Unidade 6: Linguagens Interpretadas: Javascript

Prof. Daniel Caetano

Objetivo: Apresentar algumas tecnologias de desenvolvimento de Web Dinâmica e realizar uma introdução ao JavaScript, sua integração com o navegador com o uso de suas funções mais básicas.

Bibliografia: W3,2009; MCLAUGHLIN,2008; MUTO,2006; RATSCHILLER,2000.

INTRODUÇÃO

Conceitos Chave:
- Linguagens de Programação
  * Parâmetros de trabalho?

Apesar de muito já ter sido apresentado, ao longo do curso, sobre a construção de páginas web, as páginas construídas até o momento são puramente estáticas, isto é, sem elementos que possam ser alterados automaticamente.

Efeitos dinâmicos, por exemplo, podem ser considerados em formulários, para facilitar a validação deste formulário antes de enviá-lo. Para realizar esta tarefa, será usada a linguagem JavaScript, que é uma linguagem interpretada.

Esta unidade apresenta a lógica das linguagens interpretadas e faz uma breve introdução ao uso de JavaScript em conjunto com o navegador.

1. PÁGINAS WEB DINÂMICAS

Já vimos anteriormente o que é a Web Dinâmica, mas o que são "Páginas Web Dinâmicas"? Bem, "Páginas Web Dinâmicas" são páginas que possuem a capacidade de se modificar, de alguma forma, quando o usuário faz alguma ação específica.

Para que isso possa ocorrer, estas páginas possuem, via de regra, um pequeno programa associado a elas (ou incorporado ao seu código XHTML), de maneira que ao ser detectada uma ação relevante do usuário, o programa responderá com a modificação solicitada. Ou seja: devem existir **pequenos programas** associados a **ações** do usuário.

Existem diversas linguagens que permitem este tipo de aplicação. Algumas delas podem ser vistas no quadro abaixo:

Nome	Empresa	Tipo	Similar à	Execução
JavaScript	Mozilla	Interpretada	Java/C++	Cliente
PHP	PHP	Interpretada	C++/Java	Servidor
ASP	Microsoft	Interpretada	VBasic	Servidor
JSP	Oracle	"Compilada"	Java	Servidor
Java Servlets	Oracle	"Compilada"	Java	Servidor
ASP .Net	Microsoft	"Compilada"	VBasic .Net	Servidor

Como pode ser visto na tabela, existem aquelas que rodam do lado do cliente e aquelas que rodam apenas no lado do servidor. Em especial, do lado do cliente, apenas a linguagem JavaScript é considerada um padrão (e por isso é a única apresentada nesta tabela). Como também pode ser observado, ela é uma linguagem *interpretada*. Mas o que significa isso? O que significa a linguagem ser executada "do lado do cliente" (*client-side*) e ser "interpretada"?

2. LINGUAGENS INTERPRETADAS

No mundo Web, as linguagens mais comuns são aquelas conhecidas como "interpretadas". Alguns exemplos deste tipo de linguagem são JavaScript, PHP e ASP.

Antes de explicar o que é uma linguagem interpretada, é preciso entender um conceito: praticamente nenhum programa de computador é criado em linguagem de máquina, isto é, na linguagem que o computador entende. Os programas são criados em "linguagens de programação", que são convertidas depois para um formato que o computador entenda.

Linguagens Compiladas

Uma linguagem de programação que precise de uma tradução completa antes que o programa seja executado é chamada de uma linguagem "compilada", isto é: existe um programa tradutor que irá ler o texto em uma linguagem de programação como C ou Pascal, e irá gerar um texto em linguagem de máquina, conhecido como "arquivo executável".

O processo é representado a seguir:

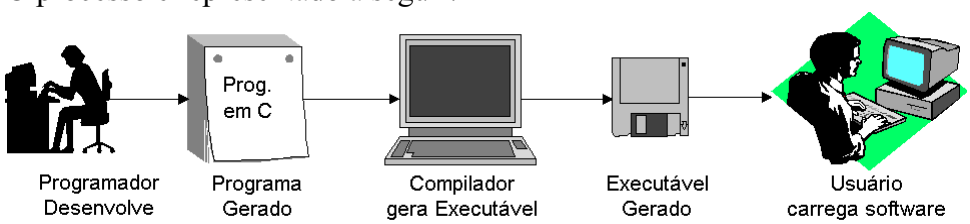

Figura 1: Processo de Criação à Execução com uso de Linguagens Compiladas

O software é, então, distribuído já em seu formato "traduzido para o computador", ou seja, no formato de arquivo executável.

Este processo é análogo ao da criação e tradução de um livro: o escritor russo, por exemplo, escreve um livro completo em sua língua; posteriormente, um tradutor o traduz, por exemplo, para a língua portuguesa e, então, os leitores da língua portuguesa podem ler o livro.

O livro, neste caso, está sendo distribuído já na língua portuguesa, traduzido no formato que o leitor da língua portuguesa compreenda.

Linguagens Interpretadas

Imagine que, ao invés de um livro, estejamos acompanhando um congresso internacional... ou mesmo a premiação do Oscar, que é totalmente narrada em inglês. Queremos assistir ao vivo e, portanto, não é possível esperar uma versão filmada com legendas posteriormente. É necessário existir uma *tradução simultânea*.

A tradução simultânea é, normalmente, feita por um "intérprete" e é daí que vem o nome das linguagens "interpretadas". Em outras palavras, a tradução vai sendo executada à medida em que é necessária.

No caso computacional, a analogia é direta: o programador desenvolve o software em uma linguagem de programação como PHP, JavaScript ou BASIC e distribui este código para as pessoas. As pessoas, por sua vez, usarão este código em seu computador. O processo pode ser representado como indicado na figura 2:

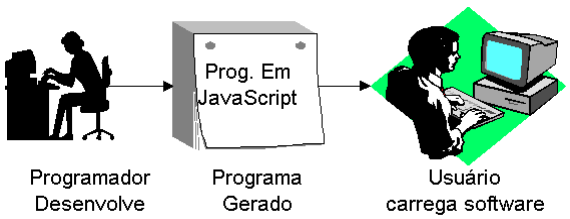

Figura 1: Processo de Criação à Execução com uso de Linguagens Compiladas

A diferença aqui é que, para que o computador do usuário carregue o software e possa executá-lo, será necessário que o computador do usuário possua um *interpretador* da linguagem para o seu computador.

Seria como ir ao Oscar levando um intérprete com você, para que ele pudesse lhe explicar tudo que você está vendo no evento.

2.1. Comparativo

Para o programador, uma das diferenças é óbvia: ele não precisa "compilar" o código. Mas será que esta é a única diferença? E, afinal, qual a diferença prática para o usuário? As diferenças são muitas e marcantes; separaremos em algumas categorias:

Para o Desenvolvedor:

- Praticidade de Desenvolvimento: desenvolver em linguagens interpretadas é, em geral, muito mais prático e rápido. Caso exista algum erro de programação, em geral o interpretador é capaz de fornecer muito mais informações úteis do que o computador executando um programa compilado. Enquanto o interpretador pode avisar que há algum erro em uma determinada linha, o computador executando um programa compilado pode, muitas vezes, simplesmente "congelar".
- Velocidade de Desenvolvimento: Além das características já citadas, o fato de não ter de compilar o programa para realizar cada teste - uma atividade que pode levar de alguns segundos até várias horas - aumenta muito a produtividade do programador.
- Compatibilidade: desenvolver em linguagens interpretadas, em geral, garante um alto nível de compatibilidade com diversos computadores e sistemas operacionais, já que o interpretador possui características constantes, independente do sistema operacional e hardware em que são executados. Por outro lado, para que um usuário possa tirar proveito do software, exige que exista um interpretador para sua máquina/sistema operacional... e que ela esteja instalada.

Em geral, é mais fácil portar um interpretador de uma linguagem para um dado sistema operacional/hardware do que portar todas as aplicações existentes no universo para aquele mesmo sistema operacional/hardware

A "estabilidade" de configurações do ambiente interpretado é a fundação do que se chama de "Virtualização de Servidores", uma prática bastante comum nos tempos atuais, cujo objetivo é exatamente permitir a troca de todo o equipamento sem a necessidade de reinstalar todo o software.

Para o Usuário:

- Desempenho da Aplicação: em geral, aplicações compiladas possuem um desempenho bastante superior ao desempenho das aplicações interpretadas. Por mais que o interpretador seja otimizado, sempre será um intermediário - um passo a mais - no processo.
- Praticidade: em geral, aplicações compiladas são mais práticas, pois basta executá-las. Em ambientes especiais (como navegadores), entretanto, essa vantagem desaparece.
- Comodidade: o mesmo programa pode ser usado em diferentes sistemas operacionais, com os mesmos arquivos de dados, preservando o investimento em aprender aquele aplicativo. Para a mesma comodidade com aplicativos compilados, o usuário fica na dependência do desenvolvedor.

3. CLIENT SIDE x SERVER SIDE

Uma vez compreendido o conceito de "linguagem interpretada", é preciso entender o conceito de linguagens que executam do lado do servidor e linguagens que executam do lado do cliente.

Linguagens Server-Side

A idéia de linguagem que executa do lado do servidor é simples: imagine que não há páginas HTML no servidor, há somente um programa capaz de gerar todas as páginas necessárias. Qualquer página que seja solicitada pelo navegador ao servidor, será gerada pelo programa apenas no momento de enviá-la e, após o envio, ela será destruída.

O processo pode ser representado conforme a figura 3.

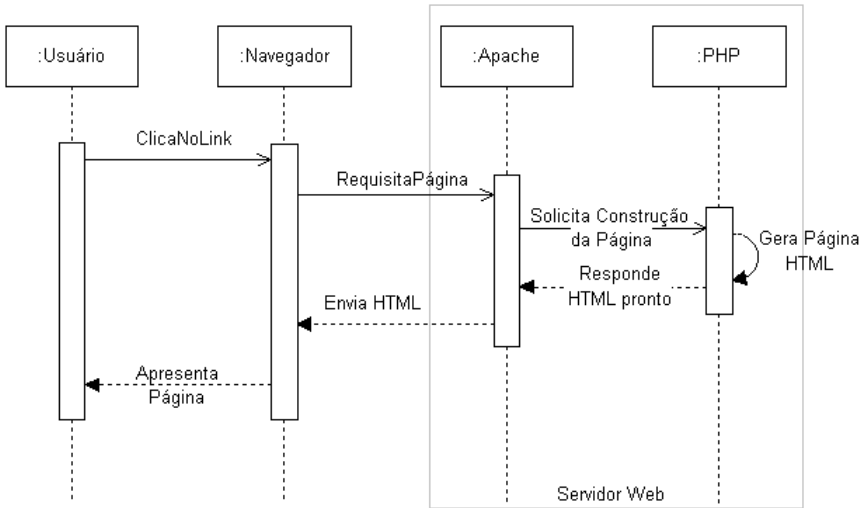

Figura 3: Linguagens processadas "no lado do servidor"

Note que qualquer modificação na página que seja executada por uma linguagem "Server Side" exige que a página HTML seja solicitada, gerada e retransmitida para o navegador. Assim, uma mudança simples no menu exigiria o recarregamento total da página.

Por outro lado, como o código da linguagem "server side" não chega ao navegador, ele é totalmente independente do navegador sendo utilizado. Além disso, o código que é executado "server side" garante mais segurança, já que o usuário do navegador não tem acesso a ele.

Por estas razões, o uso de uma linguagem server-side é interessante para mudanças grandes na página ou para funções que exigem um maior nível de segurança. As linguagens server side também são usadas quando o desenvolvedor quer garantir que um recurso funcione em absolutamente qualquer navegador, independentemente de seus recursos.

Linguagens Client-Side

As linguagens client-side, por outro lado, tem uma característica diversa. Imagine que, juntamente com a página, possamos enviar algum código, que possa ser executado - pelo Navegador - quando o usuário aciona algum botão ou move o mouse até uma dada região.

O processo pode ser representado conforme a figura 4.

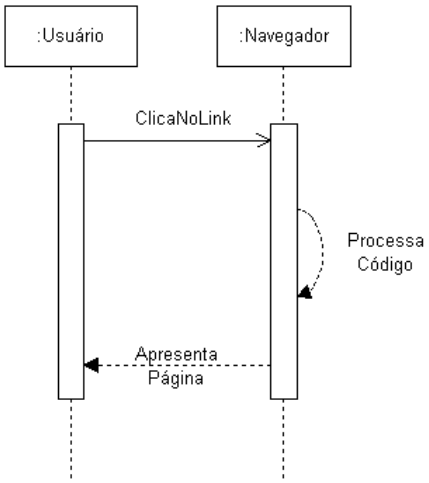

Figura 4: Linguagens processadas "no lado do cliente"

Note que as modificações que eventualmente sejam executadas por uma linguagem "Client Side" não exigem uma comunicação com o servidor, evitando que seja necessário um carregamento completo da página. Na verdade, nem mesmo um recarregamento é necessário.

> Em geral, as linguagens Client-Side como o JavaScript não realizam solicitações ao servidor, mas isso não quer dizer que isso seja impossível. Quando se constrói uma página AJAX, por exemplo, o JavaScript terá um papel ativo na solicitação de dados ao servidor, mas justamente com o objetivo de evitar o recarregamento completo da página.

Entretanto, para que o navegador possa executar um código, ele precisa ser enviado ao navegador, o que diminui a segurança - já que o usuário terá acesso ao funcionamento do código. Além disso, como ele será executado pelo navegador, ele passa a depender da capacidade de execução do navegador específico que o usuário está usando no momento.

Por estas razões, o uso de uma linguagem client-side é interessante para mudanças pequenas e que não sejam fundamentais para a segurança do site. As linguagens client side também são usadas quando o desenvolvedor deseja modificar características específicas do navegador, como sumir com barras de endereços etc.

4. A LINGUAGEM JAVASCRIPT

Conforme já apresentado, a linguagem JavaScript é, então, uma linguagem interpretada client-side. Assim, o código JavaScript é enviado pelo servidor Web para o navegador, que irá agir como interpretador desta linguagem, "executando-a" quando necessário.

Por ser executada no navegador, esta linguagem permite um alto grau de interação com o mesmo, permitindo a alteração de elementos do navegador e da página com muita facilidade. Por outro lado, deve-se evitar o uso exclusivo de JavaScript para controle de segurança de um WebSite (login, por exemplo).

Uma função útil do JavaScript é, por exemplo, modificar a cor de um texto, modificar uma figura, alterar o texto de um determinado elemento da página e assim por diante. Um uso muito comum é "interceptar" o envio de um formulário, para verificar se os dados estão corretamente preenchidos antes que eles sejam efetivamente transmitidos para o servidor.

4.1. Funcionamento Básico do JavaScript

O JavaScript é, essencialmente, uma linguagem *orientada a eventos*. Isso significa que devemos associar trechos de código aos eventos de uma página. Por exemplo, se quisermos mudar a cor de fundo de uma página quando o usuário clicar em um botão específico, devemos fazer duas coisas:

a) Criar a página com o botão
b) Criar um *pedaço de código* que mude a cor de fundo da página;
c) associaremos este *pedaço de código* ao evento *clicar* do botão.

a) Criando a página com um botão:

Isso pode ser feito com uma página simples, com um botão dentro:

teste.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>
</head>
<body>
	<p>Teste</p>
	<p><input type="button" value="Cor" /></p>
</body>
</html>
```

É claro que, ao clicar no botão que será apresentado, nada ocorrerá, porque não há função associada a ele. Isso será feito posteriormente.

b) Criando o código que mude a cor do fundo:

Antes de mais nada, precisamos criar um novo arquivo. Assim como o HTML fica em um arquivo *arquivo.html* e o CSS fica em um arquivo *arquivo.css*, também o JavaScript terá seu próprio arquivo *arquivo.js*. Criemos, com o notepad, então, o arquivo chamado ***efeitos.js***.

Neste arquivo, será indicado um trecho de código responsável por mudar a cor de fundo da página. O nome dado a um "*trecho de código*" que faz uma tarefa específica é **função**. Assim, precisaremos criar uma função para mudar a cor de fundo da página. Um bom nome para esta função seria **mudaCorDeFundo()**.

> Nota: Os parênteses ao final do nome da função são importantes. Seu uso será visto posteriormente.

Comecemos com uma função vazia:

efeitos.js

```
function mudaCorDeFundo() {
}
```

No caso, queremos modificar o "corpo" do documento, que é indicado da seguinte forma:

document.body

A cor de fundo é controlada pelo atributo ***style.backgroundColor*** deste elemento "body". Assim, podemos mudar a cor de fundo para #000000 fazendo algo como:

efeitos.js

```
function mudaCorDeFundo() {
        document.body.style.backgroundColor = "#000000";
        }
```

> Note que o estilo "background-color" tornou-se "backgroundColor". Isto ocorrerá sempre: um "-" ser eliminado e a letra seguinte a ele ser transformada em maiúscula. Isso ocorre porque no JavaScript não é permitido o nome de um elemento contendo o carctere "-".

Agora precisamos indicar este script no XHTML, para que ele possa ser usado. Isso pode ser feito conforme indicado a seguir, com a tag SCRIPT:

teste.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="efeitos.js"></script>
</head>
<body>
	<p>Teste</p>
	<p><input type="button" value="Cor" /></p>
</body>
</html>
```

Observe que a tag SCRIPT não funcionará bem com o "auto fechamento", isto é, usando o <script ... /> ao invés de <script ...> </script>. Caso não se deseje usar um arquivo externo de script (para um script que só tem sentido naquela página, por exemplo), pode-se acrescentar o script na região delimitada pelas tags <script>...</script>.

Feito isso, ao carregar a página e clicar no botão... nada ocorre! Por quê? Porque a função "mudaCorDeFundo()" ainda não foi associada ao botão! Vejamos como fazer isso!

c) Associando a função ao evento de clique do botão.

O último passo do processo será, então, associar a função mudaCorDeFundo() ao botão definido anteriormente. Para fazer isso, entretanto, precisaremos adicionar um ID ao botão, de maneira que possamos identificá-lo facilmente no JavaScript:

teste.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="efeitos.js"></script>
</head>
<body>
	<p>Teste</p>
	<p><input type="button" value="Cor" id="bmudacor" /></p>
</body>
</html>
```

Precisamos, agora, colocar alguns comando no arquivo de JavaScript que seja executado assim que o JavaScript é carregado. Isso é simples: basta colocar estes comandos no início do arquivo .js, fora de qualquer função.

No caso, precisamos associar o evento "onclick" do botão com a função MudaCorDeFundo(). Supondo que tivéssemos acesso direto ao botão, poderíamos fazer isso com uma linha do tipo:

botao.onclick = mudaCorDeFundo;

> IMPORTANTE: o nome da função, neste tipo de atribuição, NÃO deve ter os parênteses **()** no final!

Mas, infelizmente, nós não temos acesso direto ao botão. Por outro lado, podemos pedir que o JavaScript encontre um elemento através do ID deste elemento, e associe este elemento a uma variável. Isso pode ser feito da seguinte forma:

```
var botao = document.getElementById("bmudacor");
```

Depois disso, a variável *botão* irá acessar diretamente o botão desejado! Assim, basta inserir os dois comandos já indicados, na ordem apropriada, no arquivo .js:

efeitos.js

```
var botao = document.getElementById("bmudacor");
botao.onclick = MudaCorDeFundo;

function mudaCorDeFundo() {
        document.body.style.backgroundColor = "#000000";
        }
```

Mas isso ainda não funciona sempre. O problema é o seguinte: quando a página HTML é lida, o arquivo **.js** será lido ao mesmo tempo, e é possível que o código tente associar o evento ao botão "bmudacor" antes que ele tenha sido criado no navegador! Isso certamente causará um problema. A solução para isso é criar uma função de configuração (normalmente chamada de ***configura*** ou ***init***) e associá-la a um evento que ocorra *apenas* quando o conteúdo da página tiver sido carregado inteiramente.

Este evento, chamado ***onload***, que é disparado quando uma página tem seu carregamento finalizado, **não** é um evento do documento, mas da janela do navegador. Assim, o acesso a ele é feito pelo indicador ***window.onload***. A solução, que funciona garantidamente e é mais elegante do que a anteriormente apresentada, é indicada a seguir:

efeitos.js

```
window.onload = configura;

function configura() {
        var botao = document.getElementById("bmudacor");
        botao.onclick = MudaCorDeFundo;
        }
```

```
function mudaCorDeFundo() {
        document.body.style.backgroundColor = "#000000";
        }
```

Isso deve resolver o nosso problema. Carregando a página **teste.html**, um botão será apresentado. Ao clicar neste botão, a tela ficará preta.

4.2. Mudando um Texto em JavaScript

Uma das coisas mais comuns a se fazer em um JavaScript é modificar o texto de um trecho de uma página, para fazer um help-online, por exemplo. Pegando o exemplo anterior, o primeiro passo é definir um parágrafo com um identificador, por exemplo "ajuda", para que possamos alterá-lo:

```
<!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="efeitos.js"></script>
</head>
<body>
        <p id="ajuda">Aqui aparece o help!</p>
        <p><input type="button" value="Cor" id="bmudacor" /></p>
</body>
</html>
```

Agora basta acrescentar uma indicação no JavaScript para mudar este texto. Isso pode ser feito com maior facilidade usando o método "innerHTML", do parágrafo:

```
document.getElementById("ajuda").innerHTML = "Texto do Help";
```

5. EVENTOS COMUNS

A maioria doa elementos do HTML causam eventos, aos quais podemos associar funções de JavaScript. Os eventos mais comuns são listados a seguir.

Até o HTML 4.01 existiam duas formas de acessar os elementos. Uma destas formas era indicar seu caminho completo usando a seguinte sintaxe:

```
document.continente.elemento.evento = funcao
```

Por exemplo, para associar a função "corrigeTexto()" ao evento "onchange" de um elemento de formulário INPUT do tipo TEXT que tenha nome "dado", usa-se o seguinte:

.html

```
<form name="form1">
        <input type="text" name="dado">
</form>
```

.js

```
document.form1.dado.onchange = corrigeTexto;
```

MAS ATENÇÃO: ESSA FORMA NÃO É MAIS SUPORTADA NO PADRÃO. Como resolver o problema?

A forma padronizada de acessar os elementos, para evitar qualquer tipo de problema, é pedindo para o documento (XHTML) encontrar um elemento qualquer, usando seu ID como chave de busca (parâmetro ID na tag XHTML). A sintaxe é:

```
document.getElementById("identificação").evento = função
```

Repetindo o exemplo, para associar a função "corrigeTexto()" ao evento "onchange" de um elemento de formulário INPUT do tipo TEXT que tenha ID "dado", usa-se o seguinte:

.html

```
<input type="text" id="dado">
```

.js

```
document.getElementById("dado").onchange = corrigeTexto;
```

A lista de eventos mais comuns está apresentada a seguir, e uma versão completa dela está na referência de JavaScript.

Atenção: TODOS os nomes devem ser digitados EXATAMENTE como indicado, incluindo maiúsculas e minúsculas.

DOCUMENT, WINDOW, BODY e FRAMESET

onload	Quando um documento **inicia** seu carregamento

Elementos de FORM

onchange	Quando o conteúdo de um elemento for alterado
onfocus	Quando o elemento receber foco
onselect	Quando um elemento for selecionado
onsubmit	Quando o formulário for enviado

Eventos de Teclado (válido para quase todos os elementos)

onkeydown	Quando uma tecla for pressionada (com foco no elemento)
onkeypress	Quando uma tecla for pressionada e solta (com foco no elem.)
onkeyup	Quando uma tecla for solta (com foco no elemento)

Eventos de Mouse (válido para quase todos os elementos)

onclick	Quando o elemento for clicado
ondbclick	Quando o elemento for duplamente clicado
onmousemove	Quando o mouse se mover sobre o elemento
onmouseout	Quando o mouse sair de cima do elemento
onmouseover	Quando o mouse passar sobre o elemento
onmouseup	Quando o botão do mouse for solto sobre o elemento

6. PROPRIEDADES VISUAIS QUE PODEM SER ALTERADAS

As propriedades visuais dos elementos podem ser acessadas de maneira similar aos eventos, usando os dois mecanismos já apresentados. A sintaxe segue abaixo:

```
document.getElementById("identificação").style.estilo = valor
```

Exemplo:

.html

```
<input type="text" id="dado">
```

.js

```
document.getElementById("dado").style.backgroundColor = "black";
```

A lista de estilos mais comuns está apresentada a seguir, e a lista mais completa está na referência de JavaScript.

Atenção: TODOS os nomes devem ser digitados EXATAMENTE como indicado, incluindo maiúsculas e minúsculas.

Plano de Fundo

backgroundColor	Muda cor de fundo de um elemento.
backgroundImage	Muda a imagem de fundo de um elemento

Textos

color	Muda a cor do texto
fontSize	Muda o tamanho da fonte
textAlign	Muda o alinhamento do texto
textDecoration	Muda a "decoração" de um texto

Bordas e Margens

borderColor	Muda a cor das bordas todas
borderStyle	Muda estilo de todas as bordas
borderWidth	Muda largura de todas as bordas
margin	Muda todas as margens
outlineColor	Muda a cor da linha de contorno
outlineStyle	Muda o estilo da linha de contorno
outlineWidth	Muda a largura da linha de contorno
padding	Muda espaçamento interno de um elemento

Layout

cursor	Muda o cursor a ser apresentado
display	Muda a maneira que o elemento será apresentado
overflow	O que fazer com conteúdo que não cabem no elemento.
visibility	Muda a visibilidade de um elemento
width	Muda a largura de um elemento

Listas

listStyleImage	Muda a imagem de marcador de lista
listStyleType	Muda o tipo de marcador de lista

Posicionamento

zIndex	Define a ordem vertical de um elemento

Barra de Rolagem (Só no IE)

scrollbar3dLightColor	Muda a cor da parte brilhante da barra de rolagem
scrollbarArrowColor	Muda a cor da seta da barra de rolagem
scrollbarBaseColor	Muda a cor base da barra de rolagem
scrollbarDarkShadowColor	Muda a cor da parte sombreada da barra de rolagem
scrollbarFaceColor	Muda a cor de frente da barra de rolagem
scrollbarHighlightColor	Muda a parte brilhante da barra de rolagem
scrollbarShadowColor	Muda a parte sombreada da barra de rolagem
scrollbarTrackColor	Muda a cor de fundo da barra de rolagem

Propriedades Genéricas

title	Muda ou retorna o título de um elemento.

7. ELEMENTOS DE JANELA COMUMENTE USADOS

Os elementos da janela podem ser acessados iniciando-se com o indicador "window". Por exemplo: para desligar a barra de status de uma janela, usa-se:

```
window.statusbar = false;
```

Os elementos normalmente acessados são apresentados abaixo, e uma lista mais completa está na referência de JavaScript.

Atenção: TODOS os nomes devem ser digitados EXATAMENTE como indicado, incluindo maiúsculas e minúsculas.

window.location	Endereço da janela (veja na seção 8)
window.name	Nome da janela
window.parent	Janela "pai"
window.personalbar	Barra personalizada
window.scrollbars	Muda a visibilidade das barras de rolagem
window.status	Referência para a barra de status
window.statusbar	Muda a visibilidade da barra de status
window.toolbar	Muda a visibilidade da barra de ferramentas

A janela também fornece alguns métodos (apenas os mais comuns são citados):

window.alert()	Mostra uma janela de alerta com o texto indicado
window. blur()	Tira o foco da janela atual
window. close()	Fecha a janela
window. confirm()	Apresenta uma janela do tipo "OK/Cancel"
window. createPopup()	Abre uma janela popup
window. moveBy()	Move a janela relativamente à sua posição
window. moveTo()	Move a janela de maneira absoluta
window. open()	Abre uma nova janela do navegador
window. print()	Imprime o conteúdo da janela
window. resizeBy()	Muda o tamanho da janela de maneira relativa
window. resizeTo()	Muda o tamanho da janela de maneira absoluta

A janela possui, ainda, alguns eventos, sendo os mais usados apresentados abaixo:

window.onload	Função a ser executada quando a página estiver completamente carregada.

8. ELEMENTOS DE LOCAÇÃO E TELA

Os elementos de locação (window.location. ...) servem para manipular a localização atual do navegador. Os elementos de tela (screen. ...) servem para **ler** os dados da tela do usuário. Os atributos mais comuns estão listados a seguir, sendo uma lista completa apresentada nas referências.

> **Atenção**: TODOS os nomes devem ser digitados EXATAMENTE como indicado, incluindo maiúsculas e minúsculas.

window.location	URL da página atual carregada
screen.availHeight	Altura da tela (menos a barra de tarefas)
screen.height	Altura da tela
screen.width	Largura da tela

Alguns métodos também estão disponíveis (apenas os mais comuns são citados):

window.location.assign()	Carrega um novo documento
window.location.reload()	Recarrega o documento atual
window.location.replace()	Substitui o documento atual por um novo

9. ATIVIDADE

1. Crie uma página com um botão que mude a cor de fundo da tela para azul com texto em amarelo.
2. Acrescente um parágrafo para um texto de ajuda, que indique "Clique aqui para mudar a cor", quando o mouse passar por cima do botão e volte ao texto normal quando o mouse sair do botão.
3. Modifique o código para que ao clicar novamente no botão o fundo volte a ser branco com texto em preto.
4. Modifique a função de inicialização de maneira que a janela fique com um tamanho 400x300 e esteja centralizada na tela (se a configuração do navegador permitir).

10. 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.

MUTO, C.A. PHP & MySQL: Guia Introdutório. Rio de Janeiro: Brasport, 2006.

RATSCHILLER, T; GERKEN, T; Desenvolvendo aplicações Wev com PHP 4.0. Ed. Ciência Moderna, 2000.