last.fm/serve/64s/318711.jpg", "Aqui colocamos
mensagem de notificação");
```

Agora veremos o código da lib.

Exemplo 6: Lib de notificação

```
var Notifications = {
    apiAvailable: function() {
        if(window.webkitNotifications) {
            return true;
        } else {
            return false;
        }
    },

    isAuthorized: function() {
        if (!this.apiAvailable()) return false;

        return window.webkitNotifications.checkPermission() > 0 ? false : true;
    },

    authorize: function(callback) {
        var self = this;
        if (!this.apiAvailable()) return false;

        window.webkitNotifications.requestPermission(function() {
            if (self.isAuthorized()) {
                callback();
            }
        });
    },

    show: function(url, title, body) {
        if (!this.apiAvailable()) return false;

        var self = this;

        if (this.isAuthorized()) {
            var popup = window.webkitNotifications.createNotification(url, title, body);
            popup.show();
            setTimeout(function(){
                popup.cancel();
            }, 5000);
        } else {
            this.authorize(function() {
                //console.log(arguments);
                self.show(url, title, body);
            });
        }
    },

    checkForPermission: function() {
        if (!this.isAuthorized()) this.callForPermission();
    },

    callForPermission: function() {
```

```
        var authorizeBox = jQuery('<div />').addClass('notifications-authorize')
                                    .html('<p>Seu navegador possui suporte a notificações. Para solicitar uma permissão de notificação, clique no botão abaixo. Aperte "ALLOW" ou "PERMITIR" para a janela de notificação que irá aparecer. <input type="button" value="Ativar notificações" /></p>')

        jQuery('body').append(authorizeBox);

        jQuery('div.notifications-authorize input').click(function(){
            jQuery(this).remove();
            Notifications.authorize();
        });
    }
};

$(function() {
    Notifications.checkForPermission();
});
```

Como bloquear o botão CTRL e impedir impressão de página com Javascript

Como evitar que o usuário imprima a sua página utilizando o browser e como evitar que o mesmo usuário clique no botão Ctrl para salvar o que tem na sua página.

O grande desafio aqui não é bloquear tudo, é bloquear apenas o necessário. Geralmente o bloqueio é feito em partes do sistema, aquela parte que geralmente alguns perfis tem acesso com usuário e senha.

Aqui eu mostro como bloquear a impressão de tudo, mesmo utilizando o menu Arquivo>> Imprimir do browser ou clicando no botão de atalho Ctrl + P. Passando para a parte do botão Ctrl, dependendo do seu sistema é melhor adicionar uma página “full screen” ou “popup”, assim o usuário não tem como clicar Ctrl + N ou qualquer outra tecla para salvar a página. Quando o usuário clica no botão Ctrl, aparece pra ele uma mensagem falando que essa tecla está proibida e nada é feito.

A parte da impressão é feita utilizando CSS, enquanto que a parte do Ctrl é feita com JavaScript na página.

Qual a razão de utilizar essas duas funcionalidades? Bom, a minha razão é porque o cliente solicitou isso no sistema interno dele. Ele quer evitar que os outros usuário possam imprimir a página ou o documento específico. Se você leitor(a) quiser usar só a parte do CSS ou a só a parte do JavaScript, fique a vontade para fazer o que for melhor para você.

Primeiro passo

O primeiro passo é criar um arquivo chamado print.css, simples e fácil. Esse arquivo pode ser criado usando o notepad do Windows sem qualquer dificuldade. Eu resolvi chamar o arquivo de print.css pelo fato de que o arquivo é focado apenas para imprimir. O code 1.1 mostra o código desenvolvido.

Exemplo 1: Arquivo print.css

```
body {
}
#naoimprime
{
display: none;
}
#imprime
{
display:block;
}
```

Esse arquivo .css possui duas tags importantes, uma chamada “naoimprime” e a outra chamada “imprime”. Essas tags precisam ser chamadas de dentro do arquivo .html, .htm, .aspx, .jsf e tudo mais. Lembro que todo esse código funciona em qualquer linguagem e qualquer browser.

Para chamar essas tags da Exemplo 1, basta colocar o id = “naoimprime” ou id = “imprime”.

Segundo passo

O segundo passo envolve a criação de uma página em html ou qualquer extensão que está programando, chamando o arquivo .css e definindo o que pode e não pode imprimir. A Exemplo 2 mostra isso.

Exemplo 2: Página HTML

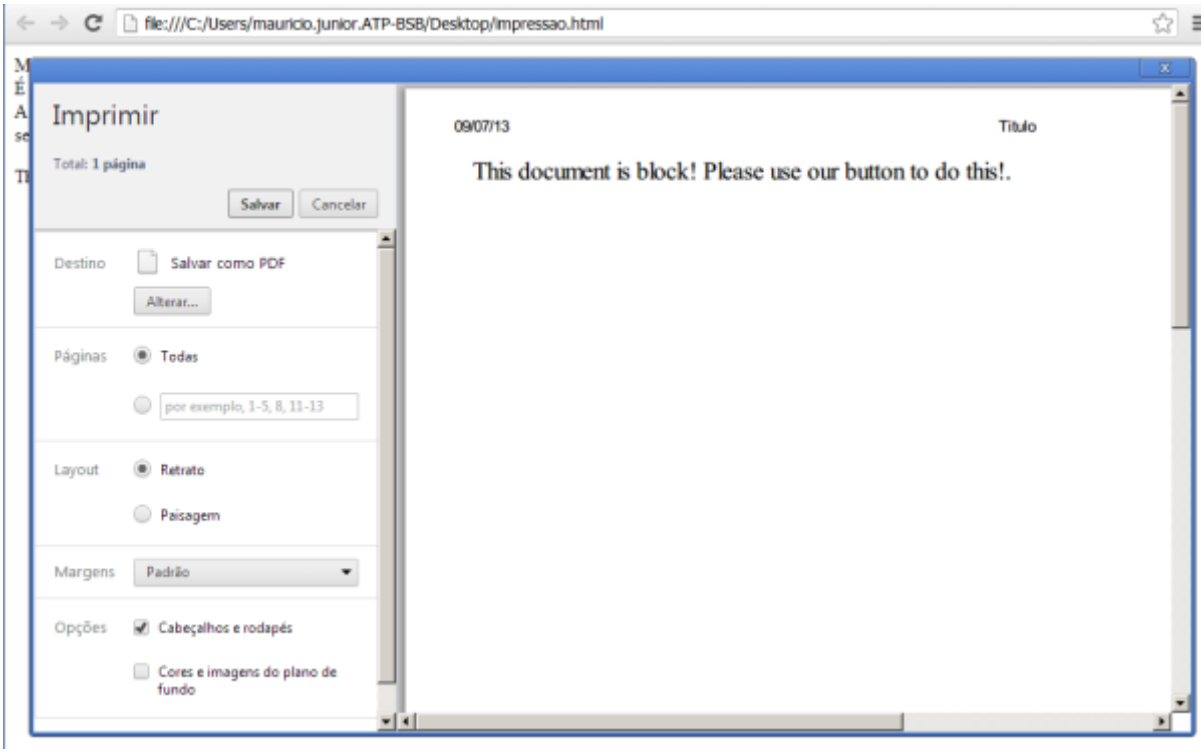
```
<html>
<head>
<title>Titulo</title>
<link rel="stylesheet" type="text/css" href="print.css" media="print" />
</head>
<body>
<div id="naoimprime">
Minha página com informações confidenciais.<br>
É necessário fazer bloquear o imprimir.<br>
A pessoa não pode imprimir esse documento, <br>
se for imprimir, é necessário clica no nosso botão primeiro
que faz o log.
</div>
<div id="imprime">
<p>
This document is block! Please use our button to do this!.
```

```
<br>
</p>
</div>
</body>
</html>
```

Na quarta linha da Exemplo 2 chama o arquivo print.css indicando a propriedade media=”print”. Depois da tag “body” existe a tag div com o id=”naoimprime”. Tudo que estiver dentro dessa div não vai imprimir. É dentro dela que você coloca o conteúdo da sua página ou sistema.

Depois que fechar a tag div, se achar necessário você pode colocar outra tag com o id=”imprime”. Tudo que estiver dentro dessa tag poderá ser impressa sem qualquer problema. Mesmo que usuário clique Ctrl + P ou acesse o endereço Arquivo >> Imprimir, tudo que será impresso é a mensagem falando que o documento está bloqueado. É assim que funciona o bloqueio de impressão sem qualquer problema.

O que acontece se o usuário tentar imprimir.



Terceiro passo

O terceiro passo envolve colocar o JavaScript dentro da página para bloquear o botão Ctrl. Se o usuário clicar nesse botão, uma mensagem aparece e não deixa que nada aconteça. Em alguns casos, é melhor usar um “popup” para evitar problemas; isso porque o usuário pode usar o menu do browser.

No geral a parte em JavaScript está pronta e você pode colocar no início da página.

Exemplo 3: Código Javascript

```
<SCRIPT LANGUAGE="JavaScript1.2">
function alerta(){
alert('A página não pode ser salva.');
return false;
}
function verificaBotao(oEvent){
var oEvent = oEvent ? oEvent : window.event;
var tecla = (oEvent.keyCode) ? oEvent.keyCode : oEvent.which;
if(tecla == 17 || tecla == 44|| tecla == 106){
alerta();
}
}
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.2">
document.onkeypress = verificaBotao;
document.onkeydown = verificaBotao;
document.oncontextmenu = alerta;
</script>
```

Note que a Exemplo 3 possui duas funções específicas. Essa função é colocada dentro da tag head da página html. A primeira função chama “alert”, onde a mensagem é exibida para o usuário. A segunda função chama “verificaBotao” onde recebe um evento. Esse evento detecta o número da tecla apertada e se for 17, 44 ou 106 então a mensagem aparece na tela.

O grande problema do JavaScript é fazer executar mesmo que não tiver dentro de um tempo “input”. Para resolver esse problema, foi atribuído no documento os eventos “onkeypress”, “onkeydown” e "oncontextmenu”, responsáveis pelo clique em qualquer parte da página. Isso faz com que, mesmo que a página não tenha nenhum campo “input”, se for colocado a tecla a mensagem aparece.

Para finalizar, a Exemplo 4 mostra toda página.

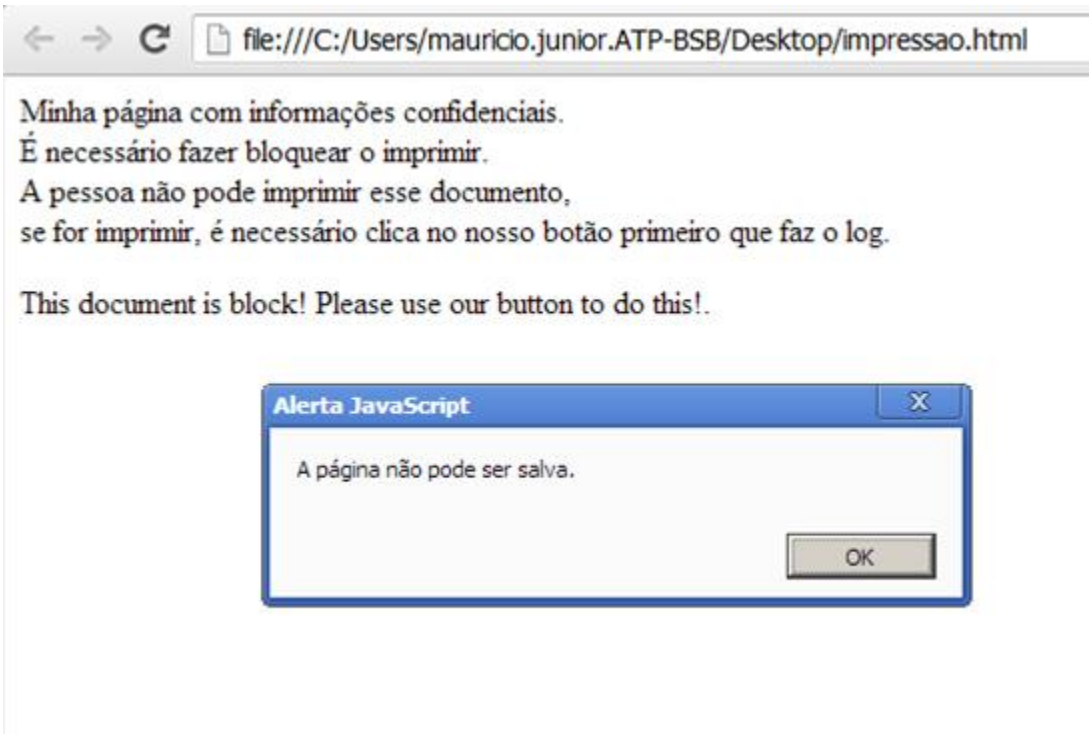
Exemplo 4: Código completo da página

```
<html>
<head>
<title>Titulo</title>
<link rel="stylesheet" type="text/css" href="print.css" media="print" />
<SCRIPT LANGUAGE="JavaScript1.2">
function alerta(){
alert('A página não pode ser salva.');
return false;
}
function verificaBotao(oEvent){
var oEvent = oEvent ? oEvent : window.event;
var tecla = (oEvent.keyCode) ? oEvent.keyCode : oEvent.which;
if(tecla == 17 || tecla == 44|| tecla == 106){
alerta();
}
}
</SCRIPT>
<SCRIPT LANGUAGE="JavaScript1.2">
document.onkeypress = verificaBotao;
document.onkeydown = verificaBotao;
document.oncontextmenu = alerta;
</script>
</head>
<body>
<div id="naoimprime">

Minha página com informações confidenciais.<br>
É necessário fazer bloquear o imprimir.<br>
A pessoa não pode imprimir esse documento, <br>
se for imprimir, é necessário clica no nosso botão primeiro
que faz o log.

</div>
<div id="imprime">
<p>
This document is block! Please use our button to do this!.
<br>
</p>
</div>
</body>
</html>
```

mostrando a mensagem na tela do usuário.



Conhecendo JSON

1. Introdução

Assim como o formato XML retornado pelo servidor em aplicações assíncronas temos também o JSON que é outro formato de dados. O formato JSON foi originalmente criado por Douglas Crockford e significa Javascript Object Notation. Sempre que quisermos enviar informações entre uma página web e um servidor precisaremos de alguma forma de formatar-las como texto puro, XML ou mesmo JSON que é mais uma maneira de enviar e retornar dados.

JSON é utilizado em grandes portais como Google e Yahoo. JSON é bastante indicado também para programadores front-end que não são acostumados a ler ou escrever em XML. JSON é muito mais simples de manipular pois é parecido com matrizes e listas que é bem comum em praticamente todas linguagens inclusive Javascript.

Todos desenvolvedores se perguntam qual formato de dados usar: texto puro, XML ou JSON. Texto puro é o menos indicado porque você precisará definir um formato próprio que só você entende e precisará manipular de forma primitiva eles, o XML é uma excelente opção sendo mais usado em estruturas de dados complexas e o JSON é bastante indicado em diversas situações incluindo aquelas em que trabalhamos com linhas de uma tabela ou matrizes.

2. JSON versus XML

Muitos dizem que o formato de dados JSON é muito superior ao formato de dados em XML principalmente quando se trata de Javascript e aplicativos assíncronos. Poucas pessoas conhecem JSON, no entanto muitos conhecem o XML que é um padrão reconhecido e bastante disseminado, no entanto XML é grande e por vezes difícil de lidar, por outro lado o XML pode manipular praticamente qualquer coisa e diversos tipos de dados. Quando fala-se em velocidade o JSON é mais rápido do que XML, porém mesmo assim o XML também é rápido, um não anula o outro mas JSON é estatisticamente mais rápido.

Outro ponto bastante discutido é que o JSON é fácil de trabalhar e manipular bastando conhecer matrizes, por outro lado XML possui uma sintaxe maior e bem definida podendo ser transformada para XSLT e manipulada em Web Services.

Apesar das diversas opções do XML o JSON se preocupa basicamente em passar informações de uma página Web para um servidor e, em seguida, fazer a trajetória inversa sem precisar de muito esforço, ferramentas extras ou percorrer uma árvore DOM. Portanto, para trabalharmos diretamente em Javascript de maneira leve e fácil utiliza-se o JSON. No entanto, no lado servidor os ambientes de desenvolvimento como Java, PHP, Perl, C#, etc precisam dar suporte ao JSON, hoje a maioria das linguagens já disponibiliza esse suporte através de APIs específicas.

Como pode-se ver existem vantagens e desvantagens em se trabalhar com uma das abordagens, cabe a você analisar cada uma delas e ver qual pesa mais para utilizar na sua aplicação. Abaixo falaremos um pouco mais sobre JSON.

3. Utilizando JSON no lado Cliente

JSON utiliza chaves ao invés de sinais de maior e menor utilizados no XML, mas pode armazenar os mesmos tipos de dados que o XML. Segue abaixo um exemplo de dados no JSON:

Exemplo 1: Exemplo de dados no JSON.

```
{“endereco”: [
  {
    “rua”: Leonor Viana,
    “cidade”: São Paulo,
    “estado”: SP
  }
]};
```

Assim como em XML o formato de dados do JSON também é bastante intuitivo. Para trabalhar com JSON usamos o Javascript comum não necessitando de nenhum modelo de objetos especial.

Para recuperar os dados JSON retornados pelo servidor poderíamos utilizar o código Javascript abaixo:

Exemplo 2: Exemplo de código Java script manipulando dados JSON.

```
var jsonData = eval(‘(‘ + request.responseText + ‘)’ );
var rua = jsonData.endereco[0].rua;
var cidade = jsonData.endereco[0].cidade;
var estado = jsonData.endereco[0].estado;
```

Veja que endereco[0] retorna o primeiro e único item endereço, se tivéssemos mais itens definiríamos um outro índice.

Utilizando JSON nota-se que não precisamos nos preocupar em manipular árvores DOM ou qualquer outra ferramenta de suporte, JSON cuida de tudo.

No código acima usamos a propriedade responseText para receber os dados do servidor em formato de texto puro, após isso convertemos para um objeto Javascript, por isso utilizamos a função eval acima.

4. Utilizando JSON no lado Servidor

Para criar e exibir dados JSON no lado servidor precisaremos de uma biblioteca para que possamos manipulá-lo. As bibliotecas JSON ajudam bastante na manipulação do JSON pois elas sabem exatamente como manipular essas informações. Entre as linguagens que suportam JSON temos C/C++, C#, ColdFusion, ActionScript, Java, Perl, PHP, ASP 3.0, Python, Ruby.

No PHP tem a biblioteca JSON.php que é bastante utilizada, enquanto que para Java existe a biblioteca JSON4J bastante popular entre os desenvolvedores Java.

As bibliotecas facilitam bastante o uso de JSON no lado servidor e para saber como manipular JSON numa linguagem especifica o desenvolvedor deverá estudar como a biblioteca escolhida manipula os dados JSON e que tipos de operações ela disponibiliza.

5. Conclusões

JSON é um formato de dados como XML, porém diferente de XML, utilizando JSON o desenvolvedor não precisará do DOM para trabalhar com dados JSON, ele possui sua própria forma através de matrizes e listas. JSON não é nada mais do que Javascript puro e podemos enviar dados em formato JSON para o servidor em nossas solicitações, assim como podemos enviar XML ou texto puro, porém o trabalho é feito de forma muito mais elegante e simples com JSON. No lado servidor se quisermos manipular dados JSON precisamos utilizar uma biblioteca para capturar os dados recebidos e converte-los em matrizes ou algum formato que ele puder usar. Ao escolher o formato de dados que se quer utilizar deve-se avaliar qual o melhor para as suas necessidades, normalmente utiliza-se XML ou JSON cada uma possuindo as suas vantagens e desvantagens como já foi discutido anteriormente.

jQuery Animate: Aprenda a fazer uma animação em jQuery

JQuery é uma ótima biblioteca do javascript que possui diversos métodos que ajudam muito a vida de qualquer desenvolvedor.

Um método bem interessante que pode ser muito bem explorado é o .animate(), ele controla controla propriedades do CSS para realizar animações em elementos HTML. Sendo assim seu uso se torna bem simples, e possibilita ao desenvolvedor criar desde pequenos efeitos até grandes animações.

Vou ensinar aqui a criar uma simples animação com duas bolas, simulando uma batendo na outra, assim conseguimos aprender como funciona este método.

Começamos com o código HTML já incluindo a biblioteca jQuery.

Exemplo 1: Código HTML

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script src="jquery.min.js" type="text/javascript"></script>
<title>Animate jQuery</title>
</head>

<body>

<div id="box">
    <div id="ball1"></div>
    <div id="ball2"></div>
    <div id="earth"></div>
</div>

</body>
</html>
```

Vamos precisar de quatro elementos. A #box vai servir para “segurar” os outros três, aí temos a #ball1 e a #ball2 que serão as duas bolas que vamos animar e a div #earth será base, como se fosse o chão.

Exemplo 2: Colocando o CSS

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script src="jquery.min.js" type="text/javascript"></script>
<title>Animate jQuery</title>
</head>

<style type="text/css">
#box{
    border:1px solid #512B11;
    height:200px;
    left:10px;
    position:relative;
    top:10px;
    overflow:hidden;
    width:600px;
}

#earth{
    background:#523723;
    border-top:20px solid #090;
    bottom:0;
    height:50px;
    position:absolute;
    width:600px;
}

#ball1, #ball2{
    background:#095fc6;
    border-radius:30px;
    height:50px;
    left:-50px;
    position:absolute;
    top:80px;
    width:50px;
}
#ball2{
    background:#282828;
    left:130px;
}
</style>
<body>
<div id="box">
    <div id="ball1"></div>
    <div id="ball2"></div>
    <div id="earth"></div>
</div>
</body>
</html>
```

Com a implementação do código CSS a cima conseguimos dar forma às coisas. Colocamos na #ball1 e #ball2 um border-radius para simular uma bola, e um position:absolute para que possamos controlar a posição delas na tela. Na div #earth não tem nada demais, só um background e um border-top para criar um desenho de uma superfície.

Exemplo 3: Código jquery

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script src="jquery.min.js" type="text/javascript"></script>
<title>Animate jQuery</title>
</head>
<script>
$(document).ready(function(){
    $("#ball1").animate({
     "left": "80px"
    }, 300, null, function(){
     $("#ball2").animate({
         "left": "200px"
     }, 400);
     $("#ball1").animate({
         "left": "75px"
     }, 300);
    });
});
</script>

<style type="text/css">
#box{
    border:1px solid #512B11;
    height:200px;
    left:10px;
    position:relative;
    top:10px;
    overflow:hidden;
    width:600px;
}

#earth{
    background:#523723;
    border-top:20px solid #090;
    bottom:0;
    height:50px;
    position:absolute;
    width:600px;
}

#ball1, #ball2{
    border-radius:30px;
    height:50px;
    position:absolute;
```

```
    top:80px;
    width:50px;
}
#ball1{
    background:#095fc6;
    left:-50px;
}
#ball2{
    background:#282828;
    left:130px;
}
</style>

<body>

<div id="box">
    <div id="ball1"></div>
    <div id="ball2"></div>
    <div id="earth"></div>
</div>

</body>
</html>
```

A animação se resume em duas partes, a segunda inicializa exatamente quando a primeira termina.

Antes de começarmos a ver como as animações foram feitas, vamos entender funciona o método animate.

Exemplo 4: Método .animate()

1.animate(properties, duration, easing, complete)

Em properties colocamos as propriedades CSS que definiram os movimentos da animação.

Em duration podemos inserir um valor em milisegundos, isso diz quanto tempo ela vai durar.

O easing é forma como as transições serão executadas, não vou me aprofundar aqui, mas existem bibliotecas que inserem outras opções de easing.

No lugar do complete podemos inserir uma função qualquer, que será executada assim que a animação terminar.

Primeiro movemos a div #ball1 para direita, fazemos isso alterando o valor da propriedade left para 50px, dentro do método animate. Como o valor inicial, definido no CSS, era -50px a animação vai realizar um movimento para a direita. Essa animação irá durar 300 milisegundos e não terá nenhum tipo de easing.

Assim que a primeira animação terminar invocamos uma função anónima responsável por comportar as próximas duas animações.

No segundo passo movemos a outra div #ball2 para direita, alterando a propriedade left para 200px. Da mesma forma que no passo anterior, tínhamos também um valor inicial para left, que era 130px, aumentando esse valor o elemento vai se mover para a direita. Logo embaixo fazemos com que a primeira div animada, #ball1, recue um pouco para traz, simulando uma batida, mudando novamente o valor da propriedade left para 75px.

Tipos de Eventos

Existem diversos eventos que podem ser utilizados para que a interação do usuário com a página seja o ponto de disparo de funções que alteram os elementos da própria página:

- ✓ onclick: clica com o mouse
- ✓ ondblclick: clica duas vezes com o mouse
- ✓ onmousemove: mexe o mouse
- ✓ onmousedown: aperta o botão do mouse
- ✓ onmouseup: solta o botão do mouse (útil com os dois acima para gerenciar drag'n'drop)
- ✓ onkeypress: ao pressionar e soltar uma tecla
- ✓ onkeydown: ao pressionar uma tecla.
- ✓ onkeyup: ao soltar uma tecla. Mesmo acima.
- ✓ onblur: quando um elemento perde foco
- ✓ onfocus: quando um elemento ganha foco
- ✓ onchange: quando um input, select ou textarea tem seu valor alterado
- ✓ onload: quando a página é carregada
- ✓ onunload: quando a página é fechada
- ✓ onsubmit: disparado antes de submeter o formulário. Útil para realizar validações

Existem também uma série de outros eventos mais avançados que permitem a criação de interações para drag-and-drop, e até mesmo a criação de eventos customizados.

Funções temporais

Em JavaScript, podemos criar um *timer* para executar um trecho de código após um certo tempo, ou ainda executar algo de tempos em tempos.

A função setTimeout permite que agendemos alguma função para execução no futuro e recebe o nome da função a ser executada e o número de milissegundos a esperar:

```
// executa a minhaFuncao daqui um segundo
setTimeout(minhaFuncao, 1000);
```

Se for um código recorrente, podemos usar o setInterval que recebe os mesmos argumentos mas executa a função indefinidamente de tempos em tempos:

```
// executa a minhaFuncao de um em um segundo
setInterval(minhaFuncao, 1000);
```

É uma função útil para, por exemplo, implementar um banner rotativo, como faremos no exercício a seguir.

clearInterval

As funções temporais devolvem um objeto que representa o agendamento que foi feito. É possível usá-lo para cancelar a execução no futuro. É especialmente interessante para o caso do *interval* que pode ser cancelado de sua execução infinita:

```
// agenda uma execução qualquer
var timer = setInterval(minhaFuncao, 1000);

// cancela execução
clearInterval(timer);
```

Exercício: banner rotativo

1. Implemente um banner rotativo na home page da Mirror Fashion usando JavaScript.

   Temos duas imagens, a **destaque-home.png** e a **destaque-home-2.png** que queremos trocar a cada 4 segundos; use o setInterval para isso.

   Há várias formas de implementar essa troca de imagens. Uma sugestão é manter um array com os valores possíveis para a imagem e um inteiro que guarda qual é o banner atual.

```
var banners = ["img/destaque-home.png", "img/destaque-home-2.png"];
var bannerAtual = 0;

function trocaBanner() {
  bannerAtual = (bannerAtual + 1) % 2;
  document.querySelector('.destaque img').src = banners[bannerAtual];
}

setInterval(trocaBanner, 4000);
```

2. (opcional, avançado) Faça um botão de *pause* que pare a troca do banner.

   Dica: use o clearInterval para interromper a execução.

3. (opcional, avançado) Faça um botão de *play* para reativar a troca dos banners.

Para saber mais: sugestão para o desafio de pause/play

Podemos criar no HTML um novo link para controlar a animação:

```
<a href="#" class="pause"></a>
```

O JavaScript deve chamar clearInterval para pausar ou novamente o setInterval para continuar a animação.

Precisamos **editar** o código anterior que chamava o setInterval para pegar o seu retorno. Será um objeto que controla aquele interval e nos permitirá desligá-lo depois:

```
var timer = setInterval(trocaBanner, 4000);
```

Com isso, nosso código que controla o pause e play ficaria assim:

```
var controle = document.querySelector('.pause');

controle.onclick = function() {
  if (controle.className == 'pause') {
    clearInterval(timer);
    controle.className = 'play';
  } else {
```

```
    timer = setInterval(trocaBanner, 4000);
    controle.className = 'pause';
  }

  return false;
};
```

Por fim, podemos estilizar o botão como pause ou play apenas trabalhando com bordas no CSS:

```
.destaque {
  position: relative;
}
.pause,
.play {
  display: block;
  position: absolute;
  right: 15px;
  top: 15px;
}
.pause {
  border-left: 10px solid #900;
  border-right: 10px solid #900;
  height: 30px;
  width: 5px;
}
.play {
  border-left: 25px solid #900;
  border-bottom: 15px solid transparent;
  border-top: 15px solid transparent;
}
```

Para saber mais: vários callbacks no mesmo elemento

Nos exercícios que trabalhamos com eventos, usamos o onclick e o onsubmit diretamente no elemento que estávamos manipulando:

```
document.querySelector('#destaque').onclick = function() {
  // tratamento do evento
};
```

É uma forma fácil e portável de se tratar eventos, mas não muito comum na prática. O maior problema do código acima é que só podemos atrelar uma única função ao evento. Se tentarmos, em outra parte do código, colocar uma segunda função para executar no mesmo evento, ela sobrescreverá a anterior.

A maneira mais recomendada de se associar uma função a eventos é com o uso de addEventListener:

```
document.querySelector('#destaque').addEventListener('click', function() {
  // tratamento do evento
});
```

Dessa maneira, conseguimos adicionar vários listeners ao mesmo evento, deixando o código mais flexível. Só há um porém: embora seja o modo oficial de se trabalhar com eventos, o addEventListener não é suportado do IE8 pra baixo.

Para atender os IEs velhos, usamos a função attachEvent, semelhante:

```
document.querySelector('#destaque').attachEvent('onclick', function() {
  // tratamento do evento
});
```

O problema é ter que fazer sempre as duas coisas para garantir a portabilidade da nossa página.

Glossário

- ✓ **Classe**: Conjunto de propriedades (atributos) e ações (métodos) que serão manipulados durante a execução do programa.
- ✓ **Método**: “Ações” que são desenvolvidas na classe que serão implementadas e manipuladas durante a execução do programa. Atributo: Propriedade ou característica de um objeto.
- ✓ **HTML**: Hyper-Text Markup Language. Linguagem de marcação de hiper-texto. É a linguagem usada para criação de páginas web.
- ✓ **Javascript**: Linguagem de programação client-side desenvolvida pela NetScape muito utilizada em páginas HTML.
- ✓ **DHTML**: Dynamic Hyper-Text Markup Language. É o uso do HTML com o auxílio de uma linguagem de programação client-side (Nesse caso, Javascript) de forma a deixar a página HTML mais dinâmica.
- ✓ **Console**: Video-game.
- ✓ **Desktop**: PC’s de casa, não-móveis ou Computadores de Mesa.
- ✓ **Elemento HTML**: Pode-se considerar como um objeto da página e são definidos pelas tags HTML.

Bibliografia

JSON. Disponível em www.json.org/
W3C XML. Disponível em http://www.w3schools.com/xml/
Brett McLaughlin. Head Rush Ajax. O‘Reilly, 2006.
DOM, W3C. Disponível em www.w3.org/DOM/