Unidade 7: PopUp e Validação de Formulário
com CSS/JavaScript

Prof. Daniel Caetano

Objetivo: Capacitar o aluno para o uso de javascript para manipular o CSS e na validação de dados client-side.

Bibliografia: W3,2009; MCLAUGHLIN,2008; MOZILLA,2009.

INTRODUÇÃO

Conceitos Chave:
- JavaScript...
  * Como usar isso para algo realmente útil?
  * Manipulação do CSS
  * Validação de Formulários Client-Side!
- Vantagens de manipulação client-side
  * Minimiza tráfego
  * Rápida resposta ao usuário
- Desvantagens
  * Não é confiável
  * Só funciona quando o navegador possui e habilita o JavaScript

Na aula anterior foram apresentados muitos aspectos que podem ser modificados com o uso do JavaScript. Entretanto, foram apresentadas apenas algumas funções "cosméticas" para o JavaScript.

O uso do JavaScript, entretanto, pode ter uma aplicação muito eficiente e prática: manipular o CSS e validar dados digitados pelo usuário ainda no navegador, isto é, produzir modificações no documento sem precisar enviá-las para o servidor.

Validar dados significa verificar se estes dados estão dentro de parâmetros aceitáveis para a aplicação.

O fato de não enviar estes dados para o servidor traz vantagens bastante relevantes. Consideremos o caso onde não há validação no lado do cliente e suponhamos que o usuário digite uma informação incorreta; por exemplo, cometeu um erro ao digitar seu CPF. Num modelo tradicional, os dados seriam enviados para o servidor como o usuário digitou;

chegando lá, o servidor verificaria o erro e teria de enviar novamente a página solicitando o correto preenchimento dos dados. O usuário teria então de corrigí-los e re-enviar a informação. Caso existam muitos erros, este processo teria que se repetir várias vezes, até que todos os erros fossem contornados.

Além de muito tráfego, a solução acima é lenta e desagradável para o usuário. Com o uso de JavaScript, podemos fazer a verificação sem mandar os dados para o servidor, o que significa menos tráfego e respostas muito mais rápidas ao cliente, de acordo com o esperado de uma aplicação para Internet Rica.

Por outro lado, por uma série de fatores, essa verificação pode falhar. Por exemplo: o navegador do usuário pode estar com o JavaScript desligado. Ou mesmo um hacker pode ter feito um programa para simular um envio de sua página, sem as devidas verificações. Em ambos os casos, as informações que chegariam ao servidor não são mais confiáveis. Por esta razão, o uso de JavaScript não irá nos livrar de checar as informações quando elas chegarem ao servidor; por outro lado, em situações normais os dados chegarão corretos e o servidor jamais terá de solicitar novamente o preenchimento, o que nos permite atingir aos objetivos desejados.

1. POPUPS COM JAVASCRPIT E CSS

A proposta é criar o efeito apresentado na figura 1:

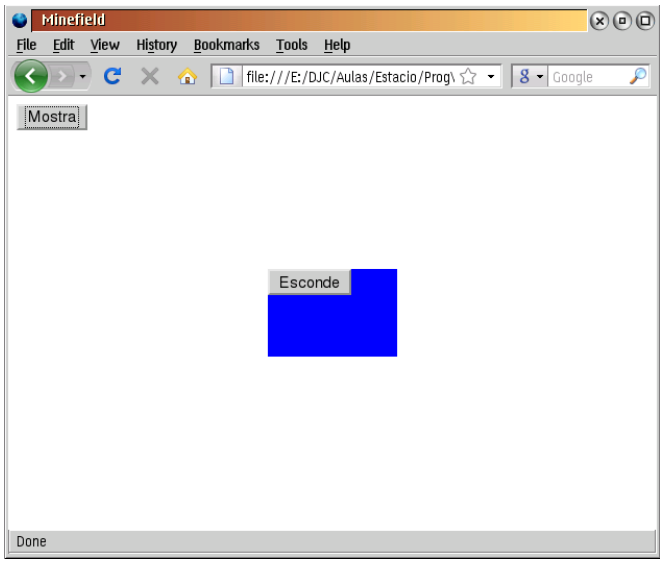

Figura 1: Observe a janela pop-up em azul

Essa região azul deve aparecer quando o botão "mostra" é clicado e deve sumir quando o botão "esconde" é clicado.

Comecemos com o HTML que gere a imagem da figura 1. Este é simples:

popup.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-BR">
<head>
	<meta http-equiv="Content-Type" content="text/html;charset=iso-8859-1" />
	<title>Teste de JavaScript</title>
	<link href="popup.css" rel="stylesheet" type="text/css" />
</head>
<body>
	<p><input type="button" value="Mostra" id="b1" /></p>
	<div id="d1"><input type="button" value="Esconde" id="b2" /></div>
</body>
</html>
```

Observe que já foi adicionado o link para o arquivo de estilo **popup.css**, que ainda não foi criado... e também já foi definido o DIV para a região popup, que conterá o botão "esconde". Vamos criar o arquivo CSS inicial:

popup.css

```
#d1 {
	position:absolute;
	left: 40%;
	width: 20%;
	top: 40%;
	height: 20%;
	background-color: blue;
	}
```

Isso já resulta em uma imagem com a aparência previamente indicada na figura 1, reproduzida abaixo:

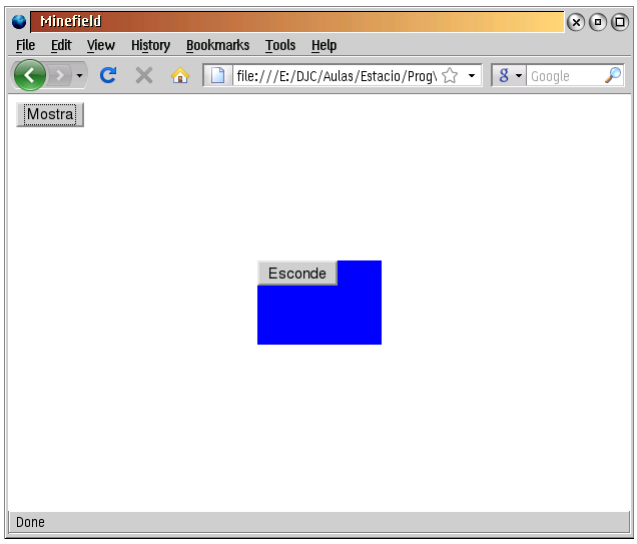

Figura 2: O resultado do código apresentado

Por outro lado, apertar os botões não produz resultado algum... mas antes de definirmos ações para os eventos de pressionar o botão, vamos dar um último retoque na figura: vamos esconder o popup, pois ele só deve aparecer quando o botão "Mostra" for pressionado. Para isso, usaremos o seguinte atributo no CSS:

popup.css

```
#d1 {
        position:absolute;
        left: 40%;
        width: 20%;
        top: 40%;
        height: 20%;
        background-color: blue;
        visibility: hidden;
        }
```

Recarregue a página e observe o resultado, que deve ser o da figura 3.

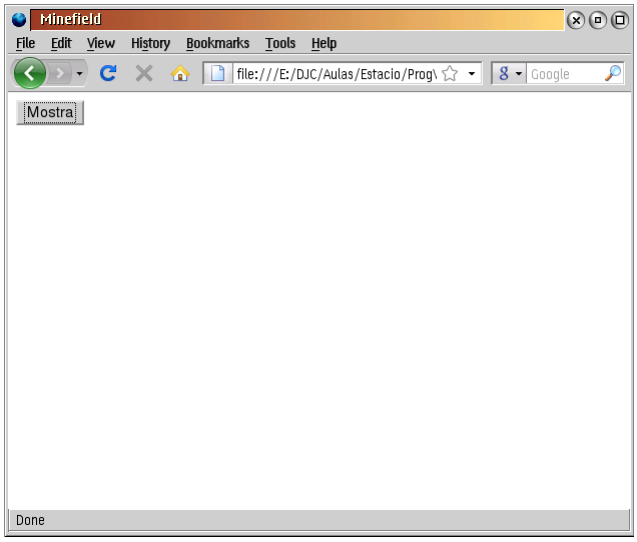

Figura 3: O popup sumiu!

O que nos fizemos foi desligar a apresentação da região D1. Assim, ela **está** na página, mas não está sendo mostrada. O nosso objetivo será fazer com que o estado de apresentação sejam mudado quando o usuário clicar no botão "Mostra".

Para realizar esta tarefa, precisaremos associar um código JavaScript ao nosso HTML. Chamaremos este código de **popup.js**, e ele será indicado no HTML como se segue.

popup.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-BR">
<head>
	<meta http-equiv="Content-Type" content="text/html;charset=iso-8859-1" />
	<title>Teste de JavaScript</title>
	<script type="text/javascript" src="popup.js"></script>
	<link href="popup.css" rel="stylesheet" type="text/css" />
</head>
<body>
	<p><input type="button" value="Mostra" id="b1" /></p>
	<div id="d1"><input type="button" value="Esconde" id="b2" /></div>
</body></html>
```

Todo o código JavaScript ficará armazenado no arquivo **popup.js**. A primeira coisa que indicaremos neste arquivo será a função de configuração dos controles, e associá-la ao evento **window.onload**, que é processado depois que **toda** a página foi carregada.

popup.js

```
/* Associa a função configura() ao evento onload da janela (window) */
window.onload = configura;	/* SEM PARENTESES! */

/* Região onde definiremos o código da função configura() */
function configura() {
	/* Aqui entrará o código de configuração */
	}
```

Podemos testar o nosso JavaScript, para verificar se a função configura está sendo chamada corretamente. Para isso, mudaremos a cor de fundo para **preto** dentro da função *configura*:

popup.js

```
/* Associa a função configura() ao evento onload da janela (window) */
window.onload = configura;	/* SEM PARENTESES! */

/* Região onde definiremos o código da função configura() */
function configura() {
	/* Teste: muda a cor do  fundo do corpo para preto */
	document.body.style.backgroundColor = "black";
	}
```

Se o fundo ficou preto, quer dizer que fizemos tudo certo até aqui. Mas a função configura não foi criada para mudar a cor do fundo, foi?

IMPORTANTE: Se a tela ficou preta assim que foi carregada é porque você provavelmente colocou parênteses depois do nome "configura", na linha do *window.onload*. O parênteses indica para o JavaScript que uma função *deve ser executada imediatamente* e não é isso que queremos. Por isso é importante se lembrar: sempre que vamos associar uma função a um evento, o nome da função vem **sem parênteses**!

Relembrando, a razão de existir da função **configura** é definir quais funções do JavaScript serão acionadas quando alguns eventos ocorrerem na página. Em nossa página, temos dois botões: o **b1**, que tem o texto "Mostra" e o **b2**, que tem o texto "Esconde".

Assim, uma das coisas que temos de fazer é indicar que, quando o botão **b1** for apertado (evento *onclick*), uma função que mostra o *pop up* seja acionada. Um bom nome para esta função é ***mostraPopUp***.

Para conseguir isso, precisamos pegar o elemento do botão "Mostra" pela ID dele, que sabemos ser **b1** e guardar o nome da função **mostraPopUp** no evento **onclick**. Isso está indicado abaixo:

popup.js

```
/* Associa a função configura() ao evento onload da janela (window) */
window.onload = configura;        /* SEM PARENTESES! */

/* Região onde definiremos o código da função configura() */
function configura() {
        /* Associa a função mostraPopUp ao evento onclick do botão de ID = b1 */
        document.getElementById ("b1").onclick = mostraPopUp;/* SEM PARENTESES! */
        }
```

Mas, se carregarmos a página e clicarmos no botão... nada acontecerá! Será que associamos corretamente? Sim, associamos! O navegador já entendeu que, quando for clicado no botão b1, a função chamada **mostraPopUp** deve ser executada. O problema é que *ainda não definimos a função* ***mostraPopUp***! Sem isso, o navegador não sabe o que fazer!

Assim, vamos definir a função **mostraPopUp** como se segue, com aquele mesmo código de pintar o fundo de preto, para verificar se as coisas estão funcionando.

popup.js

```
/* Associa a função configura() ao evento onload da janela (window) */
window.onload = configura;        /* SEM PARENTESES! */

/* Região onde definiremos o código da função configura() */
function configura() {
        /* Associa a função mostraPopUp ao evento onclick do botão de ID = b1 */
        document.getElementById ("b1").onclick = mostraPopUp;      /* SEM PARENTESES! */
        }
```

```
/* Região onde definiremos o que acontece quando a função mostraPopUp() for chamada */
function mostraPopUp() {
        /* Teste: muda a cor do  fundo do corpo para preto */
        document.body.style.backgroundColor = "black";
        }
```

Se, depois disso, carregarmos a página e apertarmos o botão, a tela deve ficar toda preta, indicando que a função **mostraPopUp** foi chamada corretamente!

Porém, o nosso objetivo com o botão não era, mais uma vez, mudar a cor do fundo para preta, e sim pegar o elemento que tem o ID igual a **d1** (o nosso DIV) e mudar o seu estilo de visibilidade para "visible", isto é, visível. Isso está indicado no código a seguir.

popup.js

```
/* Associa a função configura() ao evento onload da janela (window) */
window.onload = configura;        /* SEM PARENTESES! */

/* Região onde definiremos o código da função configura() */
function configura() {
        /* Associa a função mostraPopUp ao evento onclick do botão de ID = b1 */
        document.getElementById ("b1").onclick = mostraPopUp;      /* SEM PARENTESES! */
        }

/* Região onde definiremos o que acontece quando a função mostraPopUp() for chamada */
function mostraPopUp() {
        /* Muda estilo de visibilidade do elemento de ID = d1 para "visible" */
        document.getElementById ("d1").style.visibility = "visible";
        }
```

Experimente recarregar a página e apertar o botão "Mostra" agora! Teste! O que aconteceu?

Se tudo correu bem, a janelinha azul com o botão "Esconde" deve ter aparecido, como indicado na figura 4.

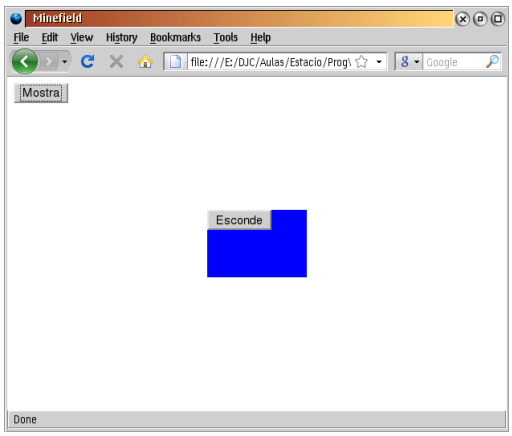

Figura 4: A janela Pop Up apareceu!

Se clicarmos no botão "Esconde", entretanto, a janelinha continua lá, nada acontece. O que está ocorrento é que não associamos nenhuma função ao evento *onclick* do botão b2, ou seja, do botão "Esconde". Como desejamos que o PopUp seja escondido, um bom nome para esta função é escondePopUp.

Para fazer essa associação, dentro da função **configura**, precisamos pegar o elemento do botão "Mostra" pela ID dele, que sabemos ser **b2** e guardar o nome da função **escondePopUp** no evento **onclick**. Isso está indicado abaixo:

```
/* Associa a função configura() ao evento onload da janela (window) */
window.onload = configura;        /* SEM PARENTESES! */

/* Região onde definiremos o código da função configura() */
function configura() {
        /* Associa a função mostraPopUp ao evento onclick do botão de ID = b1 */
        document.getElementById ("b1").onclick = mostraPopUp;      /* SEM PARENTESES! */

        /* Associa a função escondePopUp ao evento onclick do botão de ID = b2 */
        document.getElementById ("b2").onclick = escondePopUp;          /* SEM PARENTESES! */
        }

/* Região onde definiremos o que acontece quando a função mostraPopUp() for chamada */
function mostraPopUp() {
        /* Muda estilo de visibilidade do elemento de ID = d1 para "visible" */
        document.getElementById ("d1").style.visibility = "visible";
        }
```

Agora já sabemos que, se recarregarmos a página, o botão "Esconde" continua sem funcionar, porque a função **escondePopUp** ainda não foi definida. Mas não acredite só porque você esta lendo. Recarregue a página e experimente!

Assim, o próximo passo é criar essa **escondePopUp**, mas desta vez vamos "pular" a etapa de pintar o fundo de preto. Assim, vamos pegar o elemento que tem o ID igual a **d1** (o nosso DIV) e mudar o seu estilo de visibilidade para "hidden", isto é, escondido. Isso está indicado no código a seguir.

popup.js

```
/* Associa a função configura() ao evento onload da janela (window) */
window.onload = configura;        /* SEM PARENTESES! */

/* Região onde definiremos o código da função configura() */
function configura() {
        /* Associa a função mostraPopUp ao evento onclick do botão de ID = b1 */
        document.getElementById ("b1").onclick = mostraPopUp;      /* SEM PARENTESES! */

        /* Associa a função escondePopUp ao evento onclick do botão de ID = b2 */
        document.getElementById ("b2").onclick = escondePopUp;     /* SEM PARENTESES! */
        }
```

```
/* Região onde definiremos o que acontece quando a função mostraPopUp() for chamada */
function mostraPopUp() {
	/* Muda estilo de visibilidade do elemento de ID = d1 para "visible" */
	document.getElementById ("d1").style.visibility = "visible";
	}

/* Região onde definiremos o que acontece quando a função escondePopUp() for chamada */
function escondePopUp() {
	/* Muda estilo de visibilidade do elemento de ID = d1 para "hidden" */
	document.getElementById ("d1").style.visibility = "hidden";
	}
```

Experimente recarregar a página e apertar o botão "Mostra" e depois o botão "Esconde"! O que aconteceu? Se tudo correu bem, a janela azul deve ter sumido! O Pop Up está pronto! A seguir, o código completo da página:

popup.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-BR">
<head>
	<meta http-equiv="Content-Type" content="text/html;charset=iso-8859-1" />
	<title>Teste de JavaScript</title>
	<script type="text/javascript" src="popup.js"></script>
	<link href="popup.css" rel="stylesheet" type="text/css" />
</head>
<body>
	<p><input type="button" value="Mostra" id="b1" /></p>
	<div id="d1"><input type="button" value="Esconde" id="b2"></div>
</body>
</html>
```

popup.js

```
/* Associa a função configura() ao evento onload da janela (window) */
window.onload = configura;	/* SEM PARENTESES! */

/* Região onde definiremos o código da função configura() */
function configura() {
	/* Associa a função mostraPopUp ao evento onclick do botão de ID = b1 */
	document.getElementById ("b1").onclick = mostraPopUp;	/* SEM PARENTESES! */

	/* Associa a função escondePopUp ao evento onclick do botão de ID = b2 */
	document.getElementById ("b2").onclick = escondePopUp;	/* SEM PARENTESES! */
	}

/* Região onde definiremos o que acontece quando a função mostraPopUp() for chamada */
function mostraPopUp() {
	/* Muda estilo de visibilidade do elemento de ID = d1 para "visible" */
	document.getElementById ("d1").style.visibility = "visible";
	}
```

```
/* Região onde definiremos o que acontece quando a função escondePopUp() for chamada */
function escondePopUp() {
	/* Muda estilo de visibilidade do elemento de ID = d1 para "hidden" */
	document.getElementById ("d1").style.visibility = "hidden";
	}
```

2. CARREGANDO CSS ESPECÍFICO PARA INTERNET EXPLORER

Algumas vezes precisamos de mais de um arquivo CSS, um deles genérico e outro com correções específicas para o Internet Explorer (ou outro navegador qualquer). É importante lembrar que ***o ideal é não precisar destes truques***, mas caso seja necessário, vejamos como pode ser feito!

Primeiramente, vamos criar um arquivo **popupie.css**, que conterá mudanças específicas para o Internet Explorer. No caso, o arquivo simplesmente definirá a cor de fundo da página para vermelho, para que vejamos se o truque funciona ou não.

popupie.css

```
body {
	background-color: red;
	}
```

Ao carregar a página, seja com o FireFox, seja com o IE, nada acontece, porque não indicamos este CSS no HTML. Mas é importante lembrar que este arquivo CSS só será indicado **se** o navegador sendo executado for o Internet Explorer! Para detectar isso, usaremos a variável **navigator.appName**, que nos indicará o nome da aplicação.

Entretanto, este JavaScript não poderá estar junto com o outro, pois esse código precisa ser processado junto com o HTML da página, e para isso o código precisa estar dentro do cabeçalho, diretamente.

Primeiramente, então criaremos mais um campo <SCRIPT>...</SCRIPT>, com a sintaxe ligeiramente diferente:

popup.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-BR">
<head>
	<meta http-equiv="Content-Type" content="text/html;charset=iso-8859-1" />
	<title>Teste de JavaScript</title>
	<script type="text/javascript" src="popup.js"></script>
	<link href="popup.css" rel="stylesheet" type="text/css" />
	<script type="text/javascript">
	<!--
	-->
	</script>
```

```
</head>
<body>
	<p><input type="button" value="Mostra" id="b1" /></p>
	<div id="d1"><input type="button" value="Esconde" id="b2" /></div>
</body>
</html>
```

Dentro deste campo, verificaremos se o nome da aplicação-navegador é o Microsoft Internet Explorer:

popup.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-BR">
<head>
	<meta http-equiv="Content-Type" content="text/html;charset=iso-8859-1" />
	<title>Teste de JavaScript</title>
	<script type="text/javascript" src="popup.js"></script>
	<link href="popup.css" rel="stylesheet" type="text/css" />
	<script type="text/javascript">
	<!--
	  if (navigator.appName == "Microsoft Internet Explorer") {
	  }
	-->
	</script>
</head>
<body>
	<p><input type="button" value="Mostra" id="b1" /></p>
	<div id="d1"><input type="button" value="Esconde" id="b2" /></div>
</body>
</html>
```

Se a comparação for verdadeira, ou seja, o navegador **é** o Internet Explorer, então o código interno do IF será executado. Assim, precisamos instruir o JavaScript a escrever uma nova linha de inclusão para o **popupie.css**, o que pode ser feito usando a função **document.write(' ... ')**, como indicado a seguir.

popup.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-BR">
<head>
	<meta http-equiv="Content-Type" content="text/html;charset=iso-8859-1" />
	<title>Teste de JavaScript</title>
	<script type="text/javascript" src="popup.js"></script>
	<link href="popup.css" rel="stylesheet" type="text/css" />
	<script type="text/javascript">
	<!--
	  if (navigator.appName == "Microsoft Internet Explorer") {
	  document.write('<link href="popup2.css" rel="stylesheet" type="text/css" />');
	  }
	-->
	</script>
```

```
</head>
<body>
	<p><input type="button" value="Mostra" id="b1" /></p>
	<div id="d1"><input type="button" value="Esconde" id="b2" /></div>
</body>
</html>
```

Tente recarregar a página no Internet Explorer agora e veja como o fundo fica vermelho, logo ao carregar! Carregando em qualquer outro navegador, o fundo permanecerá na cor padrão. É claro que você pode criar CSS alternativos bem mais complexos que este, mas esse já serve de prova de conceito. Você pode também incluir CSS diferentes para cada navegador. Repito, entretanto, que ter vários CSS não é uma boa prática e deve ser evitado, dentro do possível. Se necessário usar estes arquivos alternativos, faça apenas **ajustes** no CSS principal. Recriar o CSS inteiro para cada navegador pode gerar muitos problemas de manutenção no futuro.

Adicionalmente, este princípio pode servir para qualquer modificação na página que se queira em navegadores específicos, incluindo a inserção de código JavaScript diferente para cada navegador: basta modificar a linha que é impressa pelo JavaScript ou, ainda, acrescentar novas linhas a serem impressas!

3. VALIDAÇÃO DE FORMULÁRIOS

Uma validação de formulários client-side bem feita usa alguns dos recursos mais interessantes e importantes do JavaScript com relação à manipulação do navegador e do CSS. Vejamos algumas destas tarefas.

3.1. Preparando um Formulário para o JavaScript

Normalmente, como já visto anteriormente, o uso de JavaScript está associado ao uso de formulários. Na aula passada foram apresentados vários métodos de acesso a elementos do HTML e de formulários usando o JavaScript.

Assim, iniciemos este estudo com um formulário de cadastro, em que devem ser digitados o nome, idade e mensagem de um usuário, para cadastro. O nome pode ter até 255 caracteres (campo de tamanho 30), a idade pode ter até 3 caracteres (campo de tamanho 4) e a mensagem pode ter qualquer tamanho (área de texto). A "ação" do formulário deve carregar o documento "**cadastro.php**".

O formulário terá a seguinte "cara":

cadastro.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-BR">
<head>
	<meta http-equiv="Content-Type" content="text/html;charset=iso-8859-1" />
	<title>Validação de Formulário</title>
</head>
<body>
	<p>Preencha o formulário abaixo:</p>
	<form method="post" action="cadastro.php" id="formCadastro">
		<p>	<label for="nome">Nome:</label>
			<input type="text" size="30" maxlength="255" id="nome" />
		</p>
		<p>	<label for="idade">idade:</label>
			<input type="text" size="4" maxlength="3" id="idade" />
		</p>
		<p>	<label for="mensagem">Mensagem:</label><br />
			<textarea rows="5" cols="30" id="mensagem"></textarea>
		</p>
		<p><input type="submit" value="Ok" id="ok" /></p>
	</form>
</body>
</html>
```

Quando manipulamos dados de formulários, apesar de todos os métodos descritos anteriormente funcionarem, é comum darmos um nome para cada elemento do formulário, usando o parâmetro NAME. Isso facilita e acelera o acesso aos dados. O resultado é apresentado a seguir:

cadastro.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-BR">
<head>
	<meta http-equiv="Content-Type" content="text/html;charset=iso-8859-1" />
	<title>Validação de Formulário</title>
</head>
<body>
	<p>Preencha o formulário abaixo:</p>
	<form method="post" action="cadastro.php" id="formCadastro">
		<p>	<label for="nome">Nome:</label>
			<input type="text" size="30" maxlength="255" id="nome" name="nome" />
		</p>
		<p>	<label for="idade">idade:</label>
			<input type="text" size="4" maxlength="3" id="idade" name="idade" />
		</p>
		<p>	<label for="mensagem">Mensagem:</label><br />
			<textarea rows="5" cols="30" id="mensagem" name="mensagem"></textarea>
		</p>
		<p><input type="submit" value="Ok" id="ok" /></p>
	</form>
</body>
</html>
```

Isso nos permitirá acessar os dados diretamente, depois de pegar a referência para o formulário. Por exemplo: se **formCadastro** apontar para o formulário, para acessar o texto do nome bastará indicar: *formCadastro.nome.value* . Feito isso, é preciso associar o formulário ao JavaScript **cadastro.js**, que iremos usar para validar este cadastro. Isso pode ser feito como indicado abaixo:

cadastro.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-BR">
<head>
	<meta http-equiv="Content-Type" content="text/html;charset=iso-8859-1" />
	<title>Validação de Formulário</title>
	<script type="text/javascript" src="cadastro.js"></script>
</head>
<body>
	<p>Preencha o formulário abaixo:</p>
	<form method="post" action="cadastro.php" id="formCadastro">
		<p>	<label for="nome">Nome:</label>
			<input type="text" size="30" maxlength="255" id="nome" name="nome" />
		</p>
		<p>	<label for="idade">idade:</label>
			<input type="text" size="4" maxlength="3" id="idade" name="idade" />
		</p>
		<p>	<label for="mensagem">Mensagem:</label><br />
			<textarea rows="5" cols="30" id="mensagem" name="mensagem"></textarea>
		</p>
		<p><input type="submit" value="Ok" id="ok" /></p>
	</form>
</body>
</html>
```

3.2. Ligando o Código JavaScript ao Formulário

Tudo pronto no HTML, é hora de trabalhar com o JavaScript. O primeiro passo é criar a função **configura()**, já vista anteriormente, responsável por inicializar os controles da página:

cadastro.js

```
/* Inicialização do Documento */
window.onload = configura;
function configura() {
  }
```

No nosso caso, por simplicidade, faremos apenas uma validação dos dados quando o usuário enviar os dados (clicar no botão "Ok"). Há duas formas de fazer isso: associando o método **validar()** ao evento de clicar no botão ou associar o método **validar()** ao evento "onsubmit" (tradução: "ao enviar") do formulário.

Dado que a segunda alternativa é mais elegante, ela será a usada neste caso. O jeito de fazer isso, como já visto anteriormente, será o seguinte:

cadastro.js

```
/* Inicialização do Documento */
window.onload = configura;
function configura() {
  document.getElementById("formCadastro").onsubmit = validar;
  }
```

Ora, o evento está associado à função "validar()", mas ela ainda não existe! É preciso criá-la! Antes, porém, é preciso conhecer um fato: o navegador espera que uma função associada ao evento "onsubmit" sempre retorne um valor "false" ou "true".

Caso o valor de retorno seja "false", o envio dos dados será abortado. Se o valor do envio for "true", o envio de dados será realizado como previsto. Assim, ao criar a função validar, vamos pressupor, inicialmente, que os dados estão corretos e, assim, a função validar() deve, por padrão, retornar o valor **true**. O resultado pode ser visto a seguir.

cadastro.js

```
/* Inicialização do Documento */
window.onload = configura;
function configura() {
  document.getElementById("formCadastro").onsubmit = validar;
  }

/* Valida Formulário */
function validar() {
  return true;
  }
```

Feito isso, o evento de envio do formulário está associado a uma função que ainda não faz nada. Para verificar se tudo está ok, adicionaremos uma janela de alerta:

cadastro.js

```
/* Inicialização do Documento */
window.onload = configura;
function configura() {
  document.getElementById("formCadastro").onsubmit = validar;
  }

/* Valida Formulário */
function validar() {
  window.alert("Entrou na validar()");
  return true;
  }
```

Ao carregar a página do formulário **cadastro.html** e clicar no botão, deverá aparecer uma janela dizendo "Entrou na validar()", antes da mensagem de erro dizendo que "cadastro.php" não pode ser encontrado. Caso isso não aconteça, revise os passos anteriores.

Com tudo funcionando, estamos prontos para as validações, que serão:
a) Nome precisa estar preenchido
b) Nome precisa ter 6 caracteres ou mais
c) Idade precisa ser um número
d) Idade precisa ser pelo menos 18 anos
e) Idade pode ser, no máximo, 150 anos
f) A mensagem precisa conter pelo menos 1 caractere

Como sempre vamos precisar acessar os elementos do formulário **formCadastro**, vamos criar uma referência para ele:

cadastro.js

```
/* Inicialização do Documento */
window.onload = configura;
function configura() {
  document.getElementById("formCadastro").onsubmit = validar;
  }

/* Valida Formulário */
function validar() {
  var formCadastro = document.getElementById("formCadastro");
  return true;
  }
```

3.2. Validando o Formulário

Vejamos, uma a uma, cada validação que precisa ser feita.

3.2.1. Verificando se o nome do usuário foi preenchido

A primeira validação pode ser feita verificando o comprimento (**length**) do valor (**value**) do campo **nome**, do formulário **formCadastro**. Se este comprimento for igual a zero, significa que nada foi digitado no campo e devemos, assim, emitir uma mensagem de erro como "Você precisa digitar um nome!".

A verificação pode ser feita assim:

```
if (formCadastro.nome.value.length == 0) {
  [... código ...]
  }
```

Lembrando que só podemos usar "formCadastro.nomeDoCampo" porque criamos uma referencia para formCadastro antes!

De qualquer forma, convém indicar o que esta trecho faz, com um comentário:

```
        /* Verifica se nome está preenchido */
        if (formCadastro.nome.value.length == 0) {
          [... código ...]
          }
```

E, se o nome está incompleto, a função deverá retornar com um "false" para cancelar o envio dos dados:

```
        /* Verifica se nome está preenchido */
        if (formCadastro.nome.value.length == 0) {
          [... código ...]
          return false;
          }
```

Colocando este código no arquivo JavaScript, o resultado será:

cadastro.js

```
/* Inicialização do Documento */
window.onload = configura;
function configura() {
  document.getElementById("formCadastro").onsubmit = validar;
  }

/* Valida Formulário */
function validar() {
        var formCadastro = document.getElementById("formCadastro");
        /* Verifica se nome está preenchido */
        if (formCadastro.nome.value.length == 0) {
          return false;
          }
  return true;
  }
```

Neste ponto, ao recarregar a página, se o botão "Ok" for clicado sem nada no campo "nome", o formulário simplesmente vai parecer não funcionar. Digitando algo, o antigo comportamento de "cadastro.php não encontrado" irá ocorrer. Entretanto, esse comportamento não é intuitivo: falta uma mensagem de erro.

Inicialmente usaremos uma janela do tipo alert() para isso:

cadastro.js

```
/* Inicialização do Documento */
window.onload = configura;
function configura() {
  document.getElementById("formCadastro").onsubmit = validar;
  }

/* Valida Formulário */
function validar() {
        var formCadastro = document.getElementById("formCadastro");

        /* Verifica se nome está preenchido */
        if (formCadastro.nome.value.length == 0) {
          window.alert("Você precisa digitar um nome!");
          return false;
          }

  return true;
  }
```

Como essa maneira de reportar um erro é feia e ruim, vamos isolá-la em uma nova função, chamada "informarErro()", que receberá como parâmetro uma mensagem de erro. Futuramente poderemos mudar a maneira de informar o erro apenas alterando esta função **informarErro()**.

cadastro.js

```
/* Inicialização do Documento */
window.onload = configura;
function configura() {
  document.getElementById("formCadastro").onsubmit = validar;
  }

/* Valida Formulário */
function validar() {
        var formCadastro = document.getElementById("formCadastro");

        /* Verifica se nome está preenchido */
        if (formCadastro.nome.value.length == 0) {
          informarErro("Você precisa digitar um nome!");
          return false;
          }
  return true;
  }

/* Função Auxiliar para Informar Erros de Preenchimento */
function informarErro(msg) {
  window.alert(msg);
  }
```

3.2.2. Verificando se o nome do usuário tem pelo menos 6 dígitos

O processo é similar ao anterior, mas agora devemos verificar se o comprimento (length) do nome é menor que 6 para indicar o erro. A mensagem deve ser "Nome muito curto!". Para fazer isso, pode-se usar o seguinte código:

```
        /* Verifica se nome tem, no mínimo, 6 caracteres  */
        if (formCadastro.nome.value.length < 6) {
          informarErro("Nome muito curto!");
          return false;
          }
```

Inserido no código, isso fica:

cadastro.js

```
/* Inicialização do Documento */
window.onload = configura;
function configura() {
  document.getElementById("formCadastro").onsubmit = validar;
  }

/* Valida Formulário */
function validar() {
        var formCadastro = document.getElementById("formCadastro");
        /* Verifica se nome está preenchido */
        if (formCadastro.nome.value.length == 0) {
          informarErro("Você precisa digitar um nome!");
          return false;
          }
        /* Verifica se nome tem, no mínimo, 6 caracteres  */
        if (formCadastro.nome.value.length < 6) {
          informarErro("Nome muito curto!");
          return false;
          }
  return true;
  }

/* Função Auxiliar para Informar Erros de Preenchimento */
function informarErro(msg) {
  window.alert(msg);
  }
```

Note o uso da função auxiliar informarErro().

3.2.3. Verificando se a idade é um número

O processo aqui é similar aos anteriores, mas verificaremos se o valor (**value**) do campo **idade** é um número. Existem muitas formas de fazer isso, mas a mais simples é usando uma função interna do JavaScript que diz se uma variável não é um número: isNaN() (de *isNotANumber*). A mensagem deve ser "A idade precisa ser um número!". Para fazer isso, pode-se usar o seguinte código:

```
        /* Verifica se idade é um número  */
        if ( isNaN( formCadastro.idade.value ) ) {
          informarErro("A idade precisa ser um número!");
          return false;
          }
```

Inserido no código geral teremos...

cadastro.js

```
/* Inicialização do Documento */
window.onload = configura;
function configura() {
  document.getElementById("formCadastro").onsubmit = validar;
  }

/* Valida Formulário */
function validar() {
        var formCadastro = document.getElementById("formCadastro");
        /* Verifica se nome está preenchido */
        if (formCadastro.nome.value.length == 0) {
          informarErro("Você precisa digitar um nome!");
          return false;
          }
        /* Verifica se nome tem, no mínimo, 6 caracteres  */
        if (formCadastro.nome.value.length < 6) {
          informarErro("Nome muito curto!");
          return false;
          }
        /* Verifica se idade é um número  */
        if ( isNaN( formCadastro.idade.value ) ) {
          informarErro("A idade precisa ser um número!");
          return false;
          }
  return true;
  }

/* Função Auxiliar para Informar Erros de Preenchimento */
function informarErro(msg) {
  window.alert(msg);
  }
```

3.2.4. Verificando se a idade é menor que 18 anos

O processo aqui é similar aos anteriores, verificando se o valor do campo idade é menor que 18, informando o erro "A idade mínima é 18 anos!":

```
/* Verifica se idade é menor que 18 anos  */
if (formCadastro.idade.value < 18) {
  informarErro("A idade mínima é 18 anos!");
  return false;
  }
```

3.2.5. Verificando se a idade é maior que 150 anos

O processo aqui é similar aos anteriores, verificando se o valor do campo idade é maior que 150, informando o erro "Ninguém vive tanto tempo!":

```
/* Verifica se idade é maior que 150 anos  */
if (formCadastro.idade.value > 150) {
  informarErro("Ninguém vive tanto tempo!");
  return false;
  }
```

Este código pode ser inserido na função validar conforme visto anteriormente, tomando o cuidado de inseri-lo **após** a verificação de que a idade é um número (a comparação realizada não tem sentido se a idade não for um número!).

3.2.6. Validação da Mensagem

Neste caso, basta testar se o comprimento é menor que 1, e retornar erro caso positivo:

```
/* Verifica se Mensagem está preenchida  */
if ( formCadastro.mesagem.value.length < 1 ) {
  informarErro("A mensagem precisa estar preenchida!");
  return false;
  }
```

Acrescentando esta função ao final do arquivo JavaScript, está completa a nossa validação do lado do cliente. O arquivo final é:

cadastro.js

```
/* Inicialização do Documento */
window.onload = configura;
function configura() {
  document.getElementById("formCadastro").onsubmit = validar;
  }

/* Valida Formulário */
function validar() {
        var formCadastro = document.getElementById("formCadastro");
        /* Verifica se nome está preenchido */
        if (formCadastro.nome.value.length == 0) {
          informarErro("Você precisa digitar um nome!");
          return false;
          }
        /* Verifica se nome tem, no mínimo, 6 caracteres  */
        if (formCadastro.nome.value.length < 6) {
          informarErro("Nome muito curto!");
          return false;
          }
        /* Verifica se idade é um número  */
        if ( isNaN( formCadastro.idade.value ) ) {
          informarErro("A idade precisa ser um número!");
          return false;
          }
        /* Verifica se idade é menor que 18 anos  */
        if (formCadastro.idade.value < 18) {
          informarErro("A idade mínima é 18 anos!");
          return false;
          }
        /* Verifica se idade é maior que 150 anos  */
        if (formCadastro.idade.value > 150) {
          informarErro("Ninguém vive tanto tempo!");
          return false;
          }
        /* Verifica se Mensagem está preenchida  */
        if ( formCadastro.mesagem.value.length < 1 ) {
          informarErro("A mensagem precisa estar preenchida!");
          return false;
          }
  return true;
  }

/* Função Auxiliar para Informar Erros de Preenchimento */
function informarErro(msg) {
  window.alert(msg);
  }
```

4. MELHORANDO O VISUAL DO ERRO

Nos exemplos acima, usamos a janela do tipo "alert()" para indicar os erros. Isso, além de feio, prejudica a usabilidade da página, já que a janela atrapalha o uso do navegador enquanto não for fechada.

Uma solução elegante para isso é indicar a mensagem de erro na própria página HTML. Para isso, antes de mais nada, acrescentaremos o espaço de um parágrafo onde serão indicadas as mensagens de erro:

cadastro.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-BR">
<head>
	<meta http-equiv="Content-Type" content="text/html;charset=iso-8859-1" />
	<title>Validação de Formulário</title>
	<script type="text/javascript" src="cadastro.js"></script>
</head>
<body>
	<p>Preencha o formulário abaixo:</p>
	<p id="erros"></p>
	<form method="post" action="cadastro.php" id="formCadastro">
		<p>	<label for="nome">Nome:</label>
			<input type="text" size="30" maxlength="255" id="nome" name="nome" />
		</p>
		<p>	<label for="idade">idade:</label>
			<input type="text" size="4" maxlength="3" id="idade" name="idade" />
		</p>
		<p>	<label for="mensagem">Mensagem:</label><br />
			<textarea rows="5" cols="30" id="mensagem" name="mensagem"></textarea>
		</p>
		<p><input type="submit" value="Ok" id="ok" /></p>
	</form>
</body>
</html>
```

Com um parágrafo nomeado de "erros", podemos inserir o texto do erro lá. Considere a função auxiliar de erro anterior:

```
/* Função Auxiliar para Informar Erros de Preenchimento */
function informarErro(msg) {
  window.alert(msg);
  }
```

Se, ao invés de chamar o método "window.alert(msg)", mudarmos o conteúdo do texto do parágrafo de ID "erros", nossos erros serão apresentados na tela. Isso pode ser feito da seguinte forma:

```
/* Função Auxiliar para Informar Erros de Preenchimento */
function informarErro(msg) {
  document.getElementById("erros").innerHTML = msg;
  }
```

Embora isso já dê conta do recado, não chama muito a atenção do usuário. Então, vamos modificar também a cor deste texto:

```
/* Função Auxiliar para Informar Erros de Preenchimento */
function informarErro(msg) {
  document.getElementById("erros").style.color = "red";
  document.getElementById("erros").innerHTML = msg;
  }
```

Substituindo a função informarErro anterior por esta acima, o resultado será muito mais amigável! Experimente! Mas... o usuário ainda pode ficar perdido em um formulário com muitos campos. A solução para isso é acrecentar uma função que **ilumine** a borda do campo na cor vermelha, além de transferir o foco para o tal elemento:

```
function setCorCampo(campo) {
  campo.style.borderColor = "red";
  campo.focus();
  }
```

No código isso fica:

cadastro.js

```
/* Inicialização do Documento */
window.onload = configura;
function configura() {
  document.getElementById("formCadastro").onsubmit = validar;
  }

/* Valida Formulário */
function validar() {
        var formCadastro = document.getElementById("formCadastro");
        /* Verifica se nome está preenchido */
        if (formCadastro.nome.value.length == 0) {
          informarErro("Você precisa digitar um nome!");
          setCorCampo(formCadastro.nome);
          return false;
          }
```

```
        /* Verifica se nome tem, no mínimo, 6 caracteres  */
        if (formCadastro.nome.value.length < 6) {
          informarErro("Nome muito curto!");
          setCorCampo(formCadastro.nome);
          return false;
          }
        /* Verifica se idade é um número  */
        if ( isNaN( formCadastro.idade.value ) ) {
          informarErro("A idade precisa ser um número!");
          setCorCampo(formCadastro.idade);
          return false;
          }
        /* Verifica se idade é menor que 18 anos  */
        if (formCadastro.idade.value < 18) {
          informarErro("A idade mínima é 18 anos!");
          setCorCampo(formCadastro.idade);
          return false;
          }
        /* Verifica se idade é maior que 150 anos  */
        if (formCadastro.idade.value > 150) {
          informarErro("Ninguém vive tanto tempo!");
          setCorCampo(formCadastro.idade);
          return false;
          }
        /* Verifica se Mensagem está preenchida  */
        if ( formCadastro.mesagem.value.length < 1 ) {
          informarErro("A mensagem precisa estar preenchida!");
          setCorCampo(formCadastro.mensagem);
          return false;
          }
  return true;
  }

/* Função Auxiliar para Informar Erros de Preenchimento */
function informarErro(msg) {
  document.getElementById("erros").style.color = "red";
  document.getElementById("erros").innerHTML = msg;
  }

/* Muda cor do Campo com Erro */
function setCorCampo(campo) {
  campo.style.borderColor = "red";
  campo.focus();
  }
```

Para finalizar e completar, precisamos "limpar" a cor das bordas dos campos no início da validação, para evitar que os campos do formulário fiquem "pintados" para sempre, entre duas validações. Isso pode ser feito no início do código do valida(), conforme indicado a seguir.

cadastro.js

```
/* Inicialização do Documento */
window.onload = configura;
function configura() {
  document.getElementById("formCadastro").onsubmit = validar;
  }

/* Valida Formulário */
function validar() {
        var formCadastro = document.getElementById("formCadastro");
        formCadastro.nome.style.borderColor = "";
        formCadastro.idade.style.borderColor = "";
        formCadastro.mensagem.style.borderColor = "";

        /* Verifica se nome está preenchido */
        if (formCadastro.nome.value.length == 0) {
          informarErro("Você precisa digitar um nome!");
          setCorCampo(formCadastro.nome);
          return false;
          }

        /* Verifica se nome tem, no mínimo, 6 caracteres  */
        if (formCadastro.nome.value.length < 6) {
          informarErro("Nome muito curto!");
          setCorCampo(formCadastro.nome);
          return false;
          }
        /* Verifica se idade é um número  */
        if ( isNaN( formCadastro.idade.value ) ) {
          informarErro("A idade precisa ser um número!");
          setCorCampo(formCadastro.idade);
          return false;
          }
        /* Verifica se idade é menor que 18 anos  */
        if (formCadastro.idade.value < 18) {
          informarErro("A idade mínima é 18 anos!");
          setCorCampo(formCadastro.idade);
          return false;
          }
        /* Verifica se idade é maior que 150 anos  */
        if (formCadastro.idade.value > 150) {
          informarErro("Ninguém vive tanto tempo!");
          setCorCampo(formCadastro.idade);
          return false;
          }
        /* Verifica se Mensagem está preenchida  */
        if ( formCadastro.mesagem.value.length < 1 ) {
          informarErro("A mensagem precisa estar preenchida!");
          setCorCampo(formCadastro.mensagem);
          return false;
          }
  return true;
  }
```

```
/* Função Auxiliar para Informar Erros de Preenchimento */
function informarErro(msg) {
  document.getElementById("erros").style.color = "red";
  document.getElementById("erros").innerHTML = msg;
  }

/* Muda cor do Campo com Erro */
function setCorCampo(campo) {
  campo.style.borderColor = "red";
  campo.focus();
  }
```

Finalmente, nosso formuário está pronto e validado!

5. BIBLIOGRAFIA

W3 schools - CSS Tutorial. Disponível em < http://www.w3schools.com/ >. Visitado em 10 de Março de 2009.

MCLAUGHLIN, B. Use a Cabeça! Ajax. Alta Books, 2008.

MOZILLA Developer Connection. Disponível em < http://developer.mozilla.org/pt >. Visitado em 30 de março de 2009.