Aula 1: Introdução à Disciplina

Fonte: Plano de Aula Oficial da Disciplina

Objetivo: Identificar os princípios que se destacam como características da Web 2.0.

INTRODUÇÃO

Se você perguntar a uma dúzia de especialistas em internet o que significa o termo Web 2.0 você ouvirá uma dúzia de respostas diferentes. Alguns dizem que a web 2.0 é um conjunto de práticas e filosofias que proporcionam uma experiência rica e profunda aos usuários de internet. Outros dizem que é um conjunto de tecnologias para tornar mais fácil a comunicação e a busca de informações on-line. Alguns jornalistas afirmam que o termo não significa nada e é apenas uma jogada de marketing utilizada para popularizar sites de redes sociais.

Nesta aula vamos mergulhar na web 2.0 e conhecer um pouco mais sobre este conceito e as tecnologias que estão envolvidas.

1. O QUE É WEB 2.0

Quando a internet começou, ela era feita de sites que publicavam conteúdo. Era uma forma digital de fazer exatamente a mesma coisa que a mídia impressa já fazia há séculos.

Desde o estouro da bolha .com em 2000 a internet vem sofrendo mudanças consideráveis:

- Os sites estáticos estão dando lugar a sites dinâmicos e interativos.
- Os internautas estão deixando de ser usuários passivos e se tornando agentes ativos em relação ao conteúdo que é gerado na internet.

Com o amadurecimento da internet, as pessoas começaram a perceber que ela é muito mais que simplesmente publicação de conteúdo em sites. Percebemos que a internet poderia ser um meio de prestar serviços. Estes serviços são prestados através de programas. Estes programas rodam em uma plataforma: a própria internet.

Até 2005 ninguém tinha uma definição clara do que realmente significava Web 2.0.

"Se houve realmente a Web 2.0, não significa que houve também uma Web 1.0?"

O uso de "2.0" implicou uma melhoria, ou uma nova geração de sites, mas não houve consenso sobre o que fez Web 2.0 diferente da Web 1.0.

Em setembro de 2005, Tim O'Reilly publicou um blog que marcou a entrada da Web 2.0. Para Tim, a filosofia da Web 2.0 inclui as seguintes idéias:

- Usar a Web como uma plataforma de aplicações;
- Democratizar a Web;
- Empregar novos métodos para distribuir informação;

Segundo Tim O'Reilly, criador do termo Web 2.0:

"*A Web 2.0 é a* ***revolução nos negócios da indústria da informática*** *causada pela mudança para a internet como plataforma, e uma tentativa de entender as regras de sucesso dessa nova plataforma.*"

Como ocorre com outros conceitos, a web 2.0 não tem uma fronteira demarcada claramente. Trata-se de um núcleo ao redor do qual gravitam princípios e práticas que aproximam diversos sites que os seguem.

Um dos princípios fundamentais é trabalhar a web como uma plataforma, viabilizando funções on-line que antes só poderiam ser realizadas por softwares instalados em um computador. É uma transição da visão de websites como unidades isoladas de conteúdo para uma estrutura integrada de conteúdo e funcionalidades.

A web 1.0 estava focada em um número relativamente pequeno de empresas e anunciantes que produziam conteúdos para usuários acessarem. A web 2.0 envolve o usuário, os usuários criam, ajudam a organizar, compartilham, criticam, atualizam os conteúdos.

Várias empresas da web 2.0 são estruturadas quase inteiramente sobre o conteúdo gerado pelo usuário e pela inteligência coletiva colaborativa. Essa estrutura de participação também pode ser vista no desenvolvimento de software através dos softwares de código aberto, onde qualquer usuário pode utilizar e modificar com pouca ou nenhuma restrição. Isso foi um fator muito importante no desenvolvimento da web 2.0. Na inteligência coletiva colaborativa, as comunidades colaboram para o desenvolvimento de softwares que são equivalentes ou até melhores que os softwares pagos.

Na web 2.0 os usuários não estão apenas contribuindo com conteúdos ou colaborando com o desenvolvimento de softwares de código aberto, mas também estão influenciando e formando opinião, direcionando a distribuição de mídia e decidindo quais canais de notícias são confiáveis.

1.1. Características da Web 2.0

Nas conferencias WEB 2.0, os seguintes princípios foram destacados como características fundamentais:

1. Web 2.0 usa a rede como uma plataforma.
2. Usuário recebe, manipula e controlados os dados no site.
3. Arquitetura participativa na qual o usuário pode adicionar ou editar total ou parcialmente uma aplicação de acordo com suas necessidades e experiência.
4. Uma interface rica, interativa e amigável, baseado em Ajax ou estruturas similares.
5. Modelos de negócios enxutos facilitados pelo gerenciamento de conteúdos e serviços.
6. Fim do Ciclo de software (versão beta)
7. Alguns aspectos de rede social.
8. Algumas interfaces gráficas, como gradientes e cantos arredondados (ausente na chamada era Web 1.0).

1.2. Alguns Conceitos Relacionados à Web 2.0

Conteúdo gerado pelo usuário: O conteúdo gerado pelo usuário tem sido a chave para muitas das empresas lideres de web 2.0. como Amazom, eBay, Monster. A comunidade agrega valor a esses sites, que, em muitos casos, são quase inteiramente criados sobre o conteúdo do usuário.

Inteligência coletiva: Inteligência coletiva é o conceito de que a colaboração pode resultar em idéias inteligentes. Trabalhando juntos, os usuários combinam seu conhecimento para o beneficio de todos.

Wikis: Wikis são sites web que permitem aos usuários editar o conteúdo existente e acrescentar novas informações, são excelentes exemplos de conteúdo gerado pelo usuário e inteligência coletiva. O wiki mais popular é o Wikipédia, uma enciclopédia gerada pela comunidade com artigos disponíveis em mais de 200 idiomas. A Wikipédia confia que seus usuários seguiram determinadas regras, como não excluir informações corretas nem acrescentar informações tendenciosas, e permite que membros da comunidade imponham as regras.

Social Bookmarking e Social News: Tendo começado como uma maneira de armazenar bookmarks online do seu navegador para que você possa utilizá-los em diferentes computadores e compartilhá-los com os amigos, bookmarking social cresceu de tal forma que agora pode ser usado para pesquisar na web em vez de confiar na tradicional motores de busca. Gerada a partir de bookmarking social, de notícias sociais é uma ótima maneira de encontrar artigos de qualidade em um oceano de notícias e participar de discussões sobre os eventos atuais, esportes, tecnologia ou outros assuntos interessantes.

Redes Sociais: Redes Sociais ou Social networking são fundamentadas em uma estrutura que permite Ã s pessoas tanto expressar sua individualidade e conhecer pessoas com interesses semelhantes. Esta estrutura inclui perfis, álbuns, amigos, blogs, widgets e outros recursos.

Blog: Um blog é um site em que os itens são colocados em uma base regular e exibidos em ordem cronológica inversa. O termo blog é uma forma abreviada de weblog ou web log. A criação de um blog, manutenção de um blog ou a adição de um artigo para um blog existente é chamada de "blogging". Artigos individuais em um blog são chamados de "posts" ou "entradas". Um blog é composto por texto, hipertexto, imagens e links (para outras páginas da web e vídeo, arquivos de áudio). Muitas vezes, os blogs se concentram em um "tema de interesse particular", como Música, Informática, Moda etc. Alguns blogs discutem suas experiências pessoais.

2. A WEB SOCIAL

A idéia de sociedade humana que se fundem com uma rede de computadores pode soar como o enredo ruim fora de um romance de ficção científica, mas é uma boa descrição do que está acontecendo em nossa sociedade hoje.

Não só estamos aumentando nosso uso da Internet â€⌂ levando em consideração o tempo passamos utilizando a internet seja em casa, no trabalho ou até mesmo no nosso bolso. Nós estamos mudando a maneira como interagimos com ela.

Isto levou-nos a uma web social, onde não estamos apenas recebendo informações, mas estamos nos conectando com outras pessoas para ouvir o que eles têm a dizer sobre o assunto.

Fazemos isso em forma de mídia social sites como blogs, redes sociais (Orkut, MySpace), de notícias sociais (Digg) e wikis (Wikipédia, a Wikia). O tema comum de cada um desses sites é a interação humana. Em blogs, postamos comentários. Em redes sociais, fazemos amigos. Em notícias votamos para artigos. E, em wikis, compartilhamos informações.

O que é Web 2.0? **Trata-se de pessoas se conectando com outras pessoas**.

TI VERDE

A TI ganhou importância quando as empresas modernas perceberam que as informações que detém fazem parte de seu patrimÃ´nio e que o modo como uma implementação informacional é efetuada em sua estrutura pode moldar toda a empresa.

Assim como outras atividades humanas, a TI provoca impactos no meio ambiente sendo tanto pela demanda de energia elétrica quanto pelos materiais utilizados na fabricação do hardware.

Neste contexto, existem empresas que adotam as ações de TI Verde suportando os negócios e outras que oferecem as soluções.

TI Verde nada mais é que um conjunto de práticas definidas para tornar mais sustentável e menos prejudicial para o meio ambiente o nosso uso da computação, visando reduzir o desperdício de recursos e aumentar a eficiência de todos os processos relacionados Ã operação dos computadores. Entre as práticas mais conhecidas estão:

Gerenciamento de energia: hoje, os processadores dos computadores gastam muito menos eletricidade que há alguns anos para realizar as mesmas tarefas. Com menos energia empregada, diminui também o calor gerado por essas máquinas. Por consequência reduz-se a necessidade de resfriamento dessas máquinas por meio de sistemas de ar-condicionado. Imagine, por exemplo, o tamanho da economia para um data center, com servidores por todos os lados. Imensa.

Virtualização: em linhas gerais, é a criação de um ambiente virtual que simula um ambiente real, tornando possível a utilização de diversos sistemas e aplicativos sem necessidade de acessar fisicamente a máquina em que estão hospedados. A principal vantagem de se virtualizar sistemas operacionais ou softwares é reduzir custos por parte das empresas, que podem economizar na compra de servidores dedicados. Também torna mais simples a realização de backups e mais rápida a manutenção das máquinas. Com menos máquinas funcionando como servidores, reduz-se a necessidade de espaço físico e cai o consumo da energia necessária para refrigerar as máquinas. A virtualização também é um dos elementos-chave para o Cloud Computing.

Interfaces Ricas para internet proporcionam maior praticidade a elaboração das tarefas cotidianas, aumentando a usabilidade e eficiência, reduzindo assim o tempo de utilização e trazendo benefícios para o meio ambiente.

3. BIBLIOGRAFIA

Plano de Aula Oficial da Disciplina "CCT0081 - Programação para Internet Rica", Centro Universitário Estácio-Radial.

Aula 2: RIA - Aplicações Ricas para Internet

Fonte: Plano de Aula Oficial da Disciplina

Objetivo: Identificar as principais características de uma Aplicação Internet Rica.

INTRODUÇÃO

A internet é, sem dúvida, uma ótima ferramenta para encontrar e organizar informações, mas no que se refere á aplicações de usabilidade, interatividade em tempo real e multimídia, está apenas começando a dar seus primeiros passos, neste ponto entra o RIA.

Ao desenvolver um projeto baseado em RIA estamos combinando a interatividade e a funcionalidade do desktop com a abrangência e flexibilidade da web para criar uma única e integrada experiência, rica em conteúdo.

1. O QUE É RIA?

O termo RIA foi usado pela primeira vez em 2001 pela Macromedia (hoje Adobe Systems) e é a abreviação de **Rich Internet Applications** ou Aplicações Ricas para Internet. É um conceito inovador no modo de pensar e desenvolver na web. Aplicações RIA, unem a funcionalidade dos softwares para desktop com o extenso alcance e facilidades econômicas de aplicativos para internet, o que proporciona um novo nível de experiências para usuários e desenvolvedores.

Tipicamente uma aplicação RIA transfere todo o processamento da interface para o navegador da internet, porém mantém a maior parte dos dados no servidor de aplicação (como por exemplo, o estado do programa, dados do banco).

A principal ferramenta para o desenvolvimento de RIA foi, inicialmnete, o Flash, porém o Flash por si só não é capaz consultar bases de dados ou fazer qualquer operação no servidor sem o auxílio de algum recurso externo, ou seja, tecnologia *Remoting* (Flex). Mesmo com suas fabulosas vantagens, o Flash possui esta restrição séria: o servidor.

O servidor deve ter suporte à tecnologia Remoting, o que torna o serviço de hospedagem para esses sites caro e inviável para pequenas empresas e sites pessoais. Em resposta a essa limitação, e dentro de um modelo de co-criação, grupos de desenvolvedores criaram uma alternativa Open-Source que viabiliza o uso de tecnologia Flash Remoting em servidores sem custos extras no servidor (necessitando somente do php, ou tecnologia Java J2EE).

Mais recentemente o foco do desenvolvimento de RIA tem se deslocado do Flash para algo mais novo: o AJAX (Asynchronous JavaScript and XML), cujo uso vem crescendo ano a ano.

Por outro lado, é importante ressaltar que a web, ainda hoje, é um ambiente onde é problemática a entrega de aplicações que exigem interfaces mais robustas e níveis mais altos de interatividade. A plataforma web foi concebida sem maior consideração a possíveis necessidades de execução de aplicativos. Praticamente todas as aplicações para a web carregam novamente as páginas inteiras, a cada clique, o que dificulta a experiência do usuário e reduz o potencial de aplicações.

Basicamente, a web é um ambiente onde o código é rodado principalmente no servidor, levando a custos maiores de transmissão de dados. Além disso, a experiência do usuário é inferior a aplicativos de desktop, com custos de desenvolvimento decorrentes de múltiplas plataformas e metodologias.

Dada a escalabilidade da web e sua maior abrangência, a implementação de aplicativos baseados na web substituiu os sistemas tradicionais cliente-servidor apresentando, porém, resultados finais limitados decorrentes dos problemas expostos anteriormente (carregar a pagina inteira novamente, por exemplo). O conceito RIA veio para tentar suprir tais limitações encontradas.

2. DESAFIOS NO ESTADO ATUAL DA WEB

* **Experiência do usuário**: em uma comparação com aplicações desktop existem limitações quanto à interface gráfica dos sistemas disponível para uso de seus respectivos usuários.

* **Aumenta os custos de centralizar as operações de dados**: pelo fato do processamento da lógica de interface no HTML em aplicações web, ser centralizada e realizada no servidor.

* **Desafios de distribuição e manutenção**: com aplicações baseadas em DHTML quando suportam múltiplos web browsers em múltiplos sistemas operacionais, onde essas aplicações devem ser desenvolvidas e mantidas garantindo que funcionem em diferentes web browser de diferentes sistemas operacionais.

3. CARACTERÍSTICAS DA INTERNET RICA

Experiência dinâmica para o usuário

* Interfaces GUI intuitivas que prevêem uma experiência de single-page (uma única página) sem os refresh das aplicações HTML web
* Uma integração sem emendas com a maioria de tipos de media em um único canvas sem separação por plugins ou por camadas
* Suporta notificação e mensagens em tempo real

Um desenvolvimento rápido da aplicação

* Um desenvolvimento familiar com os paradigmas de programação e fluxo de trabalho
* Possibilitar um desenvolvimento baseado em equipe
* Simplificar a manutenção a longo prazo do código

Disponibilização dirigida a padrões para qualquer lugar

* Compatibilidade com o prevalecimento das infra-instrutoras padrões existente (J2EE, XML, Web Services, SSL)

* Rodar em qualquer web browser sem download suplementar ou instalações

* Desloca a sobre carga do processamento dos servidores centrais de dados, ou seja, o processamento dos dados também é realizado no cliente e não totalmente centralizados no servidor.

RIA são aplicações implementadas no servidor e que tiram vantagem da tecnologia cliente para prover uma nova classe de web sites interativos com a sofisticação de aplicações de desktop, mas que não comprometem a facilidade de desenvolvimento, implementação e manuseio dos aplicativos web.

Os aplicativos de Rich Internet são baseados na tecnologia Rich Client, que fornece um ambiente dinâmico, com capacidade de hospedagem de aplicativos compilados no lado do servidor recebido como arquivos através de HTTP. Os aplicativos no lado do cliente conectam-se de volta aos Back-Ends de servidores de aplicativos existentes, por meio de uma arquitetura assíncrona de Cliente/Servidor que oferece segurança, estabilidade e que é bem adaptada ao novo modelo orientado a serviços que está sendo promovido pela adoção de serviços web.

Comparando com aplicações web baseadas em HTML, internet rica e clientes de aplicações ricas possibilitam uma vasta melhoria no tempo de resposta da aplicação e usabilidade da aplicação.

Em cada domínio, uma experiência de alta qualidade e satisfação dos clientes traduzem na melhoria dos negócios. Para os principais fornecedores de aplicações de internet para comunicações IP ao e-commerce, um modesto aumento uniforme na utilização do serviço ou das taxas de conclusão de transação pode gerar um ganho de rendimentos significantes e redução dos custos de manutenção.

4. CONSIDERAÇÕES E BENEFÍCIOS

Considerações

A adoção crescente da tecnologia Rich Client não é uma etapa evolutiva de substituição a HTML. Consiste mais em uma aplicação da capacidade dos browsers e dispositivos com interfaces de usuário mais eficazes e responsivas. A maioria dos aplicativos "Rich" é executada no contexto de browsers, e muitos são executados dentro das páginas, junto com o conteúdo HTML. Os aplicativos "Rich" acrescentam mais recursos à internet, mas a linguagem HTML continuará a ter um papel fundamental na disponibilização de conteúdo, nas interfaces de usuário e na navegação.

Internet Rica porque pode ser executada tanto em browsers como em dispositivos, ela possibilita criar aplicativos que podem ser disponibilizados uniformemente em uma ampla gama de plataformas de conexão à internet. Além disso, como a tecnologia Rich Client possibilita o uso de elementos gráficos móveis, vídeo, áudio, comunicação bidirecional e formulários complexos, ela constitui um ambiente significativamente mais sólido para criação de interfaces de usuário de aplicativos.

As qualidades descritas nesta apresentação representam os principais pontos de evolução que possibilita a utilização do conceito de Internet Rica.

* Essencial na utilização adequada destas aplicações está o planejamento adequado, centrado no usuário e em suas metas.
* Aplicações em RIA permitem que modelos mentais e objetivos de negócios sejam mais bem explorados.
* Para obtermos o potencial representado pelas tecnologias, todas as etapas do processo têm que receber o direcionamento de um planejamento adequado, conhecedor do processo e das ferramentas.

Benefícios

* Agilidade no tempo de resposta;
* Layouts gráficos mais elaborados;
* Possibilidade de interface com animação (torna as operações mais interessantes);
* Utilização de multimídia (áudio, vídeo, mais atratividade nos sites).

O resultado final para o usuário de um sistema que utiliza RIA é de modo semelhante a um sistema cliente desktop (Essa é uma das idéias centrais de utilizar-se RIA em sistemas web).

5. SITUAÇÃO ATUAL

Atualmente ainda é muito baixo o número de sites que utilizam tecnologia RIA no Brasil, mas a tendência é aumentar rapidamente. A questão chave agora é definir aplicações RIA como estratégicas e geradoras de diferencial, capazes de melhorar as experiências dentro do marketing, vendas e relacionamento. É sem duvida uma nova forma de pensar e agir na web.

Esses frameworks prometem inovações além das atualmente encontradas em sistemas web e avigorar/aprimorar as aplicações de internet, com uma nova experiência para os usuários, historicamente limitado a aplicações desktop. Estes frameworks, algumas vezes referenciados como arquiteturas de cliente rica ou executáveis de internet, permite aos desenvolvedores de aplicações, prover serviços e negócios online, criando e disponibilizar uma nova geração de aplicações web mais eficazes para satisfação de seus clientes, aumentando a adaptabilidade de seus serviços e reduzindo, os custos de desenvolvimento e manutenção dessas aplicações.

6. VÍDEO

http://www.youtube.com/watch?v=qJfzjkUi9p0

7. BIBLIOGRAFIA

Plano de Aula Oficial da Disciplina "CCT0081 - Programação para Internet Rica", Centro Universitário Estácio-Radial.

Aula 2: Fundamentos do XHTML

Prof. Daniel Caetano

Objetivo: Introduzir a linguagem XHTML e apresentar suas principais tags.

Bibliografia: W3C, RAMALHO, 1999; BOENTE, 2006; NIELSEN, 2000.

INTRODUÇÃO

Desde sua criação até os dias atuais, a Web evoluiu muito. Grande parte desta evolução só foi possível porque a linguagem usada para descrever as páginas da Web também evoluiu.

Com o surgimento de novos recursos nesta linguagem de descrição, a maneira de utilizá-la corretamente também sofreu mudanças profundas com o passar dos anos, fazendo com que muitos profissionais ficassem desatualizados, usando técnicas há muito desaconselhadas pelo W3C, o World Wide Web Consortium, responsável pela padronização da Web.

A partir de agora, veremos a maneira "correta" de usar a linguagem de descrição de páginas Web para garantir que nossas páginas e web sites sejam flexíveis e facilmente administráveis.

1. A ORIGEM DA LINGUAGEM DE DESCRIÇÃO WEB

Conceitos Chave:

- Até 1990: SGML (Linguagem de Marcação Geral Padronizada)
  - * Documentos legíveis por humanos e por máquinas
  - * Tags: especificam elementos para a máquina
    - + Indicados por "<" e ">"
    - + Marcadores de início e fim
    - + Ex.: <QUOTE>Texto</QUOTE>
- SGML é *geral*, servindo para muitas coisas.
  - * Compreensão relativamente complexa
- Tim Berners Lee => Aplicação do SGML
  - * HTML: Linguagem de Marcação de HiperTexto.
    - + Usa "tags" como as da SGML
    - + Indicar funções etruturais de um texto:
      - = Título, subtítulo, parágrafo, endereço de e-mail...
  - * **Não usar caracteres especiais/espaços em nomes de arquivo**
  - * Arquivos com nome sempre finalizado em **.html**

A linguagem de descrição de páginas Web foi inventada no início da década de 1990, por Tim Berners Lee, baseada em uma linguagem chamada SGML (Standard Generalized Markup Language - Linguagem de Marcação Geral Padronizada). A SGML serve para criar documentos que sejam legíveis tanto por seres humanos quanto por máquinas. O método para atingir este objetivo é indicar qual a função de cada trecho de texto através de "tags", que são indicadores separados pelos caracteres "<" e ">", como por exemplo: <QUOTE>Texto</QUOTE>. Observe que existe um marcador de início e um marcador de fim, sendo que neste último um caractere "/" precede o nome do indicador.

Entretanto, como o próprio nome da SGML diz, ela é "geral", ou seja, serve para muitas coisas. Por esta razão, ela é de "compreensão" relativamente complexa tanto para humanos quanto para máquinas. Assim, quando Tim Berners Lee pensou na Web, ele fez uma *aplicação* da SGML, simplificando-a para os usos que ele tinha em mente: documentos em hipertexto. Por esta razão, Lee chamou essa linguagem de HTML: HyperText Markup Language - Linguagem de Marcação de HiperTexto.

A idéia do HTML é usar "tags" (que serão vistas em breve) para indicar qual trecho de um texto é um título, qual é um subtítulo, qual é um parágrafo, qual é um endereço de e-mail, qual é um trecho de um programa de computador, qual é uma citação, qual é um link... e assim por diante.

Um detalhe importante é que define-se que, na web, **nenhum** arquivo terá qualquer caractere especial (permitidos apenas o sublinhado "_" e o traço "-"), nem mesmo devem conter espaços ou caracteres acentuados. Um arquivo html deve sempre ter seu nome terminado por .**html**. Mais adiante serão vistos mais detalhes sobre isso.

1.1. O Formato XHTML

Com a criação do formato XML (eXtensible Markup Language) e sua grande flexibilidade, foi natural a implementação do HTML através do XML, em um formato que ganhou o nome de XHTML (eXtensible Markup Language).

Essencialmente, o XHTML é um HTML que deve atender a todas as regras do XML, ou seja, é intolerante a "erros de formação", que tornam um documento mal formado. Mas o que é um documento bem formado?

Um documento bem formado tem as seguintes características:

Todas as *tags* e parâmetros devem ser grafadas em minúsculas

Exemplo incorreto: <P CLASS="teste">Parágrafo</P>
Exemplo correto: Parágrafo

- É obrigatório o uso de *tags* de fechamento

Exemplo incorreto: Parágrafo 1
Parágrafo 2
Exemplo correto: Parágrafo 1
Parágrafo 2

- Elementos vazios também devem ser fechados

Exemplo incorreto: Quebra
Régua<hr>
<img src="imagem.gif" alt="Imagem">
Exemplo correto: Quebra
Régua<hr />

- Todas as *tags* devem ser adequadamente aninhadas

Exemplo incorreto: ênfase
Exemplo correto: ênfase

- Os valores dos atributos devem sempre estar entre aspas

Exemplo incorreto: Parágrafo
Exemplo correto: Parágrafo

Além disso, é **obrigatório** o uso do elemento raiz <html> e também a definição de um tipo de documento, com a tag <DOCTYPE>. Existem algumas mudanças adicionais, que são comentadas ao longo do texto, como a *proibição* do uso de tags de posicionamento (como) e visuais (como <font>) ou atributos como *name* e *width*. Adicionalmente, o uso do parâmetro *alt* é obrigatório na *tag* <img>.

2. A ESTRUTURA DE UM DOCUMENTO HTML

Conceitos Chave:
- Leitura automatizada => Exige estrutura de documento
  * XHTML => Tem estrutura fixa
    + Sempre se repete
- Definição do tipo de documento
  * Transitional / Frameset / Strict
- Todo o conteúdo em XHTML de uma página deve ficar na região HTML
  * <html> ... </html>
  * O que estiver fora disso pode ser ignorado pelo navegador

- Duas partes importantes:
    * Cabeçalho: <head> ... </head>
        = Informações sobre a página
        = Nada de conteúdo
    * Corpo: <body> ... </body>
        = Conteúdo apresentado pelo navegador na página em si
- Relação Cabeçalho/Corpo x Área Frame/Cliente

Computadores são bastante metódicos e, sendo assim, tudo que é feito para que um computador leia, precisa ter uma certa estrutura. Assim, antes de apresentarmos algumas das tags do XHTML, é importante apresentar a estrutura de um documento XHTML.

É importante observarmos esta estrutura com atenção - e memorizá-la, porque ela será usada com freqüência e, se a estrutura de um documento XHTML não estiver correta, corre-se o risco de o navegador não interpretá-la corretamente, apresentando-a com falhas para os usuários de nossas páginas.

A estrutura fundamental de uma página XHMTL é composta pela tag DOCTYPE e mais duas seções importantes: o cabeçalho e o corpo. Cada uma destas seções fica indicada por *tags* específicas do próprio HTML.

A primeira coisa a se fazer, é indicar no documento o tipo de documento XHTML que estamos criando, com a tag DOCTYPE. Existem 3 tipos de tipo de documento:

Transitional: usado para documentos em fase de adaptação entre HTML e XHTML;
Tag DOCTYPE para Transitional:
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EM"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

Frameset: o mesmo que transitional, mas também permite o uso de frames;
Tag DOCTYPE para Frameset:
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EM"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">

Strict: só permite o uso de elementos de estruturação e *tags* modernas.
Tag DOCTYPE para Strict:
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

Após isso, indicamos que estamos incluindo uma seção de código HTML. Isso é feito pelas tags **<html>** e **</html>**. Estas tags indicam que tudo que estiver entre elas deve ser processado como HTML. Por outro lado, teoricamente, tudo que estiver fora delas deve ser ignorado pelo navegador.

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
        [ Trecho de Código HTML ]
</html>
```

Em seguida, deve-se indicar a região do cabeçalho, delimitada pelas tags **<head>** e **</head>**. Nesta região serão indicadas muitas informações da página que *não são apresentadas na área cliente do navegador*. Mais adiante essa seção será apresentada com mais detalhes.

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
        <head>
                [ Dados do Cabeçalho ]
        </head>
        [ Mais Código HTML ]
</html>
```

Finalmente, deve ser indicada a região do corpo da página, delimitada pelas tags **<body>** e **</body>**. Esta região conterá as informações que *são apresentadas na área de cliente do navegador*. Mais adiante essa seção será apresentada com mais detalhes.

Com isso, praticamente completa-se a página Web mais simples possível - que é uma página Web vazia, que terá essa cara:

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
        <head>
                [ Dados do Cabeçalho ]
        </head>
        <body>
                [ Dados do Corpo da Página ]
        </body>
</html>
```

Resumindo e ressaltando as funções das seções de cabeçalho <*head*> e de corpo <*body*>, o conteúdo da seção <head> não é para o usuário, e sim para o programa navegador. Por esta razão, o conteúdo da seção <head> não é mostrado para o usuário. Já o conteúdo da seção <body> é o conteúdo que o navegador mostrará para o usuário como sendo a página Web.

Observe que tanto a seção <head> quanto a seção <body> estão inteiramente contidas entre as tags <html> e </html>; caso isso não fosse respeitado, a página poderia apresentar problemas em alguns navegadores. Esta ordenação de tags <html>, <head> e <body> é a estrutura fundamental de uma página Web e toda página será iniciada exatamente com esta estrutura.

Como a página que vamos criar terá seu conteúdo na língua portuguesa, é conveniente indicar isso em nosso documento, de forma que os navegadores e interfaces de busca saibam qual é a língua do conteúdo de nossa página. Isso por ser feito com o parêmetro modificador *xml:lang* na tag <html>, com o valor "pt-BR" (português do Brasil). A página ficará como apresentado a seguir.

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-BR>
        <head>
                [ Dados do Cabeçalho ]
        </head>
        <body>
                [ Dados do Corpo da Página ]
        </body>
</html>
```

Esta indicação, dentre outras coisas, permite que serviços como o Google saiba em qual língua uma página foi criada e permite que a busca seja "filtrada" por esta característica.

No Google as opções são: "todas as páginas", "só páginas em português" e "só páginas do Brasil". Esta indicação influi na segunda opção, embora não seja o único critério que o Google usa.

Com isso fica concluída a estrutura fundamental de um documento HTML; vejamos agora detalhes sobre o cabeçalho e o corpo da página.

2.1. O Cabeçalho da Página

Conceitos Chave:

- XHTML => Cabeçalho => <head> ... </head>
  - * Navegadores buscam informações: Firefox, IE, Opera... agentes...
- Informações básicas de cabeçalho:
  - * Título da Página: <title> ... </title>
    - + Barra de título do navegador, Bookmark / Favoritos
  - * Outras informações
    - + Autor, folha de estilos, códigos javascript...

Anteriormente foram apresentadas as duas principais seções de um documento XHTML, que é usado para compor uma página Web. A primeira delas é o cabeçalho e, como foi dito anteriormente, esta seção contém informações para o navegador, que pode ser o Firefox, Internet Explorer, Opera, Chrome... mas também pode ser um navegador "spider" de algum serviço de busca como o Google.

> Um navegador "spider" é um agente web que vasculha as páginas e seus links para realizar algum tipo de tarefa. No caso dos navegadores spider do Google, sua função é rastrear e indexar páginas web.

Cada um destes navegadores busca coisas distintas neste cabeçalho, mas uma coisa que todos eles investigam é o *título da página*. Em XHTML, identificamos o título da página, dentro do cabeçalho, pelas tags <title> e </title>. Esta *tag* é considerada **obrigatória** no XHTML strict. Por exemplo:

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pt-BR>
        <head>
                <title>Título da página Web!</title>
        </head>
        <body>
                [ Dados do Corpo da Página ]
        </body>
</html>
```

Gravando este texto com o nome de "pagina.html" e abrindo-o em um navegador, você poderá observar que o texto "Título da página Web!" não aparece em lugar algum na página; entretanto, ele foi parar em um lugar especial: na barra de título do navegador.

Caso você faça um "bookmark" desta página, ou seja, coloque-a nos seus "favoritos", também será este texto entre **<title> e </title>** que irá ser indicado nos favoritos.

Existem outras indicações que podem ser colocadas no cabeçalho:

- O nome do autor da página
- Palavras chave
- Tipo de codificação de caractere da página
- Instruções para o navegador (recarga ou cache, por exemplo)
- Códigos em javascript
- ...

Uma outra informação obrigatória para o XHTML strict é a definição da codificação de caracteres. O padrão é indicar que os caracteres estão em UTF-8, mas é possível identificá-los no formato do Windows:

Padrão:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

Windows:

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-18" />
```

O resultado é:

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "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-8859-18" />
        <title>Título da página Web!</title>
        </head>
        <body>
                [ Dados do Corpo da Página ]
        </body>
</html>
```

Por enquanto vamos nos limitar à essas informações, que são as obrigatórias e mais importantes de todas as informações que podem figurar no cabeçalho.

Apenas a título de exemplo, algumas das indicações possíveis:

Palavras chave: <meta name="keywords" content="palavra1, palavra2, ..." />
Autor da página: <meta name="owner" content="Nome do Autor" />
Desligar cache: <meta http-equiv="pragma" content="no-cache" />
Ícone: <link rel="icon" href="imagem.png" type="image/png" />

2.2. O Corpo da Página

Conceitos Chave:

- XHTML => Corpo => <body> ... </body>
  - * Conteúdo apresentado ao usuário pelo navegador
  - * Parágrafo: ... é uma das tags principais do XHTML
- Nota: Em XHTML, quebras de linha (enter) são ignoradas!

O corpo da página, como foi dito anteriormente, deve ser delimitado pelas tags <body> e </body>; tudo que aparecerá na página deverá estar indicado nesta região. A função da maioria das tags do XHTML é *explicar ao navegador qual é o conteúdo da página*.

Uma vez que o principal formato de conteúdo nas páginas web é o formato texto, uma das tags mais fundamentais do XHTML é a tag de Parágrafo**: **. Como um parágrafo tem um início e um fim, então existe também uma tag de fechamento de parágrafo: ** **. Assim, no exemplo a seguir por ser verificado como é possível indicar um parágrafo da maneira correta em uma página XHTML.

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "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-8859-18" />
        <title>Título da página Web!</title>
        </head>
        <body>
                <p>Texto da página Web!</p>
        </body>
</html>
```

Aqui convém ressaltar um fato importante: o navegador, ao interpretar o XHTML, ignora as quebras de linha. Isso significa que indicar isso:

```
<p>Texto da página Web!</p>
```

Tem o mesmo efeito de escrever:

```
<p>
        Texto da página Web!
</p>
```

Que tem o mesmo efeito de escrever:

```
<p>
        Texto da
        página Web!
</p>
```

Nas próximas seções e aulas veremos muitas outras tags que podem ser usadas em um documento XHTML.

3. TAGS ESTRUTURANTES ADICIONAIS

Conceitos Chave:

- Manchetes / Títulos / Subtítulos
  - * Maior para menor evidência
    - + h1, h2, h3, h4, h5 e h6
    - + <h1>Manchete</h1>
  - * Devem sempre ser usados
    - + Índices automatizados
    - + Navegadores para cegos
- Quebra de linha
  - * Intervenção pontual
    - +
  - * Não substitui <p>!
- Linha de separação horizontal
  - * Cria uma separação entre duas seções da página
    - + <hr />

Manchetes/Títulos/SubTítulos: O XHTML tem tags para especificar trechos de um texto que são manchetes, títulos ou, ainda, subtítulos. Na realidade, existem 6 níveis de títulos, sendo que o nível 1 tem maior evidência até o nível 6, de menor evidência. Por exemplo:

```
<h1>1. Seção h1</h1>
    <h2>1.1. Seção h2</h2>
        <h3>1.1.1. Seção h3</h3>
            <h4>1.1.1.1. Seção h4</h4>
                <h5>1.1.1.1.1. Seção h5</h5>
                    <h6>1.1.1.1.1.1. Seção h6</h6>
```

Isso apareceria na tela mais ou menos assim:

1. Seção h1
1.1. Seção h2
1.1.1. Seção h3
1.1.1.1. Seção h4
1.1.1.1.1. Seção h5
1.1.1.1.1.1. Seção h6

Estas tags devem ser usadas com sabedoria para indicar as diversas seções do texto da página. Bons navegadores poderão, no futuro, fazer índices automáticos com estas seções, além de manter uma compatibilidade com os já existentes navegadores para cegos, que lêem estes títulos para que a pessoa escolha qual das seções quer ouvir primeiro.

Quebra de Linha: existe uma tag no XHTML usada apenas em casos muito especiais em que se deseja forçar a quebra de uma linha em uma determinada posição. Esta tag é definida por **
**, de Break Row ou line BReak (quebra de linha, em inglês). Por se tratar de uma intervenção pontual, ela deve ser sempre fechada com a / antes do >: **
**... é incorreto escrever **
. Convém lembrar, também, que **é incorreto o uso da quebra de linha em substituição à tag de parágrafo.

** Este parágrafo será quebrado no meio, bem aqui
e continua depois **

O texto acima será apresentado da seguinte forma:

Este parágrafo será quebrado no meio, bem aqui
e continua depois

Linha de Separação: Um recurso muito usado para tornar claro quando uma seção termina e onde outra começa costuma ser a linha de separação horizontal. O HTML tem uma tag para apresentar este tipo de separador: **<hr />**, de Horizontal Ruler (Régua Horizontal em inglês). Exemplo de uso:

```
<h1>1. Seção h1</h1>
        <h2>1.1. Seção h2</h2>
<hr />
<h1>2. Seção h1</h1>
        <h2>2.1. Seção h2</h2>
```

Será apresentado da seguinte forma:

1. Seção h1
1.1. Seção h2

---

2. Seção h1
2.1. Seção h2

4. CODIFICAÇÃO DE ACENTUAÇÃO

Conceitos Chave:

- Uso de acentos
  - * E computadores que não reconhecem acentos?
  - * Por que não reconhecem?
    - + ASCII
    - + ASCII Estendido
- Soluções
  - * Indicar qual codificação está sendo usada
  - * Usar tags de acentuação XHTML
- Indicação de Codificação no Cabeçalho
  - * Tag <meta>
    - + Modificador http-equiv="Content-Type"
    - + Modificador content="..."
      - = Codificação: iso-8859-1
      - = Codificação: utf-8
- Tags de Acentuação HTML
  - * Por que usar?
  - * &__nome;
    - + á + &Àgrave;
    - + ô + ü
    - + õ + ç

Se, enquanto criamos uma página, simplesmente escrevermos um texto acentuado, sem maiores preocupações, estaremos criando uma página incompatível com o sistema de muitos usuários.

Basicamente, a razão para isso acontecer é que usamos **acentuação** no texto e, em muitos países, a língua utilizada não tem acentuação. Sendo assim, os computadores e navegadores daqueles países simplesmente não entendem o que esta acentuação significa, se não forem especialmente instruídos para isso.

Para explicar porque isso ocorre, é preciso voltar um pouco às origens dos sistemas computacionais. Como você já deve saber, todo texto digitado no computador é convertido em números, quando armazenado na memória ou em um arquivo. Cada número representa uma letra. Assim, se o número 65 representa a letra A, 66 representará a letra B, 67 representará a letra C... e assim por diante. Esta codificação é chamada "Codificação ASCII", criada por uma associação norte-americana.

Esta codificação acabou se tornando padrão mundial, mas ela só definia 128 caracteres básicos (números 0 a 127) e, como na língua inglesa não há acentos ou cedilha, estes não fazem parte da padronização. Sendo assim, cada país acabou desenvolvendo um complemento de 128 caracteres (números de 128 a 255), usando estes números adicionais para indicar seus caracteres acentuados.

No Brasil, por exemplo, o caractere "á" é representado pelo número 225 e o caractere "Ã" é representado pelo número 195. Ocorre que, como cada país fez a sua extensão, estes caracteres e seus números não são padronizados internacionalmente, e mesmo dentro de um país, em alguns casos, houve diversas versões destas codificações.

Assim, se escrevemos um "à" em um arquivo aqui no Brasil (código 224) e o abrirmos em um navegador na Rússia, ao invés de um "à" veremos um "R ao contrário", já que na Rússia o código 224 foi usado para essa letra específica do alfabeto cirílico (russo).

Apenas recentemente foi criada uma codificação internacional, chamada **UNICODE**, que existe em três variações: UTF-8, UTF-16 e UTF-32, variando o número de caracteres disponível para cada uma delas. UTF-8 tem 256 caracteres e é suficiente para a maioria das línguas ocidentais. UTF-16 possui 16384 caracteres e possui caracteres de todas as línguas do mundo, mas possui apenas um conjunto limitado de caracteres de línguas como japones e chinês. O UTF-32 tem mais de 4 bilhões de caracteres e possui representação para todos os caracteres conhecidos de línguas vivas.

Porque são usadas 3 versões? Resposta simples e direta: tamanho dos arquivos de fontes.

Para contornar este problema, é possível fazer duas coisas:

a) Indicar no cabeçalho qual é o tipo de codificação que está sendo utilizada;
b) Usar tags de acentuação XHTML

Ambos os métodos serão apresentados e, em geral, deve-se usá-los sempre.

4.1. Indicação de Codificação de Página

A primeira forma de resolver o problema é deixar claro para o navegador qual é o tipo de codificação sendo usada. A maneira de fazer isso é através de uma tag de cabeçalho chamada **<meta />**, que serve para definir algumas características do próprio documento.

Assim como já foi visto para a tag <html>, a tag <meta> também admite modificadores: *http-equiv* para indicar o nome da propriedade sendo definida e *content* para indicar o valor desta propriedade. Foge ao escopo do curso detalhar todas as possíveis configurações para a tag <meta />, mas esta em específico pode ser definida da seguinte forma:

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
```

Esta tag diz que a propriedade "Content-Type" (Tipo de Conteúdo) deve ser definida como "text/html; charset=iso-8859-1", isto é, um "*texto no formato html* seguindo a codificação de nome *iso-8859-1*". Esta codificação ISO-8859-1 é a codificação padrão que o Windows usa para computadores em língua portuguesa do Brasil.

Mas e se quiséssemos indicar que a codificação de nossa página é em UTF-8? Simples! A tag seria muito parecida:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

Diferentemente da maioria das tags vistas anteriormente, a tag <meta /> também é pontual, isto é, deve ser sempre definida com uma / antes do >.

4.2. Tags de Acentuação XHTML

Apesar da estratégia anterior ser a mais amplamente usada, não é a única e, em tese, não deveria ser a única a ser utilizada pelos desenvolvedores. Há duas razões para isso:

a) O computador do cliente pode não entender a codificação indicada
b) O computador onde a página é criada pode não usar a codificação indicada

Nestes dois casos, há um problema. No primeiro deles o usuário não verá o texto corretamente, embora você pense que está tudo bem (em seu navegador tudo aparecerá perfeitamente). No segundo, nem mesmo você verá o texto adequadamente.

Para solucionar este dilema, o **XHTML tem uma maneira de indicar caracteres de acentuação**, de forma que os navegadores do mundo todo possam apresentar os símbolos de acentuação corretamente, quase que independentemente de outros fatores.

> Caso o cliente não possua instalada nenhuma fonte com os caracteres adequados, eles aparecerão incorretamente. Entretanto, isso tem se tornado relativamente incomum, dado que a maioria dos navegadores já vem com pelo menos uma fonte Unicode, incluindo a grande maioria dos caracteres usados no mundo todo.

A maneira de indicar estes caracteres é através de um código especial, que começa com o caractere &, é seguido pela letra que recebe o acento e, finalmente, temos o nome do acento - em francês, terminando a seqüência com um caractere ;. Por exemplo:

Tipo de Acento	Exemplo	Resultado
- Agudo: &_acute;	é	é
- Crase: &_grave;	À	À
- Circunflexo: &_circ;	ô	ô
- Trema: &_uml;	ü	ü
- Tilde: &_tilde;	ã	ã
- Cedilha: &_cedil;	ç	ç

5. IMAGENS NA PÁGINA WEB

Conceitos Chave:
- Grande atrativo das páginas web x Tempo de carregamento
-
    * title:

Um dos grandes atrativos da Web sempre foi sua capacidade de apresentar imagens. O uso de imagens torna uma página mais interessante, além de permitir a explicação de um assunto de forma mais elucidativa. Por outro lado, o uso exagerado e inadequado de imagens pode tornar o carregamento de uma página excessivamente lento. Assim, é importante saber usar as imagens e usá-las com parcimônia.

A tag usada para inserir uma imagem é a tag **<img />**. A tag <img /> (de IMaGe, ou IMaGem em português) é pontual, o que significa que ela precisa terminar com **/>**. Entretanto, a tag de imagem exige pelo menos um parâmetro, que indica **onde** está essa imagem, que pode estar tanto no disco, juntamente com o arquivo XHTML da página ou até mesmo em outro lugar da web. O parâmetro usado para indicar a localização da imagem é o parâmetro **src** (de SouRCe, que significa "origem", em inglês). Assim, o uso mais comum de uma figura é feito da seguinte forma:

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "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-8859-18" />
        <title>Título da página Web!</title>
        </head>
        <body>
                <img src="aflogo.gif" />
        </body>
</html>
```

O que será apresentado da seguinte forma:

Um outro parâmetro importante (e obrigatório no modo **strict**) é o parâmetro **alt**, que indica um texto curto a ser apresentado no lugar da figura, caso o navegador esteja sendo usado em um dispositivo em que não deve mostrar figuras (seja porque o aparelho não

suporta figuras, seja por razões de segurança). O texto do *alt* é usado em navegadores para deficientes visuais, pois é o texto do *alt* que é lido para descrever a figura. Assim, para ser correta, a página deve ser definida como indicado abaixo:

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "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-8859-18" />
        <title>Título da página Web!</title>
        </head>
        <body>
                <img src="aflogo.gif" alt="Logotipo Amusement Factory" />
        </body>
</html>
```

Se for executada num navegador modo texto, será apresentada da seguinte forma:

```
[IMG] Logotipo Amusement Factory
```

Alguns navegadores permitem que as imagens sejam "desligadas" para permitir uma navegação mais rápida. Também nestes casos o texto ALT será utilizado e a imagem só será carregada se o usuário clicar no texto da imagem.

Em alguns casos, queremos definir um texto explicativo mais longo para uma figura, caso o usuário deixe o ponteiro do mouse sobre a figura por alguns instantes, quando então a explicação aparecerá em um pequeno "balão de ajuda". Isso é útil, por exemplo, quando queremos descrever quem são as pessoas em uma foto, mas o número de pessoas é muito grande e não queremos gastar "espaço" da página com isso. Para atingir este objetivo, podemos usar o parâmetro **title**, conforme indicado a seguir:

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "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-8859-18" />
        <title>Título da página Web!</title>
        </head>
        <body>
                <img src="aflogo.gif" alt="Logotipo Amusement Factory"
                        title="Logotipo da Empresa do Professor" />
        </body>
</html>
```

5.1. Acelerando o Carregamento de Páginas com Imagens

Conceitos Chave:
- Dependência do tamanho da imagem para definição do layout
- Especificação do tamanho da imagem no XHTML
  * width:
  * heigth:
- Possíveis problemas: distorções, tempos de carregamento altos etc.

Muita vezes, quando o navegador vai carregar uma página com muitas imagens, ele não é capaz de mostrar **nada** da página até que carregue todas as imagens. isso ocorre porque, para distribuir e desenhar o conteúdo textual da página, o navegador precisa primeiro saber qual será o tamanho e posição das figuras que serão apresentadas.

Entretanto, para saber o tamanho das figuras, ele precisa carregá-las primeiro, o que pode ser bem demorado, dependendo do número e do tamanho das imagens. Porém, existe uma forma de ajudar o navegador a saber o tamanho das imagens, fazendo com que ele apresente o texto da página tão logo termine de carregar o arquivo .html, antes mesmo de baixar as imagens contidas na página.

Para conseguir isso, no HTML antigo, bastava indicar a largura da imagem pelo parâmetro width e a altura da imagem, pelo parâmetro height. Esta utilização é apresentada no exemplo a seguir (em HTML 4.1):

```
<HTML LANG="pt-BR">
    <HEAD>
        <TITLE>Teste de Imagem 2</TITLE>
    </HEAD>
    <BODY>
        <P><IMG src="aflogo.gif" width="330" height="80"
        TITLE="Logotipo da Empresa do Professor" ALT="Logotipo Amusement Factory">
        Logotipo do site do professor.</P>
    </BODY>
</HTML>
```

Caso a página seja carregada em uma conexão lenta, será possível observar duas etapas. Na primeira delas, o seguinte conteúdo será apresentado:

Logotipo do site do professor.

Sendo, então, a imagem preenchida posteriormente:

Logotipo do site do professor.

É claro que, em uma página com uma única (e pequena) figura, a diferença de tempo entre a apresentação do texto é praticamente imperceptível. Entretanto, em páginas com muitas figuras grandes, a diferença é bastante sensível e recomenda-se que sempre sejam indicadas as dimensões da figura.

Esse recurso, entretanto, **deve ser evitado em XHTML strict**, embora não seja considerado incorreto. Os parâmetros **width** e **height** podem ser usados com a tag img em XHTML strict, mas é mais indicado acrescentar um parâmetro **id** à imagem e usar o **id** no CSS para indicar a largura e altura da imagem (será visto posteriormente).

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "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-8859-18" />
        <title>Título da página Web!</title>
        </head>
        <body>
                <img id="aflogo" src="aflogo.gif" alt="Logotipo Amusement Factory"
                        title="Logotipo da Empresa do Professor" />
        </body>
</html>
```

De qualquer forma, o que acontece se as dimensões forem indicadas de maneira "incorreta" no CSS? Neste caso, a imagem será redimensionada pelo navegador para aparecer do tamanho especificado. Observe o que ocorre quando indicamos uma largura duas vezes maior que o tamanho original:

Logotipo do site do professor.

Entretanto, convém lembrar que não é adequado redimensionar imagens desta forma, a não ser em casos muito especiais. Esse comentário é relevante no caso de redução de imagens. Quando ela é reduzida pelo CSS, ainda assim a imagem grande é transferida pela

rede, gastando muita banda da rede. Se a imagem irá aparecer pequena, o ideal é que ela seja reduzida por meio de um editor gráfico, como o Photoshop ou Corel Draw, de maneira que ela seja transmitida já em tamanho reduzido.

Para aumentar uma imagem o recurso pode ser interessante, mas tem limitações. O redimensionamento feito pelo navegador nem sempre tem um aspecto agradável, como poderia ser conseguido com um editor gráfico. Sendo assim, é importante analisar bem a situação antes de usar o redimensionamento das imagens pelo CSS. Em geral, se usarmos esse recurso, indicaremos o tamanho exato da imagem no CSS.

6. OS LINKS WEB

Conceitos Chave:
- Endereço Web: **www.google.com.br** (incompleto!)
  * *Home Page*: www.google.com.br/index.html
- Link: marcação que permite que um texto aponte para outra página
  * http://nome_de_computador/diretorio/arquivo.html
- Exemplo: http://www.terra.com.br/portal/index.html

Quando alguém decide que vai viajar em suas férias, uma das primeiras coisas que esta pessoa precisa observar são os seus *possíveis destinos*, não é?

Na Web trabalhamos com a metáfora da *navegação*, isto é, dizemos que nossos usuários *navegam* na Web. Se considerarmos que cada página na Web é um ponto de parada de navegação, isto é, um *ancoradouro*, precisamos indicar para quais outras páginas o usuário poderá navegar, a partir dali.

Assim, em qualquer página da Web encontraremos indicações de algumas páginas que poderão ser visitadas em seguida. Estas indicações são chamadas *ligações* ou *links*. Assim, um *link* é um elo entre a página atual e uma outra página qualquer que esteja disponível na WWW.

Este *link* pode ser apresentado como um texto ou uma imagem, mas sempre tem que indicar um *endereço na internet* (também conhecido como "ancoradouro principal"). São exemplos de endereço:

www.uol.com.br
www.terra.com.br
www.google.com.br
www.yahoo.com.br
www.hotmail.com
www.w3.org

Estes endereços simplificados indicam apenas o nome de um computador na rede, e não o nome da página. Quando um endereço é fornecido neste formato, um programa que roda no servidor da página, o *servidor web*, carrega uma página padrão, chamada *home page*.

Cada servidor web pode definir uma página diferente como *home page*, mas o comportamento mais comum é o seguinte: sempre que for indicado apenas o nome de um computador ou o nome de um computador e um diretório, o servidor web irá carregar a página **index.html**. Assim, no caso do Google, por exemplo, é possível o endereço www.google.com.br, e isso fará com que nosso navegador aporte no seguinte ancoradouro:

www.google.com.br/index.html

E este sim é o nome completo do *endereço web*, ou seja, o *endereço da página*. A forma completa de especificar um endereço principal é:

nome_de_computador/diretorio/arquivo.html

Observe como ele indica o caminho a ser seguido até chegar ao arquivo de uma página web. No caso do Terra, o endereço completo da home page é:

www.terra.com.br/portal/index.html

Neste caso, o endereço indica "abra o arquivo **index.html** que se encontra no diretório **portal** do servidor **www.terra.com.br**".

Serão estes *endereços* que usaremos para indicar os possíveis destinos a partir de uma página web.

6.1. Definindo Links em XHTML

Conceitos Chave:
- Marcação de Link: <a>...
    * href: Texto do Link
    * Vai para o UOL

Ok, então cada arquivo HTML na web tem um endereço específico e precisamos usá-lo para "*linkar*" uma página. Um link será um marcador, definido pela tag **<a>** ... **** (de *Anchor*, ou âncora), que faz uma **referência a um documento XHTML** (Html REFerence, ou HREF). A tag <a> tem início e fim, pois o texto (ou imagem) que estiver marcado por ela se tornará um *link*!

Nota: o nomê da tag <a>, Âncora, vem da já revelada analogia com a navegação.

Assim, todo link para uma outra página tem pelo menos 3 componentes: a indicação da tag **a**, o endereço de uma outra página indicada pelo modificador **href** e o texto (ou imagem) que representará o link, como indicado no trecho de código-exemplo a seguir:

```
<a href="http://www.uol.com.br/">Um Link</a>
```

O que será apresentado pelo navegador assim:

Um Link

O código completo segue:

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "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-8859-18" />
        <title>Título da página Web!</title>
        </head>
        <body>
                <p>
                        <a href="http://www.uol.com.br/">Vai para o UOL</a>:
                        UOL, um dos maiores portais do Brasil na Internet.
                </p>
        </body>
</html>
```

Este código será apresentado da seguinte forma:

Vai para o UOL: UOL, um dos maiores portais do Brasil na Internet.

E, clicando no link, o navegador será redirecionado corretamente para o site do UOL! Bastou indicar corretamente o endereço do ancoradouro destino e pronto!

Convém aqui fazer um comentário acerca do texto que é convertido em link. Muitas vezes vemos na Internet links como "Clique Aqui" ou "Download". Todos que programam páginas web fazem isso, mas é importante lembrar que esse tipo de coisa deve ser evitada.

A primeira razão para isso é filosófica: na filosofia do hipertexto, você não deve ter que modificar um texto para inserir links. Em outras palavras, o texto deve ser escrito como se não existisse link algum, e os links deviam ser naturalmente associados à palavras já existentes.

A segunda razão para isso é social: navegadores para deficientes visuais e deficientes físicos, que dependem da voz para selecionar links, normalmente apresentam uma lista de links em separado (seja na forma textual, seja na forma verbal), e o usuário dita qual dos links quer entrar. Agora, imagine se a lista de links tiver essa cara:

Clique Aqui
Clique Aqui
Clique Aqui
Download
Clique Aqui
Clique Aqui
Download
Clique Aqui
Download

Fica um tanto complicado de selecionar, não é? Por estas razões, tentemos sempre usar textos elucidativos nos links. Acredite em uma coisa: se o usuário está habituado à internet (e hoje todos estão), se existe um trecho diferenciado no meio de um texto, ele já sabe que é um link e sabe que pode clicar lá. Ninguém precisa dizer para ele "clique aqui".

6.2. Links com Títulos Explicativos

Conceitos Chave:
- title: ...
    * Tamanho de arquivo, tempo de download, detalhamento do conteúdo etc.

Assim como foi visto no caso das figuras, é possível adicionar um texto explicativo ao link, de forma que este texto só seja apresentado, dentro de um pequeno "balão", se o usuário mantiver o ponteiro do mouse sobre o link por alguns instantes.

Este tipo de texto é interessante para que o usuário obtenha mais informações sobre o que vai encontrar "do outro lado do link", ou seja, no próximo ancoradouro, para saber se vale a pena navegar até lá. Um uso comum e útil deste recurso é indicar o formato de um arquivo de mídia, seu tamanho em megabytes, tempo de download, dentre outros.

Para conseguir isso, usa-se o parâmetro modificador **title** dentro da tag <a>.... Ele é usado da seguinte forma:

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "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-8859-18" />
        <title>Título da página Web!</title>
        </head>
        <body>
                <p>
                        <a href="http://www.uol.com.br/" title="Universo OnLine">
                        Vai para o UOL</A>:  UOL, um dos maiores portais do Brasil.
                </p>
        </body>
</html>
```

O que será apresentado da seguinte forma:

Vai para o UOL: UOL, um dos maiores portais do Brasil.

Observe que a aparência não muda em nada. Entretanto, se repousarmos o ponteiro do mouse por alguns segundos sobre este link, uma balãozinho de informações aparecerá com o texto "Universo OnLine", que definimos como o título do link.

7. LINKS INTERMEDIÁRIOS

Conceitos Chave:
- Ponto de Ancoragem: local onde o navegador pode parar dentro de uma página
- Todo ponto de ancoragem tem um nome.
    * Ex.: http://www.endereco.com/pagina.html#nome_da_posicao
    * Ex.: http://pt.wikipedia.org/wiki/Engenharia_de_software#Processo_de_Software

Como você deve ter observado, todos os links apresentados aqui levam ao topo de uma página. Por outro lado, você já deve ter observado que alguns sites usam links que apontam diretamente para conteúdo **no meio** de uma página. Como isso é possível?

O processo é exatamente o mesmo de criar um link tradicional, mas é preciso adicionar uma informação a mais no endereço da página, uma informação que indique para o navegador até onde ele deve rolar a página.

Isso é feito com o indicador "jogo da velha", isto é, o símbolo #. Este símbolo deve ser seguido do **nome** do ponto no qual o navegador deve parar a rolagem, chamados "***pontos de ancoragem***". Abaixo segue um exemplo genérico:

www.endereco.com/pagina.html**#nome_da_posicao**

Mais adiante veremos como indicar um *ponto de ancoragem* em uma página, mas no momento vejamos como usar o indicador #. Vamos ver isso com base em um exemplo. A página de Engenharia de Software na WikiPedia (http://pt.wikipedia.org/wiki/Engenharia_de_software) tem um ponto de rolagem chamado **Processo_de_Software**.

Assim, para indicar o link **direto** para o ancoradoro de nome "Processo_de_Software", basta indicar o endereço como se segue:

http://pt.wikipedia.org/wiki/Engenharia_de_software#Processo_de_Software

No documento HTML, isso fica da seguinte forma:

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "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-8859-18" />
        <title>Título da página Web!</title>
        </head>
        <body>
        <p>
        <a href="pt.wikipedia.org/wiki/Engenharia_de_software#Processo_de_Software">
        Processo de Software</a>: Direto para a WikiPedia.
        </p>
        </body>
</html>
```

O que será apresentado da seguinte forma:

Processo de Software: Direto para a WikiPedia.

7.1. Definindo um Ancoradouro no Meio de uma Página Web

Conceitos Chave:
- id:
    * Deve ser colocado exatamente na posição que ficará no topo da página

Bem, parece simples adicionar o link para o meio de uma página, mas e se quisermos definir um ancoradouro no meio de nossa própria página?

Isso também é bastante simples e, para isso, também se usa a tag **<a>**, que marcará o ponto em que o navegador irá parar de rolar a página**.** Neste caso, entretanto, não será usado o parâmetro href e sim o parâmetro **id**, que definirá o nome deste ponto, ou seja, o nome que deverá ser indicado no endereço Web para que o navegador role até este ponto da página.

Neste caso, mesmo que não haja texto, **é obrigatório** "fechar" a *tag* com o ****. Assim, para definir um ancoradouro intermediário chamado "Meu_Perfil" no meio de uma página, use a seguinte construção XHTML:

pagina.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EM"
        "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-8859-18" />
        <title>Título da página Web!</title>
        </head>
        <body>
        <p>     Aqui come&ccedil;a a p&aacute;gina, com uma s&eacute;rie de
                informa&ccedil;&otilde;es interessantes que se estendem por uma
                centena de linhas e tal.      </p>
        <a id="Meu_Perfil"></a>
        <p>     Aqui come&ccedil;a o seu perfil.    </p>
        <p>.</p><p>.</p><p>.</p><p>.</p><p>.</p><p>.</p><p>.</p><p>.</p>
        <p>.</p><p>.</p><p>.</p><p>.</p><p>.</p><p>.</p><p>.</p><p>.</p>
        <p>.</p><p>.</p><p>.</p><p>.</p><p>.</p><p>.</p><p>.</p><p>.</p>
        <p>.</p><p>.</p><p>.</p><p>.</p><p>.</p><p>.</p><p>.</p><p>.</p>
        </body>
</html>
```

O que será apresentado da seguinte forma, se for digitado seu endereço, como por exemplo http://www.aluno.com/index.html :

Aqui começa a página, com uma série de informações interessantes que se estendem por uma centena de linhas e tal.

Aqui começa o seu perfil.

Note que nada apareceu no lugar em que definimos o ancoradouro. Entretanto, se o usuário entrar na sua página com o endereço http://www.aluno.com/index.html#Meu_Perfil, o resultado será o apresentado abaixo:

Aqui começa o seu perfil.

O usuário poderá rolar a tela para cima e ainda verá o texto introdutório, mas ao entrar, ele caiu diretamente em seu perfil.

8. LISTAS EM XHTML

Conceitos Chave:
- Listas Ordenadas
- Não-Ordenadas
- Listas de Definições

Um recurso bastante usado do XHTML é a capacidade de exibir listas. Existem basicamente dois tipos de listas: as ordenadas (numeradas) e as não ordenadas (listas de "bullets"). Por exemplo:

Lista Ordenada:

1. Primeiro nível
1.1. Segundo nível.
2. Primeiro nível novamente.
...

Lista Não Ordenadas:

- Primeiro item
- Segundo item
- Terceiro item
...

Existem ainda as Listas de Definição:

Microprocessador
    Circuito usado em computadores para processar dados.
Sinal
    Pulso de tensão elétrica específico para acionar um dispositivo.
...

8.1. Listas Não Ordenadas

Conceitos Chave:
- Tag demarcador de Lista Não-Ordenada
    * ...
- Tag demarcador de Item de Lista
    * ...
- Exemplo

Uma lista não ordenada sempre será demarcada pelas tags **
** ... **
** (UL = Unordered List, ou Lista Não Ordenada). Adicionalmente, cada elemento de lista deve ser delimitado pelas tags **
** ... **
** (LI = List Item, ou Item de Lista).

Assim, se quisermos indicar uma lista não ordenada, em XHTML a especificaremos da seguinte maneira:

```
<ul>
        <li>Um item.</li>
        <li>Outro item.</li>
        <li>Mais outro item.</li>
</ul>
```

O que será apresentado assim:

- Um item.
- Outro item.
- Mais outro item.

É possível indicar uma lista dentro de outra:

```
<ul>
        <li>Um item.</li>
        <ul>
                <li>Um sub-item.</li>
                <li>Outro sub-item.</li>
        </ul>
        <li>Mais outro item.</li>
</ul>
```

O que será apresentado assim:

- Um item.
  - Um sub-item.
  - Outro sub-item.
- Mais outro item.

8.2. Listas Ordenadas

Conceitos Chave:
- Tag demarcador de Lista Ordenada
  * ...
- Tag demarcador de Item de Lista
  * ...

As listas ordenadas são exatamente iguais às listas não-ordenadas, mas ao invés de serem demarcadas pelas tags <ul> ... , são demarcadas pelas tags **
** ... **
** (OL = Ordered List, ou Lista Ordenada). Os elementos de lista também devem ser delimitados pelas tags **
** ... **
**. Assim, se quisermos indicar uma lista ordenada, em XHTML a especificaremos assim:

```
<ol>
        <li>Um item.</li>
        <li>Outro item.</li>
        <li>Mais outro item.</li>
</ol>
```

O que será apresentado assim:

1. Um item.
2. Outro item.
3. Mais outro item.

É possível indicar uma lista dentro de outra:

```
<ol>
        <li>Um item.</li>
        <ol>
                <li>Um sub-item.</li>
        </ol>
        <li>Mais outro item.</li>
        <ul>
                <li>Outro sub-item.</li>
        </ul>
</ol>
```

O que será apresentado assim:

1. Um item.
   1. Um sub-item.
2. Mais outro item.
   - Outro sub-item.

8.3 Listas de Definição

Conceitos Chave:
- Tag demarcador de Lista de Definição
  * <dl> ... </dl>
- Tag demarcador de Termos
  * <dt> ... </dt>
- Tag demarcador de Descrição
  * <dd> ... </dd>
- Exemplo

As listas de definição são usadas, por exemplo, para fazer glossários. Sua função é apresentar termos e sua explicação. A lista deve ser demarcada pelas tags **<dl>**...**</dl>** (de Definition List). Os termos são demarcados pelas tags **<dt>**...**</dt>** e a descrições por **<dd>**...**</dd>**.

```
<dl>
        <dt>Microprocessador</dt>
        <dd>Circuito usado em computadores para processar dados.</dd>
        <dt>Sinal</dt>
        <dd>Pulso de tensão elétrica específico para acionar um dispositivo.</dd>
</dl>
```

O que será apresentado da seguinte forma:

Microprocessador
        Circuito usado em computadores para processar dados.
Sinal
        Pulso de tensão elétrica específico para acionar um dispositivo.

9. TAGS DIVERSAS DE MARCAÇÃO

Além das tags já apresentadas, existe ainda um importante conjunto de tags a serem apresentados. Serão abordadas, a seguir, a maior parte delas, incluindo a importante tag de tabelas. Entretanto, um conjunto muito importante de tags, as de formulário, serão deixadas para o futuro.

<abbr>...</abbr> - Usado para indicar uma abreviatura, como por exemplo, <abbr>Prof.</abbr>.

<acronym>...</acronym> - Usado para indicar uma sigla, como por exemplo, <acronym>GNU</acronym>.

<address>...</address> - Usado para marcar o endereço (de e-mail, por exemplo) do autor da página. Por exemplo: <address>Rua do Limoeiro, 37</address>.

<base>...</base> - Muda a referência dos links de uma página. Pode ser usado com modificador HREF ou TARGET.

<bdo>...</bdo> - Especifica a direção do texto. O modificador *dir* pde ter os valores **rtl** ou **ltr**.

<big>...</big> - Usado para fazer com que um trecho do texto seja apresentado em letras maiores, ressaltadas.

<blockquote>...</blockquote> - Usado para marcar citações exatas longas.

<cite>...</cite> - Usado para marcar um texto como uma citação (média).

<code>...</code> - Usado para marcar um texto como sendo um código de programação.

<comment>...</comment> ou **<!- ... ->** - Usados para comentários que não devem ser exibidos pelo navegador.

~~...~~ - Usado para marcar um trecho do texto como não sendo mais válido (riscado).

<dfn>...</dfn> - Usado para marcar a definição de um termo.

** ... ** - Usado para marcar logicamente uma seção dentro de uma página HTML. Seu uso é muito importante e será visto nas aulas seguintes.

... - Usado para marcar um texto de forma que ele seja enfatizado.

<ins>...</ins> - Marca um texto que deve ser adicionado ao texto. Usado, normalmente, junto com os delimitadores ~~...~~.

<kbd>...</kbd> - Marca um texto que deve ser digitado pelo usuário, em alguma situação.

<link>...</link> - (tag de cabeçalho) Usado para indicar associação do documento atual com algum outro. Em geral usado para indicar folhas de estilo.

<meta>...</meta> - (tag de cabeçalho) Usado para indicar informações sobre a página web.

<noscript>...</noscript> - Usado para indicar um texto avisando ao usuário que o navegador dele precisa de suporte a script para que a página funcione.

<object>...</object> - Insere um elemento externo na página web, como um plugin, por exemplo.

<pre>...</pre> - Usado para marcar um texto pré-formatado. Dentro desta região, os "enters" do texto serão interpretados pelo navegador como quebras de linha.

<q>...</q> - Usado para citações exatas curtas.

<samp>...</samp> - Usado para marcar um texto como sendo um exemplo de código de programação.

<script>...</script> - (Tag de Cabeçalho) Serve para indicar um script para ser usado na página.

<small>...</small> - Usado para fazer com que um trecho do texto seja apresentado em letras menores.

... - Usado para marcar que trecho de um texto deve estar bastante ressaltado.

<style>...</style> - Usado para indicar um estilo de formatação dentro da própria página html. Evitar. É melhor usar folhas de estilo externas.

_{...} - Usado para colocar índices inferiores (subscrito).

^{...} - Usado para colocar índices superiores (sobrescrito).

<var>...</var> - Usado para marcar uma palavra como uma variável de programa ou uma parte variável de um texto.

10. TAGS DEPRECIADAS ("PROIBIDAS")

... - Texto em negrito. Não é uma tag proibida, mas deve ser evitada. Prefira .

... Texto em itálico. Não é depreciada, mas deve ser evitada. Prefira .

<applet>...</applet> - Insere um elemento externo na página web, como um plugin, por exemplo. Tag depreciada. Use <object>.

<basefont> - Modifica a fonte padrão da página. Tag depreciada. Use CSS.

** **...** ** - Centraliza o texto. Tag depreciada. Use CSS.

<dir>...**</dir>** - Marca uma lista de diretório. Tag depreciada. Use <ul>.

****...**** - Muda a fonte usada em um texto. Tag depreciada. Use CSS.

<iframe>...</iframe> - Carrega uma outra página em uma área da sua página. Não é exatamente padrão, embora seja suportado pela maioria dos navegadores. Tag depreciada. Use CSS.

<menu>...**</menu>** - Marca uma lista de menu. Tag depreciada. Use <ul>.

<nobr>...</nobr> - Usado para indicar um trecho de texto que o navegador não deve quebrar no fim de linha. Tag depreciada. Use CSS.

<s>...</s> Corta um texto. Esta tag é depreciada. Use <del>.

<strike>...</strike> Corta um texto. Esta tag é depreciada. Use <del>.

<u>...</u> Texto sublinhado. Esta tag é depreciada. Use ou , de acordo com a situação.

<wbr>...</wbr> - Usado para indicar locais possíveis de quebra de palavra. Usado normalmente dentro de um <nobr>...</nobr>. Tag depreciada. Use CSS.

11. TABELAS

Um recurso muito útil e importante no XHTML - porém freqüentemente muito mal utilizado - é o de apresentação de tabelas. Para que uma tabela seja apresentada adequadamente e rapidamente pelo navegador, ela precisa estar completamente definida, algo que muitos programadores XHTML se esquecem de fazer.

Uma tabela é basicamente demarcada pelas tags **<table>** ... **</table>**. Dentro destas tags, temos três seções: a seção **<thead>...</thead>**, onde devem ser colocados os cabeçalhos da tabelas, a seção **<tbody>** ... **</tbody>**, onde ficam as linhas de informação da tabela e também a seção **<tfoot>...</tfoot>,** onde devem ficar o rodapé da tabela. Estes marcadores de seções são opcionais, mas são altamente recomendados para facilitar a aplicação de estilos no futuro. É comum que se remova, entretanto, a região <tfoot>, por não ter função dentro de uma tabela específica.

Logo em seguida à tag <table> e antes de qualquer outra, deve ser indicada a tag **<caption>** ... **</caption>**, que serve para indicar a legenda da tabela. Com estes elementos posicionados, o código fica como especificado a seguir:

```
<table>
        <caption>Tabela 1: Uma tabela</caption>
        <thead>
        ...
        </thead>
        <tbody>
        ...
        </tbody>
        <tfoot>
        ...
        </tfoot>
</table>
```

Dentro de cada uma das seções <thead>, <tbody> ou <tfoot> cada linha da tabela tem sua estrutura própria e deve ficar demarcada pelas tags **<tr>** ... **</tr>** (Table Row, ou Linha de Tabela). Assim, se nossa tabela terá 3 linhas, podemos escrever sua estrutura da seguinte forma:

```
<table>
        <caption>Tabela 1: Uma tabela</caption>
        <thead>
        <tr>
        ... [ linha 0 ]
        </tr>
        </thead>
```

```
        <tbody>
        <tr>
        ... [ linha 1 ]
        </tr>
        <tr>
        ... [ linha 2 ]
        </tr>
        <tr>
        ... [ linha 3 ]
        </tr>
        </tbody>
</table>
```

Dentro das linhas iremos colocar as "células" de nossa tabela. Uma célula pode ser de um de dois tipos: uma **célula título** ou uma **célula de dados**. No primeiro caso, delimitamos a informação com as tags **<th>** ... **</th>** (de Table Heading). No segundo, em caso de células de dados, delimitamos a informação com as tags **<td>** ... **</td>** (de Table Data).

Assim, se na primeira linha tivermos títulos de coluna e nas outras duas linhas tivermos dados, numa tabela com 2 colunas, o código fica:

```
<table>
        <caption>Tabela 1: Uma tabela</caption>
        <thead>
        <tr>
                <th>Título Coluna 1</th>
                <th>Título Coluna 2</th>
        </tr>
        </thead>
        <tbody>
        <tr>
                <td>Coluna 1, Linha 1</td>
                <td>Coluna 2, Linha 1</td>
        </tr>
        <tr>
                <td>Coluna 1, Linha 2</td>
                <td>Coluna 2, Linha 2</td>
        </tr>
        </tbody>
</table>
```

O que será apresentado da seguinte forma (lembrando que, por padrão, as linhas das tabelas não vão aparecer. Veremos como acrescentar as linhas posteriormente):

Tabela 1: Uma tabela

Título Coluna 1	Título Coluna 2
Coluna 1, Linha 1	Coluna 2, Linha 1
Coluna 1, Linha 2	Coluna 2, Linha 2

Lembrando aqui que é possível colocar uma tabela dentro de outra, como é apresentado no código a seguir.

```
<table>
	<caption>Tabela 2: Uma tabela com outra dentro</caption>
	<thead>
	<tr>
		<th>Título Coluna 1</th>
		<th>Título Coluna 2</th>
	</tr>
	</thead>
	<tbody>
	<tr>
		<td>
		<table>
			<tbody>
				<tr>
					<th>Título Sub Coluna 1</th>
					<th>Título Sub Coluna 2</th>
				</tr>
				<tr>
					<td>Sub Coluna 1, Linha 1</td>
					<td>Sub Coluna 2, Linha 1</td>
				</tr>
			</tbody>
		</table>
		</td>
		<td>Coluna 2, Linha 1</td>
	</tr>
	<tr>
		<td>Coluna 1, Linha 2</td>
		<td>Coluna 2, Linha 2</td>
	</tr>
	</tbody>
</table>
```

E o resultado será como o apresentado abaixo:

Tabela 2: Uma tabela com outra dentro

Título Coluna 1	**Título Coluna 2**	
Coluna 1, Linha 1	**Título Sub Coluna 1**	**Título Sub Coluna 2**
	Sub Coluna 1, Linha 1	Sub Coluna 2, Linha 1
Coluna 1, Linha 2	Coluna 2, Linha 2	

Outra possibilidade é expandir uma linha por duas colunas, usando o modificador **colspan** dentro da tag TD ou TH. ou seja: para obter a aparência a seguir, use colspan como aparece no código em seguida.

Tabela 3: Uma tabela com coluna expandida

Título das Colunas	
Coluna 1, Linha 1	Coluna 2, Linha 1

Observe, no código a seguir, o uso de colspan dentro da tag <th>. O número 2 indica o número de colunas que aquela célula deve ocupar:

```
<table>
	<caption>Tabela 3: Uma tabela com coluna expandida</caption>
	<thead>
	<tr>
		<th colspan="2">Título das Colunas</th>
	</tr>
	</thead>
	<tbody>
	<tr>
		<td>Coluna 1, Linha 1</td>
		<td>Coluna 2, Linha 1</td>
	</tr>
	</tbody>
</table>
```

O mesmo vale para estender uma célula para ocupar mais de uma linha, bastando usar o modificador **rowspan**.

12. FORMULÁRIOS: O QUE SÃO?

Conceitos Chave:
- Tag <form>...</form>
- Parâmetros
  * action = "pagina_destino"
  * method = "post"
  * id = "form1"
- Campos possuem nome!

Formulários são conjuntos de campos de entrada de dados que permitem que o conteúdo destes campos sejam enviados para um endereço específico, onde eles serão processados.

Um exemplo de um formulário:

A tag que permite a criação de formulários é a tag <form>.

```
<form>
    [... conteúdo do formulário ...]
</form>
```

A tag de formulário precisa indicar o que precisa ser feito quando o usuário clicar no botão "enviar". Isso pode ser feito com o parâmetro "action" da tag **form**. Neste parâmetro podemos indicar uma função de javascript (veremos isso no futuro) ou uma outra página, que terá a função de processar os dados enviados (veremos isso no futuro, também).

Assim, o primeiro parâmetro pode ser especificado como indicado a seguir.

```
<form action="pagina.php">
    [... conteúdo do formulário ...]
</form>
```

Entretanto, como foi dito anteriormente, há duas formas de enviar dados: pelo método GET ("linha de comando") ou pelo método POST (usando um "arquivo de dados"). Bem, com os formulários podemos enviar por qualquer um dos modos, então é necessário indicar também o formato de envio na tag **form**, o que pode ser feito com o parâmetro method:

```
<form action="pagina.php" method="post">
    [... conteúdo do formulário ...]
</form>
```

O que diferencia o método POST do método GET? Basicamente, tudo que é enviado pelo método GET precisa ser colocado na URL. Como o tamanho de uma URL é relativamente limitado (varia de navegador para navegador, mas situa-se em torno de 4KB), se for necessário enviar muitos dados ou um arquivo, o método GET se torna inadequado. Adicionalmente, o método GET é menos seguro, pois os dados poderão expostos na URL.

Por outro lado, o método GET é mais rápido, carrega menos o servidor e, como veremos, é muito útil no debugging de aplicações Web.

Agora, imagine que uma página tem vários formulários, todos com os mesmos campos. Por exemplo:

Formulário 1: Reclamação
- Campo 1: Nome
- Campo 2: E-Mail
- Campo 3: Dados

Formulário 2: Sugestão
- Campo 1: Nome
- Campo 2: E-Mail
- Campo 3: Dados

Quando o usuário aperta "enviar", talvez o programador queira processar o campo "Dados" do formulário 2... mas como fazer isso, se ambos os formulários possuem o mesmo campo? Para isso, é comum darmos **id**s (identificações) formulários. Por exemplo:

```
<form action="pagina.php" method="post" id="form1">
    [... conteúdo do formulário ...]
</form>
```

Desta maneira, se eu quiser me referir ao conteúdo do campo "Dados" deste primeiro formulário, eu posso indicá-lo como:

form1.Dados

Isso permite diferenciar campos com o mesmo nome que estejam em formulários distintos. No segundo formulário o mesmo campo seria acessado pelo seguinte nome:

form2.Dados

Note que, até o momento, falamos apenas da tag <form>...</form>, que delimita a área do formulário. Mas um formulário não é só isso! Um formulário precisa ter campos e botões de ação!

Minimamente, um formulário deverá conter um botão "Enviar", "Aplicar", "Atualizar"... ou seja, um botão que acionará a ação indicada no "action", que, como já dito, pode ser um método javascript que processará os dados ou mesmo um envio direto a um programa no servidor, com a indicação de um endereço.

Além deste botão, diversos outros elementos podem aparecer em um formulário, como campos te texto, caixas de seleção, listas de seleção etc. Cada um deles será apresentado a seguir.

12.1. Tags de Elementos de Formulário

Conceitos Chave:

- <input />
    * Type; Name; id; Value; AccessKey; Title; Dir...
    * Disabled
- Botões
    * <input type="submit" name="Texto" />
        + Reset; Button
- Entrada de Texto
    * <input type="text" value="valor_inicial" />
        + MaxLength; ReadOnly... password
- CheckBox
    * <input type="checkbox" value="valor" />
        + Checked
- RadioBox
    * <input type="radio" value="valor" name="nome" />
        + Checked

```
- ComboBoxes
        * <select name="nome" id="id">
                <option value="valor">Texto</option>
          </select>
                + Size; Multiple; Disable
                + Selected; Disable
- Área de Texto
        <textarea>...</textarea>
                + Rows; Cols; MaxLength; ReadOnly
- <fieldset> <legend>...</legend> ... </fieldset>
- <input type="hidden" name="nome" value="valor" />
- <input type="file" name="nome" />
        + <form action="pag.php" method="post" ENCTYPE="multpart/form-data">
```

Como dito anteriormente, um formulário pode conter diversos elementos internos. Cada um destes elementos é especificado por uma tag específica. Cada um deles será especificado a seguir, mas vale ressaltar desde já que a tag **<input />** é pontual, ou seja, deve ser grafada com fechamento **/>** e que será usada para a maioria dos controles de formulários.

Para especificar o tipo de controle, a tag **input** aceita o parâmetro **type**, que indica o tipo de elemento. Outros parâmetros comuns são o name, que indica o nome do elemento, e o parâmetro **value**, que indica o valor inicial de preenchimento.

Adicionalmente, há o parâmetro id, que serve para dar uma identificação única para um elemento (para uso com a tag **label**, por exemplo) e o parâmetro **accesskey**, que indica uma tecla de atalho para o elemento. Assim, o formato básico e genérico da tag **input** é:

```
<input type="tipo" name="nome" value="valor" id="id" accesskey="tecla" />
```

Todo elemento dentro de um formulário pode ser desligado, usando o parâmetro **disabled="disabled"**. A utilidade disso surge apenas quando associarmos os formulários ao uso de javascript.

Há outros parâmetros aceitáveis, como **title**, dentre outros, que possuem o mesmo uso visto anteriormente em outras tags. Os detalhes para cada tipo de controle serão especificados nas seções seguintes.

12.1.1. Botões

Como dito inicialmente, praticamente todos os formulários precisam ter um botão de envio, para que o usuário indique quando quer que as informações sejam transmitidas. Para incluir um botão deste tipo, é usada a tag **input**, já mencionada anteriormente. A sintaxe mais comum de um botão de envio é:

```
<input type="submit" value="texto_do_botao" />
```

Dificilmente se coloca um nome em um botão de envio, dado que só deve existir um por formulário, podendo ele ser acessado por *nome_do_formulario.submit* . Este botão, por padrão, executa a ação indicada pelo campo action da tag **form**.

Um outro tipo de botão comum é o botão "reset", que limpa todos os campos de um formulário, cuja sintaxe é descrita abaixo:

```
<input type="reset" value="texto_do_botao" />
```

Os botões podem ser usados para chamar métodos específicos de javascript, mas isso deve ser definido no código javascript. Para criar botões deste tipo, usa-se o tipo "button":

```
<input type="button" value="texto_do_botao" />
```

12.1.2. Campos de Texto

Os campos de texto são compostos por uma linha onde é possível digitar um texto qualquer, sendo que cada campo de texto deve ter seu próprio nome. A sintaxe é a seguinte:

```
<input type="text" value="valor_inicial" />
```

Isso faz aparecer na página o seguinte campo:

Uma versão alternativa é o tipo "password", que esconde os caracteres digitados:

```
<input type="password" value="valor_inicial" />
```

Note que tanto o tipo "text" quanto "password" possuem dois parâmetros especiais: o parâmetro **maxlength** e o parâmetro **size**. O parâmetro **maxlength** permite especificar o maior número de caracteres que um campo aceita. O parâmetro **size** especifica o comprimento da caixa de texto (parte visível da caixa de texto).

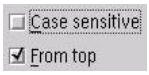
Finalmente, há o parâmetro **readonly**, que indica que o texto de um campo não pode ser modificado.

12.1.3. Caixa de Seleção (checkbox)

As caixa de seleção são controles que permitem você marcar algo como "sim" ou "não", ou seja, indicar opções que você deseja e desmarcar opções que você não deseja. A sintaxe básica é:

```
<input type="checkbox" value="valor" [checked="checked"] />
```

Onde o **checked="checked"** é um parâmetro adicional que indica se, inicialmente, a opção estará marcada ou não. O valor **value** será associado ao controle apenas se a caixinha estiver selecionada. Caso contrário, o valor associado ao controle será vazio. Exemplo de caixas de seleção:



12.1.4. Caixas de Opção (radiobox)

Os botões de opção são usados quando temos um pequeno número de alternativas fixas para uma mesma opção, e apenas uma delas pode ser selecionada de cada vez. Um exemplo de sintaxe seria:

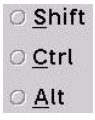
```
<input type="radio" name="sexo" id="fem" value="fem" [checked="checked"]  />
<input type="radio" name="sexo" id="masc" value="masc" />
```

Note que ambos os campos possuem o mesmo nome (name), já que ambos se referem à mesma seleção. O próprio navegador de encarrega de garantir que apenas um deles está selecionado de cada vez. Mais uma vez, a propriedade **checked** existe para indicar qual opção estará marcada inicialmente.

Um exemplo de uso de botões de opção segue abaixo:

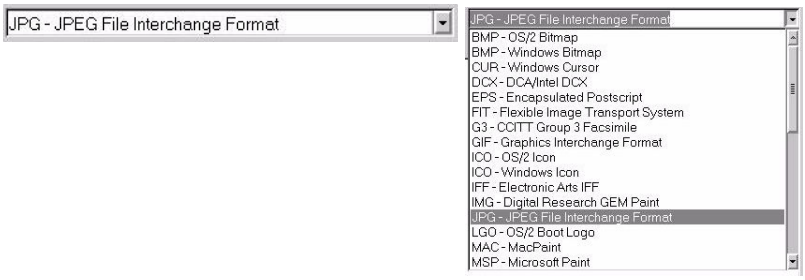
```
<input type="radio" name="key" id="shf" value="0" accesskey="S" />Shift<br />
<input type="radio" name="key" id="ctr" value="1" accesskey="C" />Ctrl<br />
<input type="radio" name="key" id="alt" value="2" accesskey="A" />Alt
```

Isso seria apresentado de maneira similar à indicada abaixo:



12.1.5. Lista de Seleção (combobox)

As listas de seleção são aquelas listas "drop-down", que apresentam apenas um valor mas, se clicarmos no botão à direita delas, uma lista maior é mostrada para que selecionemos um dos valores existentes. A seguir, temos um exemplo de uma lista de seleção:



Para criar uma lista deste tipo, a sintaxe é a seguinte:

```
<select name="nome_da_lista" id="id_da_lista>
        <option value="valor1">Valor 1: a</option>
                [...]
        <option value="valor2">Valor 2: b</option>
</select>
```

Cada linha <option>...</option> será uma opção da lista. O **value** de cada option é o valor que será associado à lista de seleção (valor selecionado) quando o conteúdo do formulário for enviado. O texto entre <option>...</option> é o texto que será apresentado na lista.

Tanto o campo de seleção <select>, quanto cada campo de opção <option> podem ser desligados com o parâmetro **disable="disable"**. Podemos ter uma caixa de seleção múltipla com o parâmetro **multiple="multiple"** na tag <select>. A tag <select> ainda aceita um parâmetro que identifica quantas linhas serão apresentadas ao clicar na lista de seleção: **size="numero_de_linhas"**. Para que uma opção inicie selecionada, basta usar o parâmetro **selected="selected"** dentro da tag <option> correspondente.

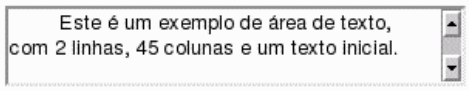
12.1.6. Área de Texto

As áreas de texto são usadas para que o usuário possa incluir um texto mais longo, da maneira que ele desejar. São usados, por exemplo, em formulários de envio de mensagens do tipo "fale conosco".

A indicação é simples, usando a tag <textarea>...</textarea>, sendo que o texto delimitado aparecerá, por padrão, na área de texto. Como o elemento de área de texto será apresentado na tela, é possível especificar seu tamanho, usando os parâmetros rows e cols para linhas e colunas respectivamente. Um exemplo pode ser verificado a seguir.

```
<textarea rows="2" cols="45">
        Este é um exemplo de área de texto, com 2 linhas, 45 colunas e um texto inicial.
</textarea>
```

E o resultado apresentado na tela será:



Este controle também aceita o parâmetro **maxlength**, que indica o número máximo de caracteres que é possível digitar na área de texto.

12.1.7. Conjunto de Controles

Em caixas de diálogo de aplicativos é muito comum que alguns controles sejam agrupados visualmente com o uso de uma borda que, eventualmente, possui um texto (legenda) no canto superior esquerdo. É possível criar este tipo de agrupamento em formulários XHTML, usando para isso a tag <fieldset>. A tag <legend> é usada para especificar o texto que aparece na borda a ser desenhada. O formato é:

```
<fieldset>
        <legend>Título da borda</legend>
                [...]
</fieldset>
```

Um exemplo mais completo segue abaixo:

```
<form>
        <fieldset>
                <legend>Dados Pessoais:</legend>
                        Name: <input type="text" SIZE="30" id="nome" /><br />
                        E-Mail: <input type="text" SIZE="30" id="email" /><br />
                        Data Nasc.: <input type="text" SIZE="10" id="nasc" /><br />
        </fieldset>
</form>
```

Que seria apresentado no documento da seguinte forma:

12.1.8. Dados "Escondidos"

Algumas vezes é preciso definir alguns valores em um formulário sem que os mesmos sejam apresentados para o usuário. Este valor pode indicar o tipo de formulário ou alguma instrução para o processamento posterior do formulário (indicando tratar-se de uma edição de dados já existentes, por exemplo).

Para este fim, existe o campo de entrada <input> do tipo **hidden**. Os parâmetros deste tipo de campo são, simplesmente, o nome do campo e o valor, respectivamente indicados pelos parâmetros name e **value**. É possível também indicar uma id para este campo. O formato padrão é indicado a seguir:

```
<input type="hidden" name="nome" value="valor" />
```

12.1.9. Campo de Envio de Arquivos

O campo de envio de arquivos deve ser especificado pela tag <input> do tipo **file**. Este controle será apresentado como um campo de texto e um botão "browse" ou "navegar", que serve para indicar um (ou vários) arquivo que será enviado junto com o restante do formulário. A sintaxe é simples:

```
<input type="file" name="arquivo" />
```

Isso, entretanto, não é suficiente para que o arquivo chegue "em paz" ao servidor. Como haverá dados binários sendo enviados com o formulário (ao invés de apenas texto), é preciso indicar na tag de declaração do formulário <form> que dados binários serão enviados também. Isso pode ser feito usando o parâmetro **enctype**, conforme indicado:

```
<form action="pagina.php" method="post" name="form1" enctype="multpart/form-data" />
    [... conteúdo do formulário ...]
</form>
```

12.2. Outras Tags para Formulários

Conceitos Chave:
- <label for="id">
- <button type="button">...</button>
- <optgroup label="Grupo">...</optgroup>
- <input type="image" src="icone.gif" width="32" height="32">

Além dos controles básicos já apresentados, existem algumas outras tags que podem ser usadas em formulários. Nesta seção veremos alguns destes tags.

12.2.1. LABEL

Em todos os exemplos apresentados, indicamos o texto de um campo como um "texto jogado". Existe uma forma mais correta de indicar os textos de campos de formuário, usando a tag **label**.

A tag <label> ... </label> marca um texto qualquer como sendo a "etiqueta" de um campo cujo id esteja definido. Isso é feito da seguinte forma:

```
<label for="id_do_elemento">Texto</label>
<input type="tipo" id="id_do_elemento">
```

Note que o valor do parâmetro **for** da tag **label** precisa ser **exatamente o mesmo** valor do parâmetro id ao qual ele se refere.

12.2.2. BUTTON

Além da tag **input** do tipo "button", que cria um botão simples, é possível usar a tag <button type="button"> em seu lugar. A tag **button**, diferentemente da tag **input**, permite coisas muito interessantes, pois ela define uma espécie de "sub página" dentro do botão. Isso significa que é como se o botão fosse um **div**, e é possível inserir qualquer tipo de código XHTML dentro deste botão. Exemplo:

```
<button type="button">
    <table>
        <tr><td>
            <img src="http://www.caetano.eng.br/main/images/aflogo_horiz.gif" alt="logo" />
        </td><td>
            <p>Um texto qualquer</p>
        </td></tr>
    </table>
</button>
```

O que será apresentado da seguinte forma:

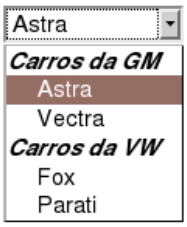
Observe a flexibilidade que esta tag proporciona, por permitir que praticamente se crie elementos XHTML dentro do botão e que, obviamente, podem ser configurados com o uso de CSS.

12.2.3. Grupos de Opção em Lista de Seleção (ComboBoxes)

Sempre que temos uma combobox com muitas opções, podemos organizá-las em grupos usando a tag <optgroup>...</optgroup>. Esta tag funciona da seguinte forma:

```
<select>
	<optgroup label="Carros da GM">
		<option value="astra">Astra</option>
		<option value="vectra">Vectra</option>
	</optgroup>
	<optgroup label="Carros da VW">
		<option value="fox">Fox</option>
		<option value="parati">Parati</option>
	</optgroup>
</select>
```

Isto será apresentado da seguinte forma:



12.2.4. Campo Imagem

O campo imagem serve para acrescentar uma figura ou ícone no formulário. Este campo é indicado com a tag **input**, com o tipo **image** identificado. Como uma imagem será apresentada, é preciso indicar o arquivo da imagem com o parâmetro **src**, além da largura e altura, usando os parâmetros **width** e **height** (que também podem ser definidos pelo CSS), conforme indicado a seguir:

```
<input type="image" src="icone.gif" width="32" height="32" />
```

Um outro uso para este tipo de campo é transformá-lo em um "botão-imagem". Para isso é preciso dar também um nome para o campo, usando o parâmetro name, para que se possa associar uma função ao evento de "clicar na imagem" usando JavaScript.

```
<input type="image" src="botao.gif" name="ok" width="32" height="32" />
```

13. BIBLIOGRAFIA

W3C: http://www.w3.org/

RAMALHO, J.A. *HTML 4 Prático e Rápido*. Editora Berkeley, 1999.

BOENTE, A. *Programação Web Sem Mistérios*. Editora Brasport, 2006.

NIELSEN, J. *Projetando Websites*. Ed. Campus, 2000.

Unidade 4: Introdução à Tecnologia CSS

Prof. Daniel Caetano

Objetivo: Apresentar conceitos de introduzir o uso de Folhas de Estilo em Cascata.

Bibliografia: W3, 2009; CASCADE, 2006; RAMALHO, 1999; NIELSEN, 2000.

INTRODUÇÃO

Conceitos Chave:
- XHTML => descrever estrutura
  * Página feia!
- Páginas modernas: XHTML + CSS
- CSS => descrever apresentação visual

Até a presente aula, vimos várias tags do XHTML que servem para descrever a função estrutural de cada trecho do texto dentro da página XHTML: o que é um título de seção, o que é um subtítulo, o que é parágrafo... e assim por diante.

Muitos alunos devem ter se perguntado por que um site feito desta forma é tão feio, e como é que outros sites na Web são tão coloridos e variados. A resposta para isso é que uma página Web atual não é feita apenas de XHTML.

Além do XHTML, que descreve os elementos existentes em uma página, é necessário também um arquivo de folhas de estilo, mais conhecido como arquivo CSS (Cascade Style Sheets ou, em bom português, Folhas de Estilo em Cascata).

Nesta e nas próximas aulas estaremos estudando como construir um arquivo CSS e como usá-lo para dar à nossa página a aparência que desejarmos.

1. FOLHAS DE ESTILO EM CASCATA (CSS)

Conceitos Chave:
- Documento de Estilos
  * Define a apresentação de cada estrutura do XHTML
  * Não dá para fazer no XHTML?
    + Praticidade e Eficiência
    + Facilidade de modificação de layout
    + Melhor uso do cache

Uma folha de estilo em cascata é nada mais do que um documento onde são definidas as características de apresentação de cada tipo de estrutura do XHTML.

Alguns alunos podem se perguntar: "Mas por que não fazer isso dentro do próprio XHTML?". Bem, originalmente era assim que as coisas eram feitas. Aliás, muita gente programa ainda daquela forma, embora a definição de apresentação dentro do XHTML seja muito pouco recomendável hoje em dia. É preciso tomar cuidado, já que a grande maioria dos livros e sites ainda ensina HTML antigo, induzindo os novos programadores aos erros dos velhos programadores.

Mas, se antigamente se usava a codificação visual dentro do próprio XHTML, por que hoje isso não é mais recomendado? Bem, as razões para isso chamam-se praticidade e eficiência.

Quando separamos os detalhes de apresentação em um arquivo separado, podemos usar o mesmo arquivo de definição de apresentação para **todas** as páginas. Em outras palavras, apesar do trabalho inicial para criar o arquivo de estilos em separado, ele **economiza** digitação nas páginas seguintes, acelerando o desenvolvimento.

Além disso, com um (ou poucos) arquivo de estilos, é muito mais fácil alterar a aparência de todo um site, de forma consistente e **sem ter que editar todas as páginas do site**!

Adicionalmente, os arquivos de CSS ficam no cache dos navegadores. Como ele é o mesmo para todas as páginas, acaba sendo uma economia de banda de transferência usá-lo, além de fazer com que o site fique mais rápido para o usuário.

1.1. Que Recursos as CSSs Possuem?

Conceitos Chave:
- Modificar a propriedade visual de qualquer elemento do XHTML
- Exemplos:
    * Fonte de texto
    * Alinhamento de texto
    * Posição de imagens
    * Cores de elementos
    * Cores de botões
    * Posições de tabelas
    * ...

Basicamente, as folhas de estilo CSS são capazes de modificar qualquer propriedade visual de qualquer coisa que apareça em uma página XHTML.

As folhas de estilo permitem modificar as fontes de texto, os alinhamentos de texto, a posição das imagens, cores dos elementos da página, cores de botões, posição de tabelas... enfim, uma infinidade de recursos.

Nesta aula serão vistos alguns destes recursos, e outros seguirão nas próximas aulas.

1.2. Porque o nome "Em Cascata"

Conceitos Chave:
- Cascata?
    * Navegador => Arquivo => No cabeçalho => Na tag
    * Classes e subclasses (opcional!)
        + <H1>
        + <H1.editorial> => herda propriedades de <H1>

As razões mais práticas para considerar o termo "Cascata" são:

1) Podemos ter quatro níveis de definição de estilo:
    a) O definido pelo navegador
    b) Em um arquivo separado
    c) No próprio arquivo XHTML, no topo da página
    d) No próprio arquivo XHTML, dentro da tag

2) Se definirmos o estilo de uma tag como <H1> e depois definirmos um estilo derivado de <H1>, como por exemplo <H1.editorial>, este estilo derivado "herdará" todas as mudanças de estilo que foram realizadas na tag <H1>.

O primeiro conceito pode ser explicado assim: normalmente o navegador tem estilos definidos (que podem ser configurados pelo usuário). O programador da página define estilos globais em arquivos separados, com a extensão .CSS. Isso permite o uso de todos os benefícios citados anteriormente para os arquivos CSS. Se quisermos que apenas em alguma página um dos estilos seja modificado, podemos redefinir este estilo no topo da página, sendo que a mudança terá efeito apenas nesta página. Se quisermos, ainda, modificar um estilo apenas em uma tag, podemos redefinir o estilo dentro dela, e terá efeito apenas na tag. Em geral, usaremos o arquivo de estilo separado, por ser a maneira mais "correta" e limpa de usar folhas de estilo.

Com relação ao segundo conceito, sua assimilação é mais fácil com o uso. Na prática é como dizer que se mudamos a fonte do H1 para Arial (ao invés de Times), automaticamente o estilo <H1.editorial> passará a ser também em Arial.

1.3. Passos para a Criação de um Arquivo de Estilo (CSS)

Conceitos Chave:
1) Criar arquivo com definições (.CSS)
2) Indicar este arquivo no cabeçalho do XHTML
3) Editar o estilo conforme desejado

São três os passos básicos para criar um arquivo de estilo:

1) Criar arquivo para inserir as definições de estilo (normalmente com extensão .css)
2) Indicar este arquivo na página XHTML, obviamente na seção <head>...</head>
3) Editar o arquivo de estilo e a página até que tudo fique como desejado.

2. USANDO ARQUIVOS CSS

Conceitos Chave:
- Primeiro passo: criar arquivo vazio (ex.: *estilo.css*)
- Indicar no XHTML
    * <link href="estilo.css" rel="stylesheet" type="text/css" />

Quando vamos criar uma página com folhas de estilo, a primeira coisa é criar um arquivo texto vazio, com um nome qualquer (por exemplo: **estilos.css**), para armazenar os estilos de uma dada página.

Criado este arquivo, temos de indicá-lo no arquivo XHTML, de forma que o navegador possa encontrá-lo e usar os estilos definidos no mesmo. Como a informação do arquivo de folhas de estilo é para o navegador, esta indicação virá dentro da seção <head>...</head>.

Esta indicação é feita da seguinte forma:

```
<link href="estilo.css" rel="stylesheet" type="text/css">
```

Notando que "estilo.css" é o nome do arquivo de estilos criado pelo desenvolvedor. **rel** e **type** são definições para que o navegador saiba o que fazer com as informações que encontrar neste arquivo.

2.1. Estrutura de um Arquivo .CSS

Conceitos Chave:

- Estrutura do arquivo .CSS

```
tag {
    propriedade1: valor1;
    propriedade2: valor2;
    ...
    propriedadeN: valorN;
}
```

- Cuidado com erros!
- Exemplo:

```
h1 {
      text-align: center;
      font-size: 3em;
      }
```

- Se houver unidades, não deixar espaço!
- Múltiplas Tags

```
tag1, tagG2, tagN {
      propriedade1: valor1;
      propriedade2: valor2;
      ...
      propriedadeN: valorN;
      }
```

Dentro do arquivo de estilo (que é um arquivo texto comum), devemos seguir estritamente uma estrutura para que ele funcione. Um erro neste arquivo e a página não funcionará corretamente.

A estrutura é a seguinte:

```
tag {
   propriedade1: valor1;
   propriedade2: valor2;
   ...
   propriedadeN: valorN;
}
```

Por exemplo:

```
h1 {
  text-align: center;
  font-size: 3em;
  }
```

Isto indica, no arquivo de estilo, que os textos do tipo <h1> serão centralizados na tela (text-align) e seu tamanho terá uma ênfase (em) de 300%, ou seja, ficará 3x maior.

NOTA IMPORTANTE: sempre que houver uma "unidade" no valor de um parâmetro ("em", no caso, é uma "unidade"), lembre-se de NÃO deixar espaço entre o número e a unidade. Caso contrário, o efeito desejado NÃO será obtido.

É possível ainda modificar duas (ou mais) tags simultaneamente, da seguinte forma:

```
tag1, tag2, tagN {
   propriedade1: valor1;
   propriedade2: valor2;
   ...
   propriedadeN: valorN;
}
```

Por exemplo:

```
h1, h2 {
  font-family: verdana, arial, sans-serif;
}
```

Que altera o tipo de fonte dos títulos e subtítulos simultaneamente.

É possível definir sub-tipos das tags também, mas isso será visto mais adiante.

3. TIPOS DE ESTILOS FUNDAMENTAIS

Conceitos Chave:
- Plano de Fundo
- Texto
- Fonte
- Bordas
- Linhas de Contorno
- Margens
- Espaço de Contorno
- Marcadores de Lista
- Propriedades de Tabelas

Os estilos não são definidos "por tag", mas sim por característica que se deseja alterar. Por exemplo: há propriedades de estilo para planos de fundo, para textos, para células de tabela... e assim por diante. Todos eles estão detalhadamente descritos no Guia de Referência CSS Volume 1.

É fortemente sugerido que tal guia de referência seja seu livro de cabeceira neste semestre.

4. ALGUNS EXEMPLOS DE ESTILOS PARA TAGS BÁSICOS

Conceitos Chave:
- body
- h1 a h6
- p
- a
- hr
- Classes e Centralização de Imagens

Obviamente, os estilos que podem ser definidos variam de acordo com a tag que está sendo modificada. Por exemplo: não faz sentido mudar propriedades de texto em uma imagem. Por essa razão, se isso for tentado, nada acontecerá.

Nesta parte serão indicados alguns tags e alguns modificadores que podem ser usados com os mesmos. Entretanto, esta lista não é, de forma alguma, exaustiva. Para maiores detalhes sobre os parâmetros e valores possíveis, consulte a seção anterior.

Tag body:

É possível mudar, por exemplo, as margens (margin-left, margin-right, margin-top, margin-bottom), a cor do fundo (background-color), imagem de fundo (background-image), se a imagem de fundo estará repetida (background-repeat), dentre outros.

Por exemplo, vamos definir um fundo com margem de 20 pixels em cada lateral da tela, com um fundo creme claro:

```
body {
  margin-left: 20px;
  margin-right: 20px;
  background-color: rgb(255,255,200);
}
```

Tags h1, h2, h3, h4, h5 e h6:

É possível mudar o alinhamento (text-align), o tamanho da fonte (font-size), o tipo de fonte (font-family), realce de fonte (font-weight), margem (margin-left, margin-right, marting-top, margin-bottom), cor (color), dentre outras.

Por exemplo, vamos redefinir h1 centralizado, com fonte 1.6x maior que o normal, usando uma fonte sem serifa (verdana, arial ou fonte sem serifa padrão do navegador), com texto em realce (mais grosso) e com cor azul escuro:

```
h1 {
  text-align: center;
  font-size: 1.6em;
  font-family: verdana, arial, sans-serif;
  font-weight: bold;
  color: rgb(0,0,50);
  }
```

Outro exemplo, vamos redefinir h2 com fonte 1.3x maior que o normal, usando uma fonte sem serifa (verdana, arial ou fonte sem serifa padrão do navegador), com texto em realce (mais grosso) e com cor azul escuro:

```
h2 {
  font-size: 1.3em;
  font-family: verdana, arial, sans-serif;
  font-weight: bold;
  color: rgb(0,0,50);
  }
```

Tag p:

É possível mudar muitas coisas da tag de parágrafo, dentre elas: o alinhamento (text-align), o tamanho da fonte (font-size), o tipo de fonte (font-family), realce de fonte (font-weight), margem (margin-left, margin-right, marting-top, margin-bottom), cor (color)...

Por exemplo, vamos redefinir P como texto "justificado", com fonte 1.2x maior que o normal, usando uma fonte com serifa (garamond, times new roman ou fonte com serifa padrão do navegador), em cor azul desbotado:

```
p {
   text-align: justify;
   font-size: 1.2em;
   font-family: garamond, times new roman, serif;
   color: #004080;
   }
```

Um desejo comum dos usuários é colocar o "espaço inicial no parágrafo" na tag <P>. Neste caso usamos o seguinte, para colocar 200% do tamanho de uma letra como "indentação":

```
p {
   text-indent: 2em;
   }
```

Tag a:

É possível mudar várias coisas também na tag de âncora (links), e é muito comum que se mude a aparência de um link. Dentre as coisas possíveis de modificar estão: o tamanho da fonte (font-size), o tipo de fonte (font-family), realce de fonte (font-weight), margem (margin-left, margin-right, marting-top, margin-bottom), cor (color), a decoração da fonte (text-decoration)...

Por exemplo, vamos redefinir **a** como usando uma fonte com serifa (garamond, times new roman ou fonte com serifa padrão do navegador), em letra mais forte (negrito) e sem sublinhado, mas em cor azul mais claro:

```
a {
   font-family: garamond, times new roman, serif;
   font-weight: bold;
   text-decoration: none;
   color: #3030D0;
   }
```

Tag hr:

Também é possível mudar muitas coisas na linha divisória. Dentre as coisas possíveis de modificar estão: a altura da barra (height), a largura da barra (width), a cor da barra (background-color), a borda (border), a imagem de fundo (background-image), a repetição da imagem (image-repeat) e assim por diante.

Por exemplo, vamos redefinir **hr** como 7 pixel de altura, 75% da largura da tela, cor de fundo azul escuro, sem borda 3D:

```
hr {
  height: 7px;
  width: 75%;
  background-color: #000480;
  border: none;
  }
```

4.1. Classes e Centralização de Imagens (OPCIONAL!)

Para centralizar uma imagem, podemos fazer o seguinte: definir um novo tipo de parágrafo <p class="centrado"> que será centralizado, e colocar a imagem dentro deste parágrafo.

Primeiramente definiremos a nova classe de parágrafo:

```
p.centrado {
  text-align: center;
  }
```

E em seguida alteramos o documento original, indicando esta versão alternativa de parágrafo no parágrafo da figura:

```
<p class="centrado">
        <img src="nome_da_imagem.jpg" alt="desc" />
</p>
```

Na verdade, o conceito de classes pode ser explorado em qualquer uma das tags. Como um exemplo, vamos definir a tag **p** de duas formas:

```
p {
   text-indent: 2em;
   text-align: justify;
   font-size: 1.2em;
   font-family: garamond, times new roman, serif;
   color: #004080;
   }

p.centrado {
   text-indent: 0;
   text-align: center;
   }
```

Sempre que usarmos <p>, teremos como resultado um parágrafo justificado e com o espaçamento inicial na margem da esquerda, tamanho 20% maior que o normal na fonte garamond (ou uma de suas alternativas) em uma cor azulada.

Sempre que usarmos <p class="centrado">, teremos como resultado um parágrafo centralizado e sem o espaçamento adicional na margem da esquerda. Como a classe "p.centrado" é uma "especialização" de "p", ela herdará as outras definições: fonte garamond 20% maior e cor azulada.

5. BIBLIOGRAFIA

CASCADE Style Sheets, level 2 revision 1: CSS 2.1 Specification - W3C Working Draft 06 November 2006. Disponível em < http://www.w3.org/TR/CSS21/ >. Visitado em 21 de Dezembro de 2006.

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

RAMALHO, J.A. *HTML 4 Prático e Rápido*. Editora Berkeley, 1999.

BOENTE, A. *Programação Web Sem Mistérios*. Editora Brasport, 2006.

NIELSEN, J. *Projetando Websites*. Ed. Campus, 2000.

EXEMPLO 1 (SEM CSS)

pir_ap04ex.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>Primeira P&aacute;gina Web com CSS</title>
</head>
<body>

	<h1>Primeira p&aacute;gina XHTML com CSS</h1>
	<hr />
	<h2>Primeira Se&ccedil;&atilde;o</h2>
	<p><img src="http://www.caetano.eng.br/main/images/aflogo_horiz.gif"
		width="330" height="80" title="Empresa do professor"
		alt="Amusement Factory Logo" />
	</p>
	<p>A primeira se&ccedil;&atilde;o ir&aacute; conter dois par&aacute;grafos.
	   Note, tamb&eacute;m, que o t&iacute;tulo desta se&ccedil;&atilde;o &eacute;
	   em H2, sendo que o t&iacute;tulo da p&aacute;gina foi em H1.
	</p>
	<p>Este &eacute; o segundo par&aacute;grafo da primeira se&ccedil;&atilde;o.
	</p>

	<h2>Segunda Se&ccedil;&atilde;o</h2>
	<p>Esta &eacute; a segunda se&ccedil;&atilde;o e tamb&eacute;m cont&eacute;m
	   dois par&aacute;grafos. O t&iacute;tulo desta se&ccedil;&atilde;o
	   tamb&eacute;m &eacute; em H2.
	</p>
	<p>Este &eacute; o segundo par&aacute;grafo da segunda se&ccedil;&atilde;o.
	</p>

	<p>Mais informa&ccedil;&otilde;es na
	<a href="http://www.caetano.eng.br/aulas/psw/" title="Disciplina
		Programa&ccedil;&atilde;o para Internet Rica">p&aacute;gina da
		disciplina de Prorgama&ccedil;&atilde;o para Internet Rica</a>.
	</p>
</body>
</html>
```

EXEMPLO 2 (Com CSS)

pir_ap04ex1.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>Primeira P&aacute;gina Web com CSS</title>
	<link href="estilo.css" rel="stylesheet" type="text/css" />
</head>
<body>

	<h1>Primeira p&aacute;gina XHTML com CSS</h1>
	<hr />
	<h2>Primeira Se&ccedil;&atilde;o</h2>
	<p><img src="http://www.caetano.eng.br/main/images/aflogo_horiz.gif"
		width="330" height="80" title="Empresa do professor"
		alt="Amusement Factory Logo" />
	</p>
	<p>A primeira se&ccedil;&atilde;o ir&aacute; conter dois par&aacute;grafos.
	   Note, tamb&eacute;m, que o t&iacute;tulo desta se&ccedil;&atilde;o &eacute;
	   em H2, sendo que o t&iacute;tulo da p&aacute;gina foi em H1.
	</p>
	<p>Este &eacute; o segundo par&aacute;grafo da primeira se&ccedil;&atilde;o.
	</p>

	<h2>Segunda Se&ccedil;&atilde;o</h2>
	<p>Esta &eacute; a segunda se&ccedil;&atilde;o e tamb&eacute;m cont&eacute;m
	   dois par&aacute;grafos. O t&iacute;tulo desta se&ccedil;&atilde;o
	   tamb&eacute;m &eacute; em H2.
	</p>
	<p>Este &eacute; o segundo par&aacute;grafo da segunda se&ccedil;&atilde;o.
	</p>
```

```
       <p>Mais informa&ccedil;&otilde;es na
       <a href="http://www.caetano.eng.br/aulas/psw/" title="Disciplina
              Programa&ccedil;&atilde;o para Internet Rica">p&aacute;gina da
              disciplina de Prorgama&ccedil;&atilde;o para Internet Rica</a>.
       </p>
</body>
</html>
```

estilo.css

```
body {
   margin-left: 20px;
   margin-right: 20px;
   background-color: rgb(255,255,200);
}

h1 {
   text-align: center;
   font-size: 1.6em;
   font-family: verdana, arial, sans-serif;
   font-weight: bold;
   color: rgb(0,0,50);
   }

h2 {
   font-size: 1.3em;
   font-family: verdana, arial, sans-serif;
   font-weight: bold;
   color: rgb(0,0,50);
   }

p {
   margin: 5px;
   text-indent: 2em;
   text-align: justify;
   font-size: 1.1em;
   font-family: verdana, arial, sans-serif;
   color: rgb(0,60,120);
   }

a {
   font-family: verdana, arial, sans-serif;
   font-weight: bold;
   text-decoration: none;
   color: rgb(50,50,200);
   }

hr {
  height: 7px;
  width: 50%;
  background-color: rgb(0,60,120);
  border: none;
  }
```

EXEMPLO 3 (Com CSS)

pir_ap04ex2.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>Primeira P&aacute;gina Web com CSS</title>
       <link href="estilo2.css" rel="stylesheet" type="text/css" />
</head>
<body>

       <h1>Primeira p&aacute;gina XHTML com CSS</h1>
       <hr />
       <h2>Primeira Se&ccedil;&atilde;o</h2>
       <p><img src="http://www.caetano.eng.br/main/images/aflogo_horiz.gif"
              width="330" height="80" title="Empresa do professor"
              alt="Amusement Factory Logo" />
       </p>
       <p>A primeira se&ccedil;&atilde;o ir&aacute; conter dois par&aacute;grafos.
          Note, tamb&eacute;m, que o t&iacute;tulo desta se&ccedil;&atilde;o &eacute;
```

```
         em H2, sendo que o t&iacute;tulo da p&aacute;gina foi em H1.
      </p>
      <p>Este &eacute; o segundo par&aacute;grafo da primeira se&ccedil;&atilde;o.
      </p>

      <h2>Segunda Se&ccedil;&atilde;o</h2>
      <p>Esta &eacute; a segunda se&ccedil;&atilde;o e tamb&eacute;m cont&eacute;m
         dois par&aacute;grafos. O t&iacute;tulo desta se&ccedil;&atilde;o
         tamb&eacute;m &eacute; em H2.
      </p>
      <p>Este &eacute; o segundo par&aacute;grafo da segunda se&ccedil;&atilde;o.
      </p>

      <p>Mais informa&ccedil;&otilde;es na
      <a href="http://www.caetano.eng.br/aulas/psw/" title="Disciplina
            Programa&ccedil;&atilde;o para Internet Rica">p&aacute;gina da
            disciplina de Prorgama&ccedil;&atilde;o para Internet Rica</a>.
      </p>
</body>
</html>
```

estilo.css

```
body {
   border-color: rgb(200,200,160);
   border-width: 20px;
   border-style: solid;
   margin-left: 100px;
   margin-right: 100px;
   background-color: rgb(240,240,220);
   padding: 10px;
}

h1 {
   margin-left: 50px;
   margin-right: 50px;
   text-align: center;
   font-size: 2.0em;
   font-family: verdana, arial, sans-serif;
   font-weight: normal;
   background-color: rgb(50,50,100);
   color: rgb(255,255,200);
   }

h2 {
   font-size: 1.3em;
   font-family: verdana, arial, sans-serif;
   font-weight: bold;
   font-variant: small-caps;
   color: rgb(0,0,50);
   }

p {
   margin: 5px;
   text-indent: 2em;
   text-align: justify;
   font-size: 1.1em;
   font-family: verdana, arial, sans-serif;
   color: rgb(0,60,120);
   }

a {
   font-family: verdana, arial, sans-serif;
   font-weight: bold;
   text-decoration: none;
   font-variant: small-caps;
   color: rgb(50,50,200);
   }

hr {
  height: 0px;
  width: 0%;
  background-color: rgb(0,60,120);
  border: none;
  }
```

Unidade 5: Posicionamento com CSS

E outras Técnicas Avançadas

Prof. Daniel Caetano

Objetivo: Apresentar recursos adicionais do CSS e como usá-lo para a construção de layout de página.

Bibliografia: W3, 2009; CASCADE, 2006; RAMALHO, 1999; NIELSEN, 2000.

INTRODUÇÃO

Conceitos Chave:
- Estilos "Avançados"
  * Transparência, links visitados etc...
- CSS => não serve apenas para mudar estilos!
  * Posicionamento de Elementos!
  * Mudar o alinhamento... só?
- Auxílio do XHTML
  * É preciso indicar o "início" e o "fim" de cada elemento.
  * É preciso ordenar estes elementos
    + Tags de Divisão de documento

Anteriormente, foi apresentado como usar as CSSs para alterar a aparência de nosso documento. Muitas propriedades do documento podem ser alteradas, como cor de fundo, cor de texto, dentre outras.

Entretanto, o mecanismo visto para modificação de propriedades é um tanto quanto limitado com relação ao posicionamento dos diversos elementos de uma página: podemos mudar alinhamentos e, talvez, a ordem de alguns elementos. Entretanto, se quisermos posicionar o conteúdo de alguma forma menos tradicional, não será possível apenas com o que já vimos até agora.

O objetivo desta aula é, então, apresentar mais alguns recursos do CSS e como definir elementos que permitam posicionar diferentes "blocos" de uma página Web no lugar desejado. Como será visto, tudo isso será feito primordialmente no CSS, com poucas modificações no documento HTML.

1. PSEUDO-CLASSES E PSEUDO-ELEMENTOS

Conceitos Chave:
- Definindo estilos específicos de ações
  * tag:estado { ... }
- Exemplos
  a:hover
  a:visited
  p:first-child

Alguns elementos do HTML, com a tag <A>...</A>, que define um link, possuem diversos "estados": um link pode ser um link não visitado, um link já visitado, um link com o ponteiro do mouse sobre ele, e assim por diante. Assim, para definir características específicar em cada um destes estados, é necessário acrescentar uma informação no nome da tag indicada no CSS.

```
elemento:estado {
        propriedade: valor;
        }
```

Estes "estados" são chamados de "pseudo-classes" e as mais comuns são:

:active	Especifica estilo para um elemento ativo (ex.: a:active)
:focus	Especifica estilo para elemento em foco (ex.: input:focus)
:hover	Especifica estilo para elemento com mouse sobre ele (ex.: a:hover)
:link	Especifica estilo para link não visitado (a:link)
:visited	Especifica estilo para link já visitado (a:visited)
:first-child	Especifica estilos diferentes internos a uma primeira ocorrência de um elemento em uma região da página.

Por exemplo: para fazer com que um link não visitado seja verde e sublinhado, mas se torne vermelho e sublinhado quando o mouse passar por cima, usa-se o seguinte CSS:

```
a:link {
        color: green;
        text-decoration: none;
        }

a:hover {
        color: red;
        text-decoration: underline;
        }
```

1.1. Pseudo-Elementos

Os pseudo-elementos são muito similares às pseudo-classes, mas definem o comportamento de elementos que não estão claramente definidos no HTML. Os pseudo-elementos mais comuns são:

:first-letter	Especifica estilo para a primeira letra de uma tag (ex.: p:first-letter)
:first-line	Especifica estilo para a primeira linha de uma tag (ex.: p:first-line)
:before	Insere algum conteúdo (áudio? vídeo?) antes de um elemento (url:)
:after	Insere algum conteúdo (áudio? vídeo?) depois de um elemento (url:)

2. TRANSPARÊNCIA

Conceitos Chave:
- Transparência/Opacidade de imagens
  * FireFox, IE 9 em diante - CSS3 (0.0 <= x <= 1.0)
    opacity: x
  * IE 5 a 7 (0 <= x <= 100)
    filter:alpha(opacity=x)
  * IE 8 (0 <= x <= 100)
    -ms-filter:"progid:DXImageTransform.Microsoft.Alpha(Opacity=X)
  * IE8 deve vir antes do IE 5-7

Algumas propriedades ainda não padronizadas pelo CSS2 já estão disponíveis nos navegadores. Estas propriedades estão padronizadas no CSS3, mas como o CSS3 não é, ainda, exatamente oficial, cada navegador implementa estas propriedades de um jeito, obrigando o programador a especificar o mesmo efeito em mais de um formato dentro do arquivo CSS.

Uma destas propriedades que é muito importante é a transparência, sendo este um dos mais desejados pelos web masters. Ele é conseguido através dos seguintes atributos:

Propriedade:	**Onde e Como Funciona:**
opacity: *x*	Firefox e IE9+
-ms-filter:"progid:DXImageTransform.Microsoft.Alpha(Opacity=x)	IE8, x de 0 a 100
filter:alpha(opacity= *x*)	IE5 a 7, x de 0 a 100

O padrão CSS3 é o mesmo adotado pelo FireFox.

Considere a página a seguir, que contém um pano de fundo, um texto em um H1 e um texto dentro de um parágrafo. Observe como o texto tem baixa legibilidade quando se encontra sobre a figura do planeta Terra, devido ao baixo contraste entre as cores da imagem e a cor usada no texto.

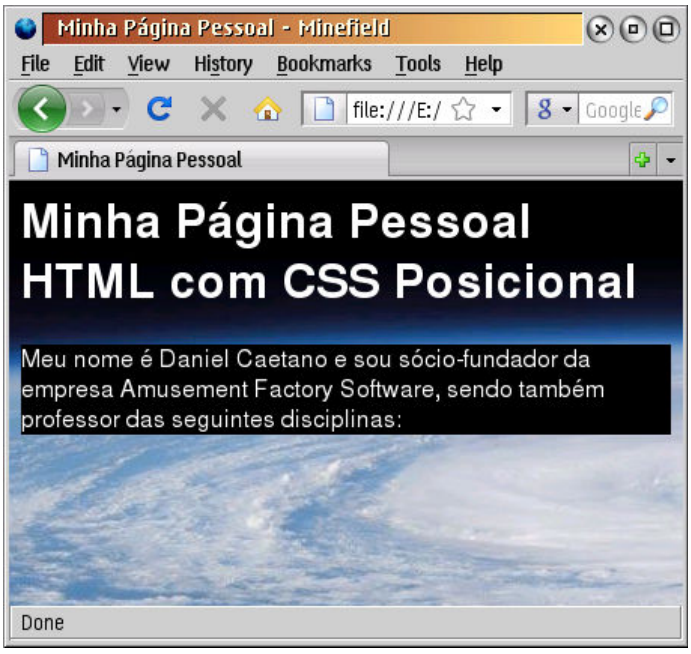
exemplo.html

```
<!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" />
      <link href="exemplo.css" rel="stylesheet" type="text/css" />
      <title>Minha página pessoal.</title>
</head>
<body>
      <h1>Minha P&aacute;gina Pessoal HTML com CSS Posicional</h1>
      <p>     Meu nome &eacute; Daniel Caetano e sou s&oacute;cio-fundador da empresa
              Amusement Factory Software, sendo tamb&eacute;m professor das seguintes
              disciplinas:
      </p>
</body>
</html>
```

exemplo.css

```
body {
  background-color: rgb(0,0,0);
  background-image: url("earth2.jpg");
  color: rgb(255,255,255);
}

p {
  background-color: rgb(0,0,0);
}
```



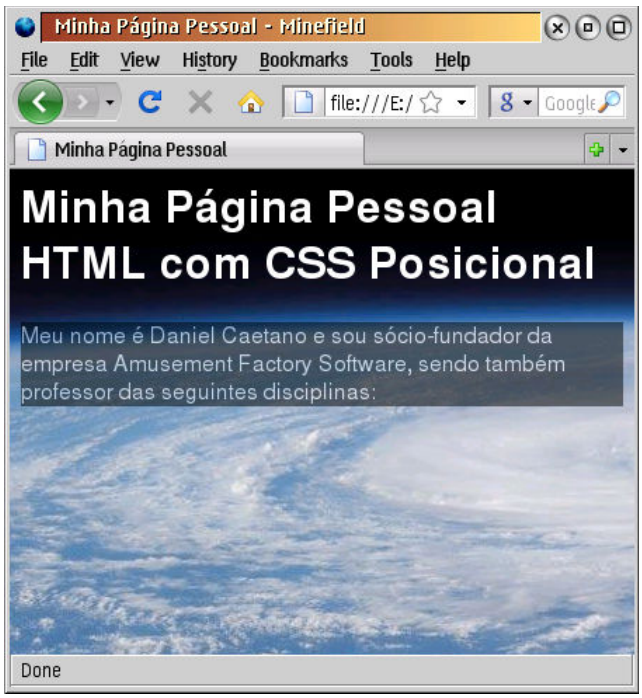
É possível que o WebMaster não queira um fundo "bloco preto" para seu parágrafo, e considere interessante poder defini-lo com um bloco preto com 50% de transparência, permitindo que o fundo seja visível através do bloco. Para isso, basta a seguinte modificação no arquivo .CSS:

exemplo.css

```
body {
  background-color: rgb(0,0,0);
  background-image: url("earth2.jpg");
  color: rgb(255,255,255);
}

p {
  background-color: rgb(0,0,0);
  opacity: 0.5;                                                    // Padrão
  -ms-filter:"progid:DXImageTransform.Microsoft.Alpha(Opacity=50)  // IE 8
  filter:alpha(opacity=50);                                        // IE5 a 7
}
```

O resultado é apresentado abaixo.



Caso não se preocupe com navegadores antigos e fora do padrão, o CSS fica:

exemplo.css

```
body {
  background-color: rgb(0,0,0);
  background-image: url("earth2.jpg");
  color: rgb(255,255,255);
}

p {
  background-color: rgb(0,0,0);
  opacity: 0.5;                                                    // Padrão
}
```

Observe, porém, que em todos os casos a transparência está sendo "herdada" pelo texto, o que a torna quase inútil... mas ela é a única que se aplica tanto a imagens quanto a cor de fundo.

Quando quisermos deixar apenas a cor de fundo de um elemento transparente, podemos definir essa cor com o elemento **rgba(vermelho, verde, azul, transparência)**. Exemplo:

exemplo.css

```
body {
   background-color: rgb(0,0,0);
   background-image: url("earth2.jpg");
   color: rgb(255,255,255);
}

p {
   background-color: rgba(0,0,0,0.5);
}
```

O resultado será esse:

3. DIVISÃO DE DOCUMENTO

Conceitos Chave:
- DIV => <DIV ID="*nome*">...</DIV>
    * Regiões a Reposicionar x Redefinir Estilos
- Exemplo

Até este momento, trabalhamos com corpo do documento HTML como se fosse um elemento único, isto é, não houve uma identificação clara do que seria "menu", "cabeçalho do conteúdo", "área de notícias", "área de conteúdo"...

Ora, se queremos posicionar cada uma destas partes de maneira independente, isto é, queremos indicar exatamente onde cada uma delas deve estar, então precisamos dividir o documento HTML em todas estas partes; entretanto, para evitar confusão e aumentar ainda mais o número de arquivos, não faremos uma divisão física. Como fazer, então?

A idéia é fazer uma divisão lógica do documento, ou seja: mantemos um único arquivo, mas indicaremos com **tags** o que cada trecho representa. Deixaremos de pensar no corpo da página como um documento único, e passaremos apensar nele como um conjunto de blocos, como se definíssemos várias páginas distintas dentro de um mesmo documento.

Pensando assim, como temos várias páginas dentro do mesmo arquivo, passa a ser natural que possamos dar características próprias a cada um dos elementos, não só de posição, mas também de cores, fontes etc. Para tanto, a única modificação que será feita no HTML é justamente a adição de algumas tags que indicarão o que cada trecho da página representa. Por exemplo, na página abaixo:

```
<!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>Título da Página</title>
</head><body>
	<h1>T&iacute;tulo da P&aacute;gina</h1>
	<hr />
	<h3>Menu</h3>
	<p>	<li><a href="...">Item 1</a></li>
		<li><a href="...">Item 2</a></li>
		<li><a href="...">Item 3</a></li>
	</p>
	<h2>Conteúdo</h2>
	<p>Esta é uma página pessoal!</p>
</body><html>
```

Claramente existe uma seção de "cabeçalho do conteúdo", que é composto pelo <H1>...</H1> e o <HR>. Em seguida, temos um menu, composto pelo <H3> e o primeiro <P>...</P> e, finalmente, temos o conteúdo da página, composto pelo <H2> e o segundo <P>...</P>. Embora seja possível perceber isso analisando o documento, isso não está claramente identificado.

A identificação clara é precisa é o que será feita agora, usando a **tag** de *DIVisão de documento*, a tag **<DIV>...</DIV>**. Como a tag DIV define uma região, ela tem um início e um final. O documento marcado com as tags fica como indicado abaixo:

```
<!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>Título da Página</title>
</head><body>
	<div>	<h1>T&iacute;tulo da P&aacute;gina</h1>
		<hr />
	</div>
	<div>	<h3>Menu</h3>
		<p>	<li><a href="...">Item 1</a></li>
			<li><a href="...">Item 2</a></li>
			<li><a href="...">Item 3</a></li>
		</p>
	</div>
	<div>	<h2>Conteúdo</h2>
		<p>Esta é uma página pessoal!</p>
	</div>
</body><html>
```

Bem, mas isso ainda não está claro o suficiente. Para que possamos dar clareza (e posteriormente modificarmos as propriedades e posição de cada região) é preciso dar um nome, uma IDentificação para cada seção. Isso pode ser feito com o parâmetro **ID="*nome_da_seção*"**. Assim, o documento anterior poderia ficar da seguinte forma:

```
<!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>Título da Página</title>
</head><body>
	<div id="titulo">
		<h1>T&iacute;tulo da P&aacute;gina</h1>
		<hr />
	</div>
	<div id="menu">
		<h3>Menu</h3>
		<p>	<li><a href="...">Item 1</a></li>
			<li><a href="...">Item 2</a></li>
			<li><a href="...">Item 3</a></li>
		</p>
	</div>
	<div id="conteudo">
		<h2>Conteúdo</h2>
		<p>Esta é uma página pessoal!</p>
	</div>
</body><html>
```

Um comentário importante é que a tag DIV só terá efeito visual no navegador se realizarmos mudanças efetivas na mesma, por meio do arquivo CSS. Caso contrário, será como se o navegador simplesmente tivesse ignorado a tag. Na página definida na aula anterior, podemos definir várias seções. Por exemplo:

** **

Minha Página Pessoal

** **
** **

ÍNDICE

1. INFORMAÇÕES PESSOAIS
2. NOTAS DE PROGRAMAÇÃO WEB
3. CONTATO

** **
** **

1. INFORMAÇÕES PESSOAIS

AMUSEMENT FACTORY

Meu nome é Daniel Caetano e sou sócio-fundador da empresa Amusement Factory Software, sendo também professor das seguintes disciplinas:

- Programação Web
- Segurança e Auditoria de Sistemas
- Pesquisa Operacional
- Arquitetura de Computadores
- Sistemas Operacionais

** **
** **

2. NOTAS DOS ALUNOS DE PROGRAMAÁÃO WEB

Tabela 1: Notas de 2009/01

NOME DO ALUNO(1)	NOTA FINAL
Sem Dados	Sem Dados

(1) Alunos do 2o. semestre de Análise e Desenvolvimento

Mais informações na PÁGINA DA DISCIPLINA PROGRAMAÇÃO WEB.

** **
** **

3. ENTRE EM CONTATO

Qualquer dúvida pode ser enviada para:
Prof. Daniel Caetano
E-Mail: daniel@caetano.eng.br

** **

Ao carregar esta página no navegador, ela será exibida exatamente como antes. Entretanto, isso só ocorre porque não realizamos qualquer mudança no arquivo CSS. Mas o que deveríamos modificar no arquivo CSS? Resumidamente, devemos indicar **o que** estas seções devem possuir de diferente!

4. MODIFICANDO SEÇÕES COM CSS

Conceitos Chave:
- Definindo estilos de seções
  * #secao { ... }
- Exemplos
  * Titulo
  * Indice
  * Secao1, secao2, secao3

Como agora não iremos mais modificar apenas a apresentação de algumas tags do HTML e sim como serão apresentados pedaços inteiros do documento, a definição no arquivo CSS é um pouco diferente daquela que vimos antes.

O formato geral é:

```
#secao {
  ...
  }
```

Como uma regra geral, também aqui o CSS é muito chato com a sintaxe. Um pequeno erro de digitação, um ponto-e-vírgula faltando ou algo do gênero pode ser responsável por sua página não aparecer corretamente. Por esta razão, muita atenção ao digitar o arquivo .CSS!

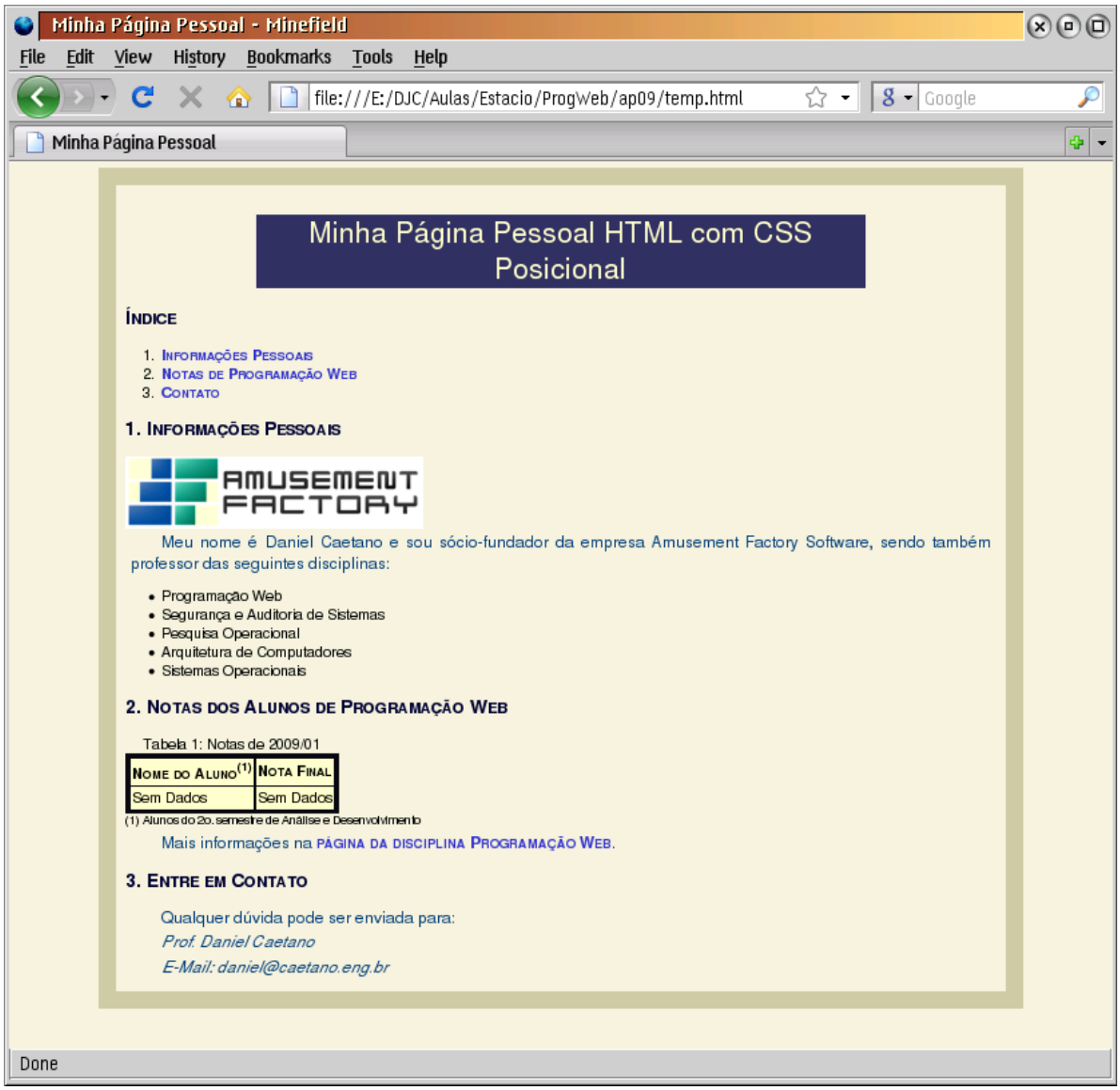
Nas próximas páginas serão apresentados alguns exemplos de modificação de propriedades e posicionamento através de CSS.

4.1. Exemplos Aplicando CSS para Posicionamento

Pegando como exemplo a página da aula anterior, para modificar a seção de título (definida como no item anterior) de forma que ela tenha um fundo azul claro e margens diferentes, devemos indicar no arquivo estilo.css como apresentado a seguir.

```
#titulo {
  background-color: rgb(50,50,100);
  margin-left: 15%;
  margin-right: 15%;
  }
```

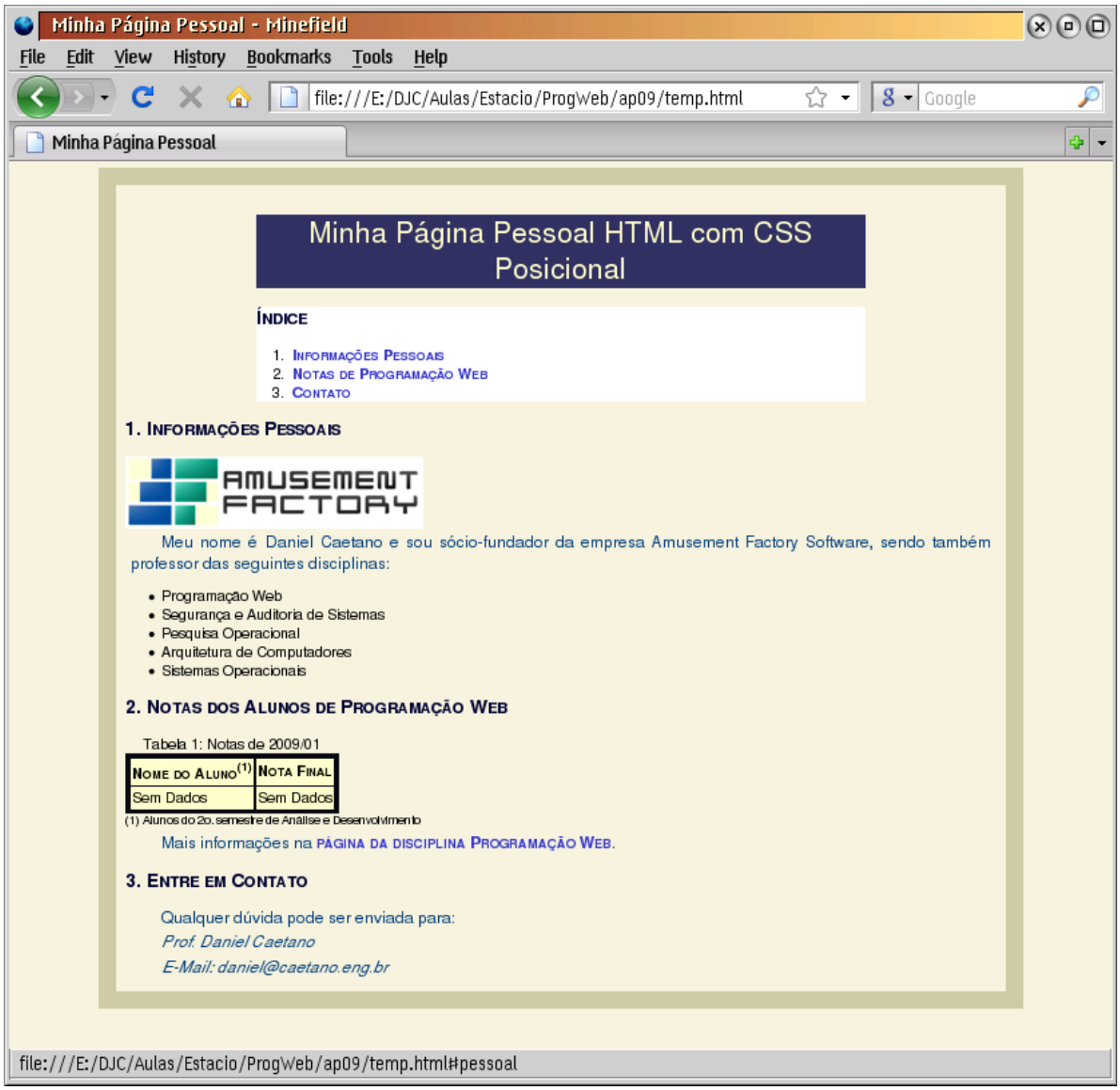
O resultado será o apresentado a seguir.



Um outro exemplo: para fazer com que o índice apareça como um retângulo de fundo branco:

```
#indice {
  margin-left: 15%;
  margin-right: 15%;
  background-color: rgb(255,255,255);
  }
```

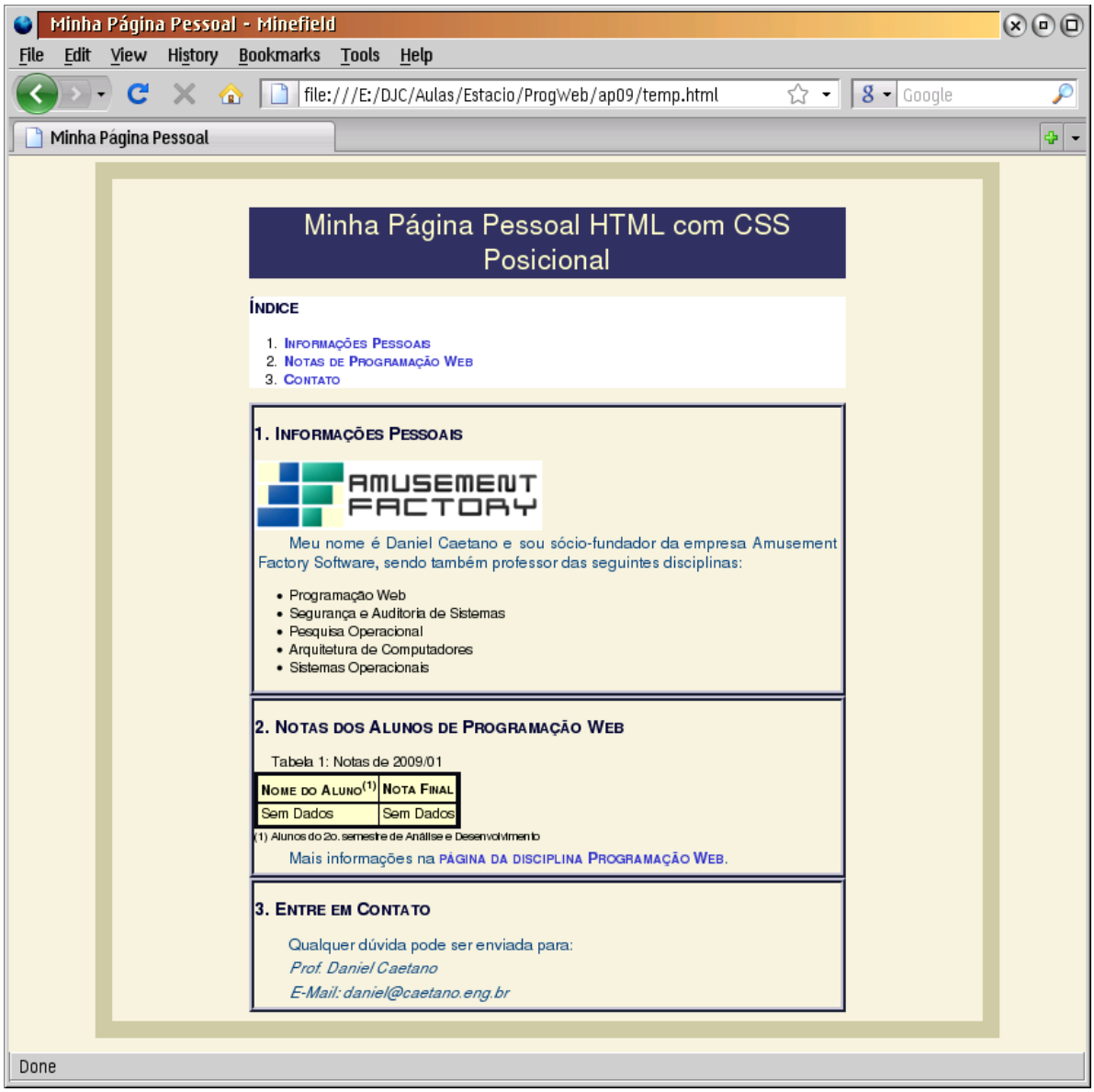
O resultado é apresentado a seguir.



Vamos movimentar fazer com que as seções 1, 2 e 3 recebam uma borda azul escura, de 6 pixels de largura, 3D arredondada:

```
#secao1, #secao2, #secao3 {
  margin-left: 15%;
  margin-right: 15%;
  border-color: rgb(50,50,100);
  border-width: 6px;
  border-style: ridge;
  }
```

O resultado é apresentado a seguir.



Como é possível observar, ficaram uns espaços "estranhos" entre a região de título, índice e as seções no FireFox. Estes espaços não aparecem no Internet Explorer (ao menos até a versão 8.0). Isso ocorre por uma pequena diferença na interpretação das margens dos títulos (H1 a H6).

Se quisermos corrigir esta diferença (e, neste caso, desejamos), basta acrescentar a propriedade "overflow: hidden" em cada uma das regiões DIV:

```
#titulo {
  overflow: hidden;
  background-color: rgb(50,50,100);
  margin-left: 15%;
  margin-right: 15%;
  }
```

```
#indice {
  overflow: hidden;
  margin-left: 15%;
  margin-right: 15%;
  background-color: rgb(255,255,255);
  }

#secao1, #secao2, #secao3 {
  overflow: hidden;
  margin-left: 15%;
  margin-right: 15%;
  border-color: rgb(50,50,100);
  border-width: 6px;
  border-style: ridge;
  }
```

Isso corrige a diferença para o FireFox, mas tem dois efeitos colaterais: se reduzirmos demais a janela, o texto ficará cortado. Da outra forma o texto não é cortado, mas sairá para fora das regiões delimitadas. Ou seja: de qualquer forma o resultado de uma janela muito pequena não será bonito. Veremos no futuro como minimizar este problema. O segundo efeito colateral é muito importante e útil, e será apresentado em breve.

Para melhorar um pouquinho o visual de nossa página, vamos acrescentar alguns espaçamentos nas seções DIV, acrescentando as linhas indicadas abaixo:

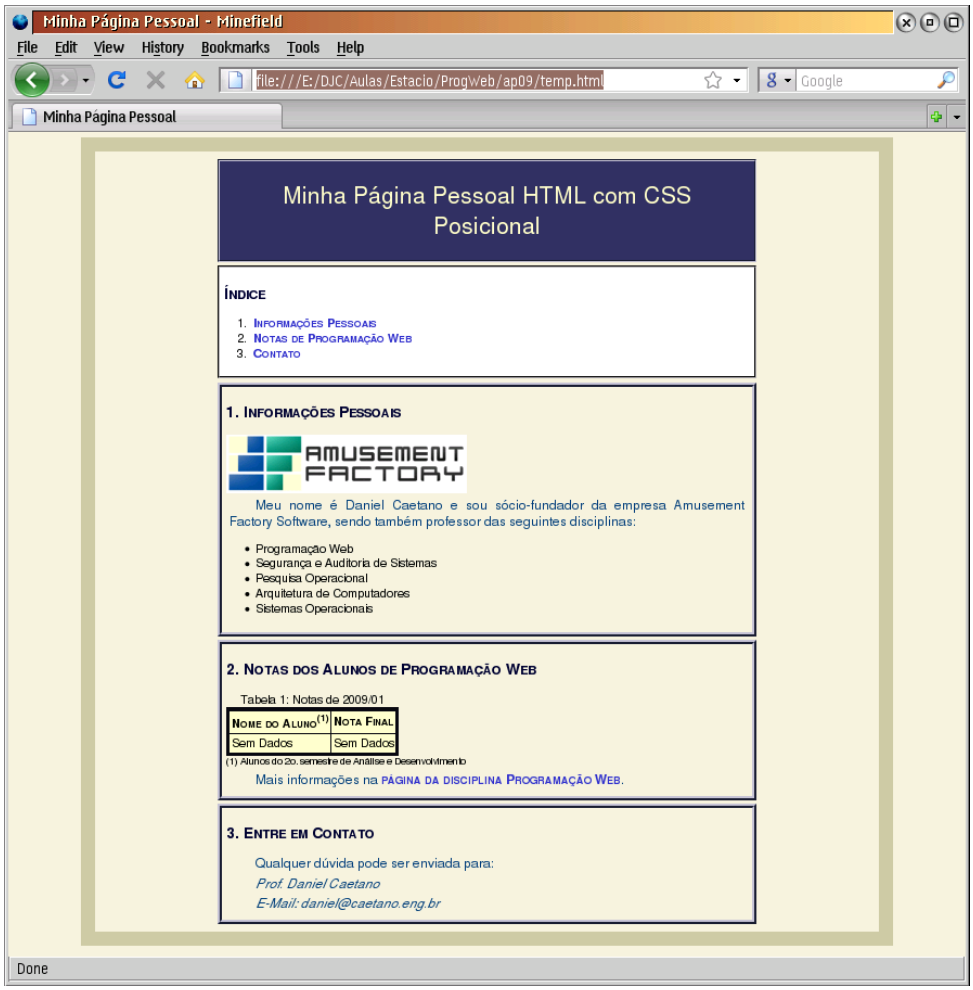
```
#titulo {
  overflow: hidden;
  background-color: rgb(50,50,100);
  margin-left: 15%;
  margin-right: 15%;
  border-style: groove;
  border-color: rgb(50,50,100);
  padding: 5px;
  }

#indice {
  overflow: hidden;
  margin-left: 15%;
  margin-right: 15%;
  background-color: rgb(255,255,255);
  margin-top: 5px;
  border-style: groove;
  padding: 5px;
  }

#secao1, #secao2, #secao3 {
  overflow: hidden;
  margin-left: 15%;
  margin-right: 15%;
  border-color: rgb(50,50,100);
  border-width: 6px;
```

```
    border-style: ridge;
    margin-top: 5px;
    padding: 5px;
    }
```

E o resultado obtido é:



Certo, usamos alguns efeitos interessantes. Mas o layout não mudou muito com relação ao que tínhamos antes. Isso ocorre porque apenas criamos as regiões com os DIVs e mudamos algumas poucas propriedades de cores e posição delas, mas não alteramos significativamente o *fluxo de informações* da página.

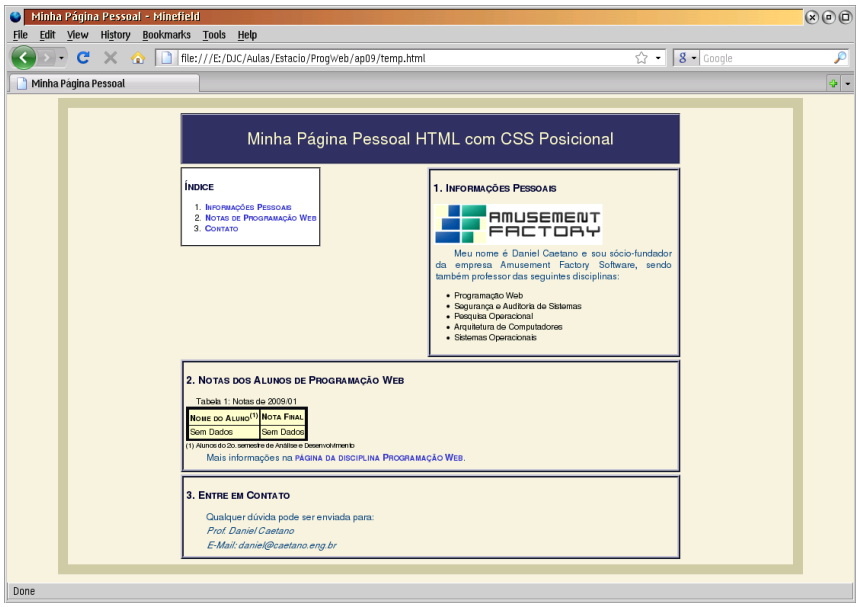
Isso significa que os dados estão sendo dispostos da mesma maneira que o padrão, isto é, de cima para baixo, um elemento em baixo do outro. Esse é o fluxo normal do HTML e do CSS.

Se quisermos, por exemplo, que o índice fique do lado esquerdo da tela, com informações ao seu lado direito, teremos que mudar o fluxo de apresentação das informações na tela. Quando queremos fixar um elemento em um dos lados e permitir texto do outro, dizemos que este elemento está **flutuando**, fixo a um dos lados. A propriedade que permite a

um elemento flutuar é a a propriedade **float**, que pode receber como valor as duas laterais: left (esquerda) ou right (direita). Vamos fazer com que o índice fique flutuando à esqueda:

```
#indice {
   float: left;
   overflow: hidden;
   margin-left: 15%;
   margin-right: 15%;
   background-color: rgb(255,255,255);
   margin-top: 5px;
   border-style: groove;
   padding: 5px;
   }
```

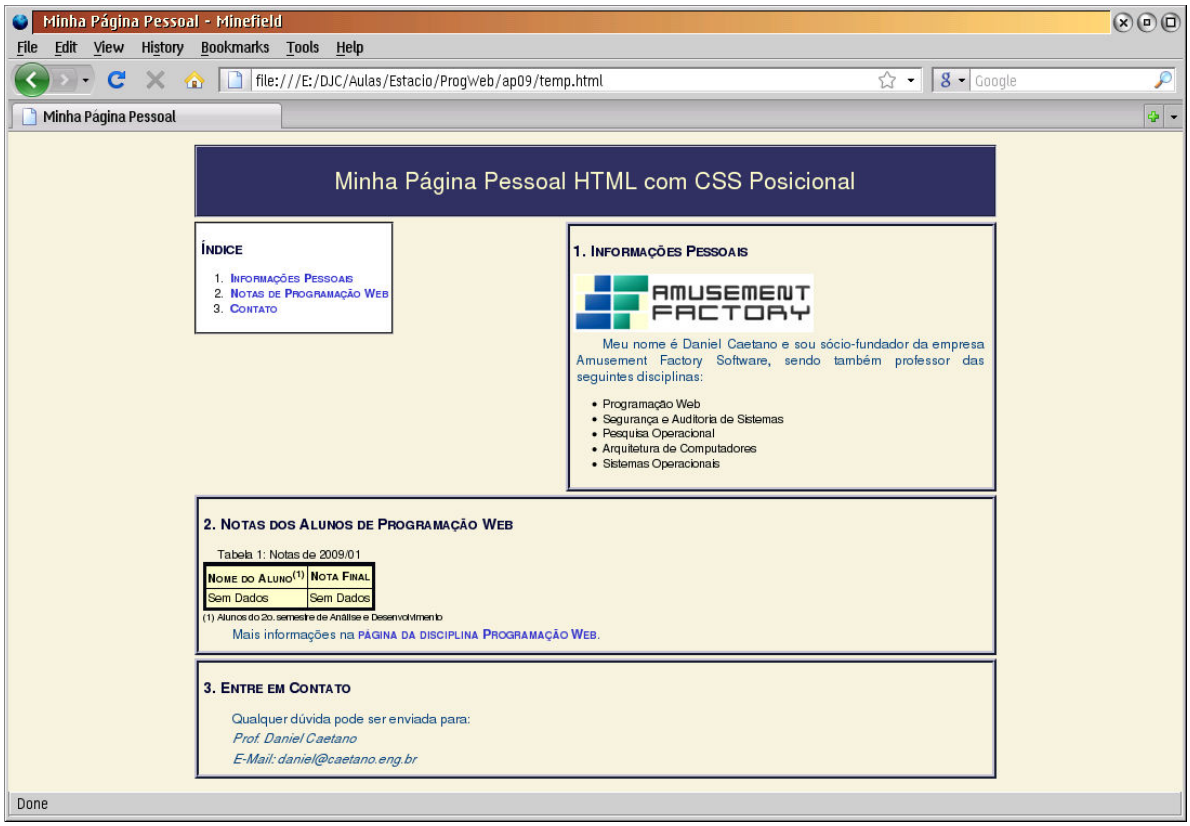
Faça a modificação e veja o que ocorreu:



Interessante, não? Pois bem, para ganharmos um pouco de espaço, vamos agora eliminar a borda que havíamos colocado no corpo da página antiga, e reduzir as margens:

```
BODY {
   font-family: verdana, arial, sans-serif;
   border-color: rgb(200,200,160);
   border-width: 20px;
   border-style: solid;
   margin-left: 10px;
   margin-right: 10px;
   background-color: rgb(240,240,220);
   padding: 10px;
}
```

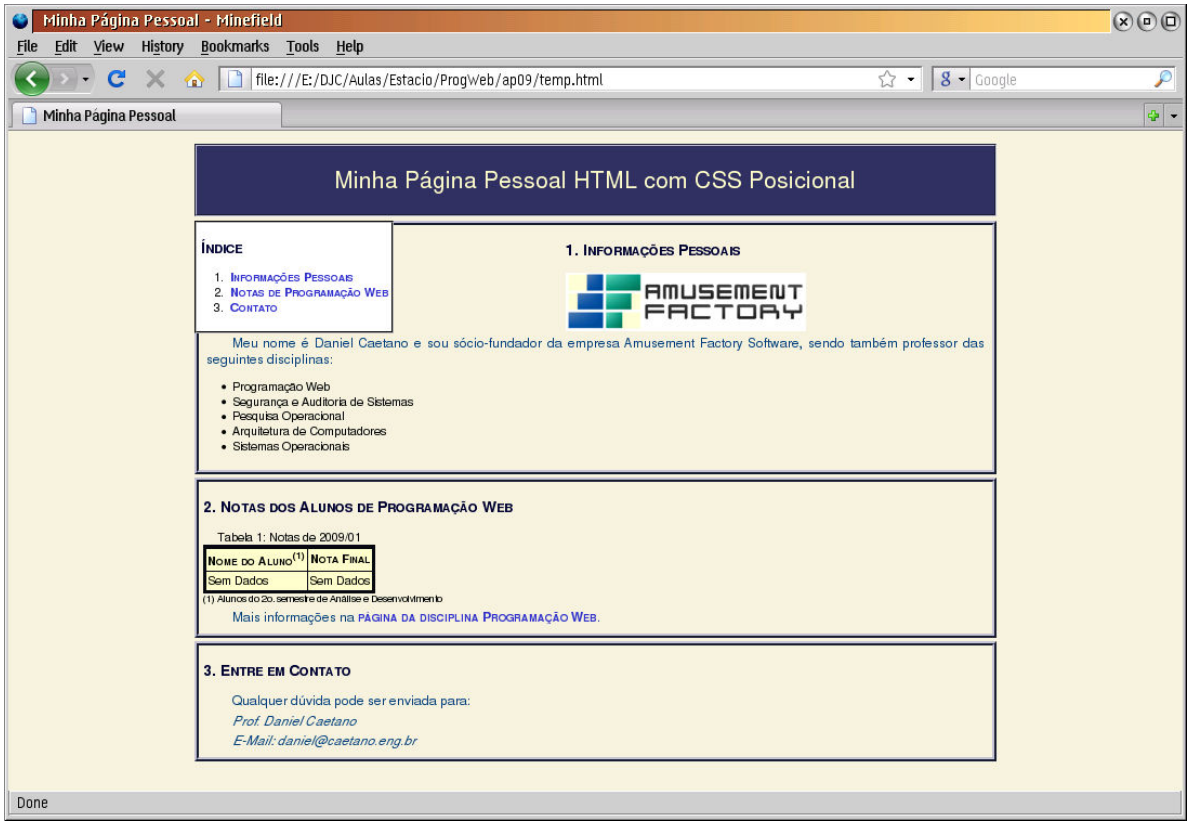
Observe o resultado:



Antes de continuarmos a "ajeitar" nossa página, lembram-se do segundo efeito colateral do "overflow: hidden"? Pois bem: retiremos o "overflow: hidden" das seções 1 a 3:

```
#secao1, #secao2, #secao3 {
  overflow: hidden;
  margin-left: 15%;
  margin-right: 15%;
  border-color: rgb(50,50,100);
  border-width: 6px;
  border-style: ridge;
  margin-top: 5px;
  padding: 5px;
  }
```

Recarregue a página e observe o resultado:



Note que a área da seção 1 ficou sobreposta à área do índice, mas o texto da seção 1 não ficou sobreposto (e nem sobrepôs) o conteúdo do índice. A esta altura você já deve ter identificado o "efeito colateral" do *overflow: hidden*, que é justamente o de impedir que regiões DIV diferentes se sobreponham horizontalmente.

Se não colocamos "overflow: hidden" nas seções 1, 2 e 3, a margem destas seções é contada a partir da borda da janela do navegador. Se colocamos "overflow: hidden" nestes elementos, a margem será contada a partir da borda dos elementos que estiverem aos seus lados; no caso, ao lado esquerdo da seção 1 existe o índice, então, ao colocar "overflow: hidden" na seção 1, fazemos com que a margem seja contada a partir do elemento do índice (e não da borda da janela).

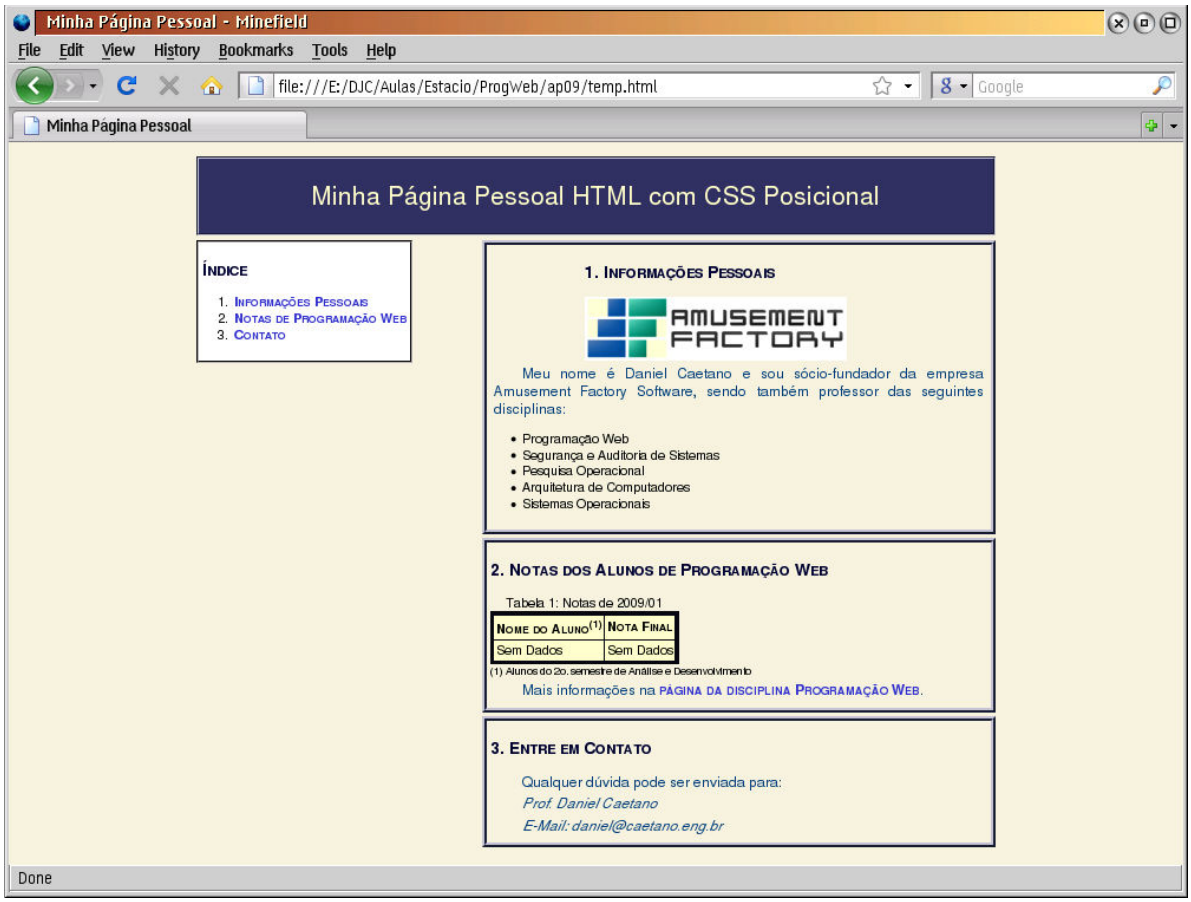
No caso das seções 2 e 3, como não há elementos à esquerda do bloco, a margem continua sendo contada a partir da borda da janela. É muito importante compreender estes "efeitos colaterais" de algumas tags para conseguir produzir exatamente o efeito que desejamos em nossas páginas.

Como queremos todas as regiões da direita alinhadas, vamos alinhá-las todas pela esquerda, a partir da margem e, por isso, continuaremos sem o "overflow: hidden" nas seções. Futuramente veremos como fazer este alinhamento com o uso do overflow: hidden.

Como sem o "overflow: hidden" as regiões estão sobrepostas porque a margens são iguais (a margem esquerda é de 15% da tela, tanto para o índice quanto para as seções 1 a 3), vamos modificar isso. Coloquemos uma margem esquerda de 40% para as seções 1, 2 e 3:

```
#secao1, #secao2, #secao3 {
  margin-left: 40%;
  margin-right: 15%;
  border-color: rgb(50,50,100);
  border-width: 6px;
  border-style: ridge;
  margin-top: 5px;
  padding: 5px;
  }
```

Observe o resultado:

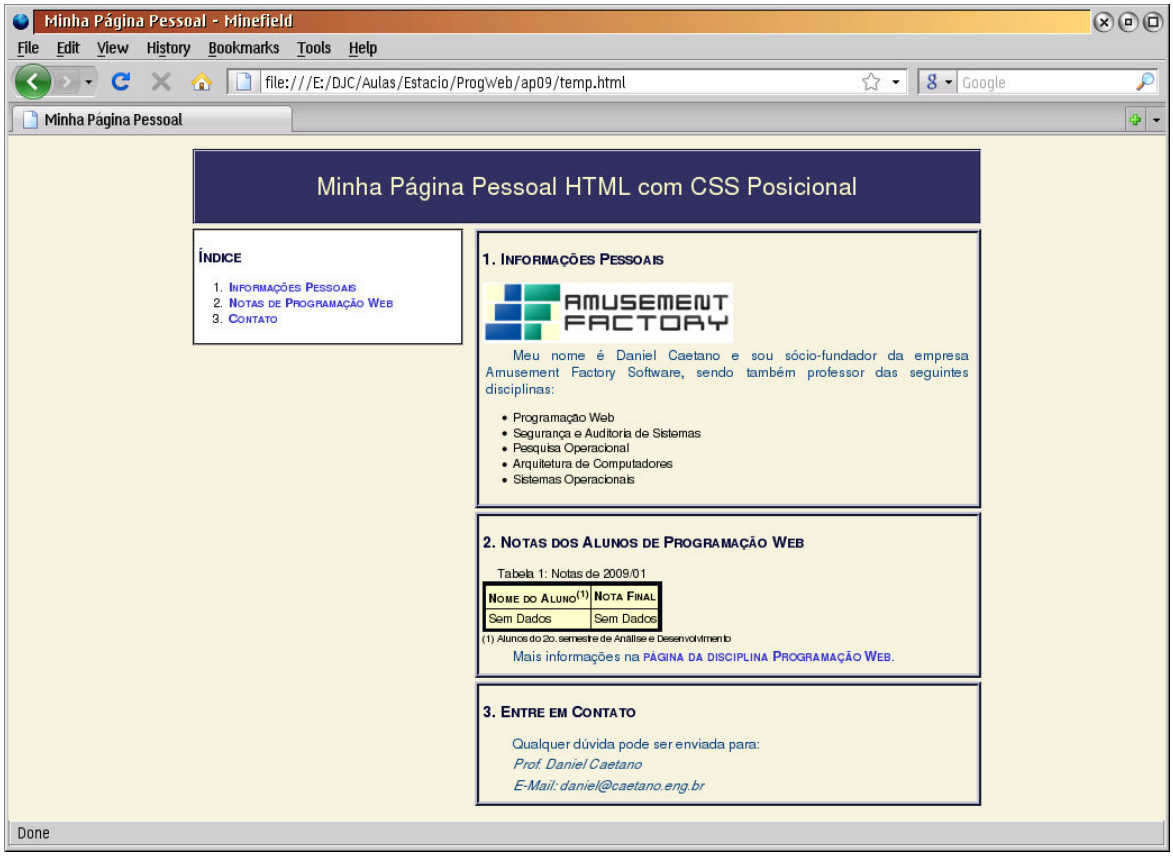


Melhorou bem! Observe que o texto da seção 1 parece meio "empurrado". Isso ocorre porque a seção índice tem uma margem esquerda definida. Vamos mdificar a seção índice de duas formas: a primeira, eliminando esta margem esquerda que está empurrando o nosso texto e a segunda, que será ampliar um pouquinho a largura do índice, usando para isso a propriedade width, com o valor 23%:

```
#indice {
  float: left;
  overflow: hidden;
  margin-left: 15%;
```

```
~~margin-right: 15%;~~
width: 23%;
background-color: rgb(255,255,255);
margin-top: 5px;
border-style: groove;
padding: 5px;
}
```

O resultado é este:



O nosso primeiro layout criado totalmente com CSS foi feito! Agora veremos alguns recursos bastante úteis do CSS, para facilitar a nossa vida. Futuramente voltaremos ao tópico de layouts HTML+CSS, explorando outros recursos destas poderosas tecnologias.

5. DEFININDO ESTILOS DENTRO DE BLOCOS

Conceitos Chave:

- Definindo estilos de tags dentro de seções
  - * tag_bloco tag { ... }
- Exemplos

Algumas vezes pode ser que exista a necessidade de definir uma propriedade diferenciada para um elemento dentro de outro. Por exemplo: foi definido como sendo, em geral, a aplicação de **bold** (negrito) no texto. Entretanto, dentro do parágrafo, gostaríamos que esta tag fosse definida como texto normal, mas com sublinhado. Isso é feito assim:

```
strong {
   font-weight: bold;
   }

p strong {
   font-weight: normal;
   text-decoration: underline;
}
```

Observe que esta definiçao segue um formato:

```
tag_bloco tag {
   ...
   }
```

Isso serve para definir estilos específicos dentro de uma parte da página, também:

```
#secao1 strong {
   font-weight: normal;
   text-decoration: underline;
}
```

6. MÍDIAS ALTERNATIVAS (OPCIONAL)

Conceitos Chave:
- Web é acessada por várias mídias
    * @midia { ... }

Algumas vezes o web master deseja criar alguns efeitos em sua página web que a inviabilizam para impressão (fundos escuros com letras claras, em geral, geram péssimas impressões em papel). Por esta razão, o CSS permite que sejam discriminadas as definições que serão ativas para cada tipo de mídia.

A forma de especificar isso é:

```
@tipo_de_midia {
  tag {
        atributo: propriedade;
  }
}
```

Em um mesmo arquivo CSS é possível definir todos os estilos para diferentes tipos de mídia. Os tipos de mídia possíveis são:

Tipo	Descrição
all	Usado para todos os tipos de mídia e dispositivos
aural	Usado para fala e sintetizadores de som
braille	Usado para dispositivos táteis em braile
embossed	Usado para impressoras braile
handheld	Usado para dispositivos pequenos ou de mão
print	Usado para impressoras
projection	Usado para apresentações projetadas, como slides
screen	Usado para as telas de computador
tty	Usado para mídias de fonte fixa (terminais, teletipos etc.)
tv	Usado para dispositivos tipo TV

7. BIBLIOGRAFIA

CASCADE Style Sheets, level 2 revision 1: CSS 2.1 Specification - W3C Working Draft 06 November 2006. Disponível em < http://www.w3.org/TR/CSS21/ >. Visitado em 21 de Dezembro de 2006.

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

RAMALHO, J.A. *HTML 4 Prático e Rápido*. Editora Berkeley, 1999.

BOENTE, A. *Programação Web Sem Mistérios*. Editora Brasport, 2006.

NIELSEN, J. *Projetando Websites*. Ed. Campus, 2000.

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:

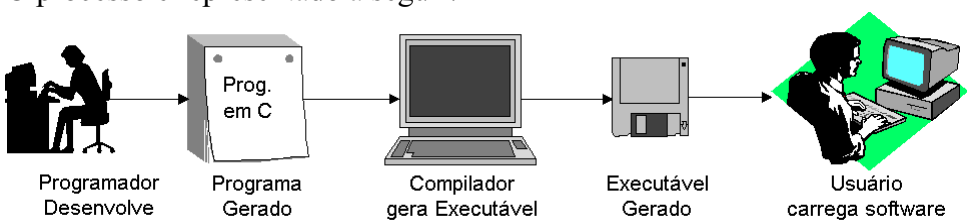


Figura 1: Processo de Criação à Execução com uso de Linguagens Compiladas

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:

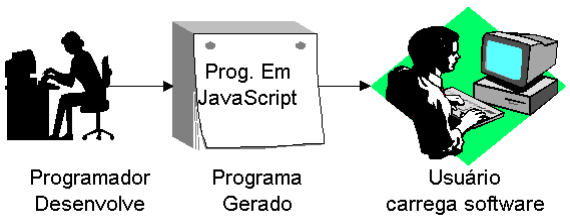


Figura 1: Processo de Criação à Execução com uso de Linguagens Compiladas

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.

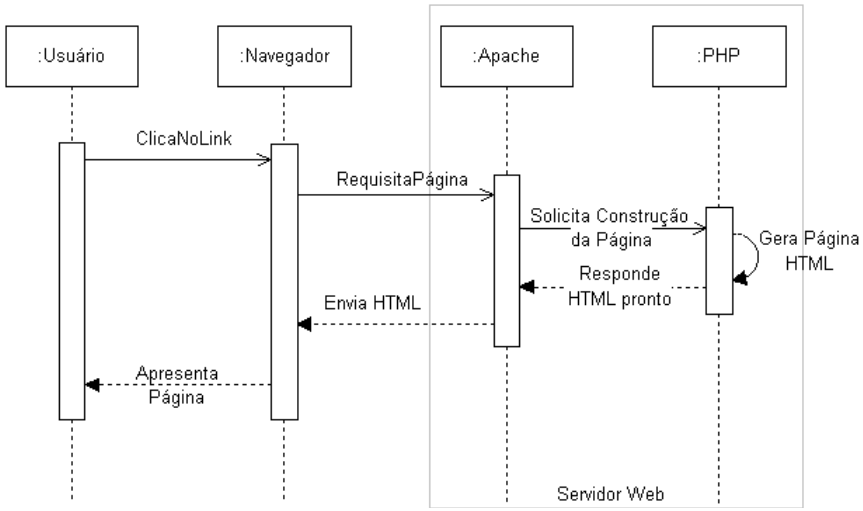


Figura 3: Linguagens processadas "no lado do servidor"

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.

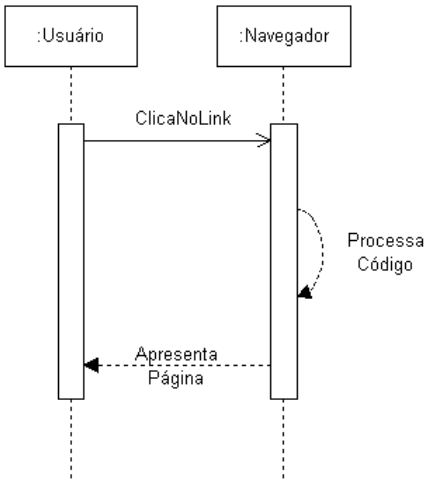


Figura 4: Linguagens processadas "no lado do cliente"

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.

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:

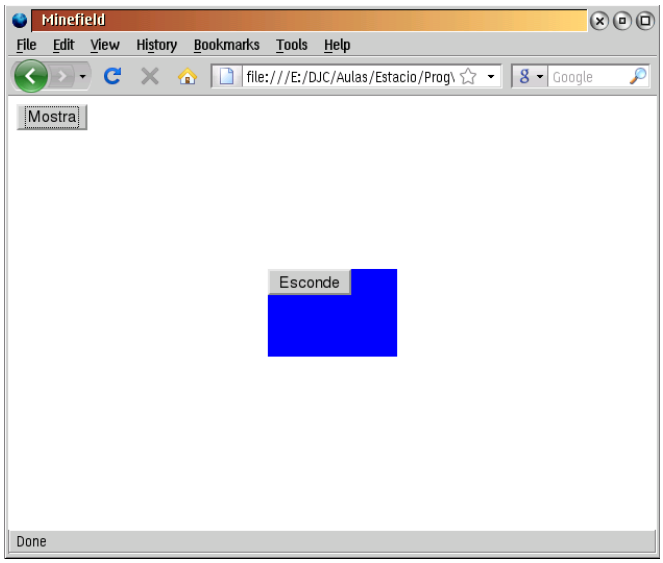


Figura 1: Observe a janela pop-up em azul

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:

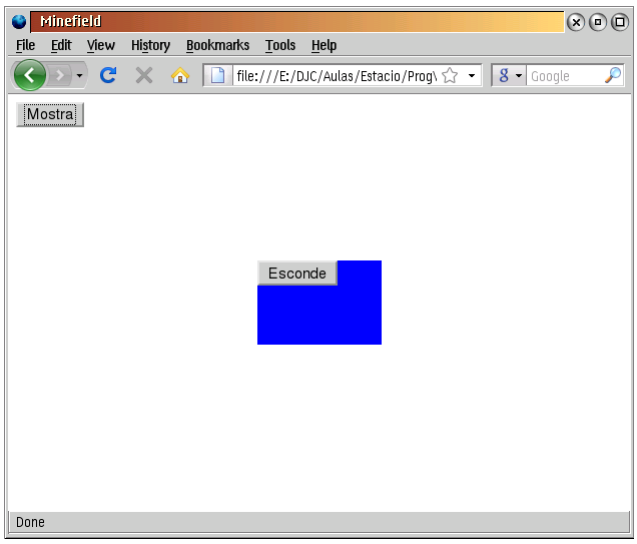


Figura 2: O resultado do código apresentado

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.

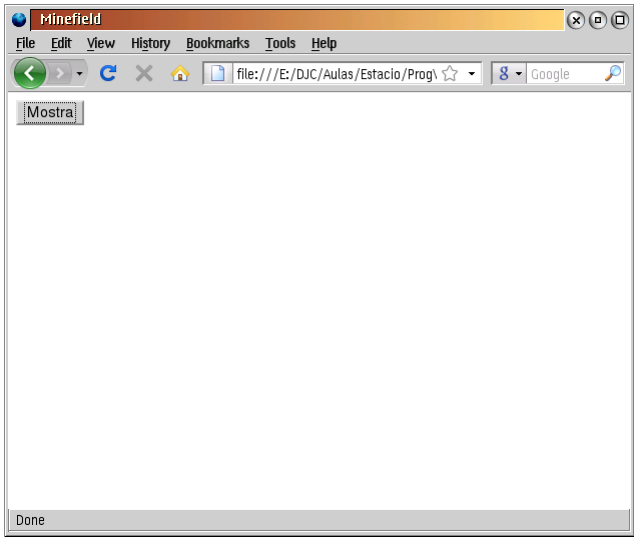


Figura 3: O popup sumiu!

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.

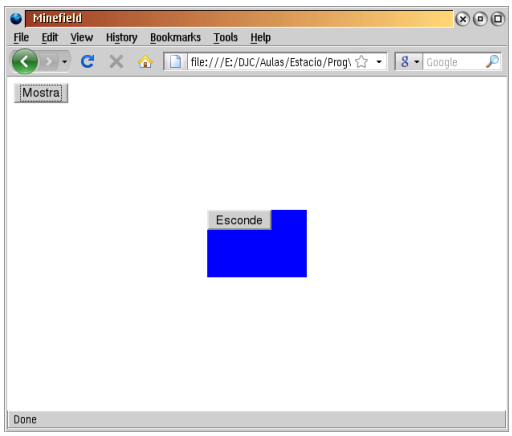


Figura 4: A janela Pop Up apareceu!

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.

Unidade 8: DHTML - O HTML Dinâmico

Prof. Daniel Caetano

Objetivo: Capacitar o aluno para entententer e manipular todos os elementos do XHTML e CSS com o uso de JavaScript.

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

INTRODUÇÃO

Nas aulas anteriores foram apresentadas diversas tecnologias, como o HTML, CSS e JavaScript. De maneira intuitiva, foram apresentadas algumas interações superficiais entre essas tecnologias para produzir efeitos interessantes.

Entretanto, a criação de sites mais elaborados e aplicações para a Web exige um conhecimento mais aprofundado e apurado destas tecnologias e da forma com que os elementos de uma página - HTML e CSS - são organizados na memória, estrutura essa que ficou conhecida pelo nome de Document Object Model (DOM, Modelo de Objeto de Documento).

Nesta aula serão apresentados os detalhes do DOM e como manipular a maior parte de seus elementos usando JavaScript.

1. O QUE É DHTML

O DHTML surgiu na época do HTML4 e, por essa razão, algumas coisas são, hoje, diferentes do que eram há 10 anos atrás. As tecnologias básicas do DHTML são: HTML, CSS e JavaScript.

Para que o DHTML fosse possível, na ocasião do HTML4 todos os navegadores foram programados para que cada elemento da página fosse um objeto: o parágrafo é um objeto, a imagem é um objeto, a lista é um objeto... Cada objeto pode ter outros objetos dentro de si e todos os objetos de uma página estão organizdos entro de um objeto mestre chamado **document**.

A organização destes objetos na memória e a maneira de interagir com eles através do JavaScript ficou conhecida pelo nome de Modelo de Objeto de Documento (DOM, Document Object Model, em inglês).

Os sites dinâmicos e as aplicações web nada mais são que páginas comuns em que scripts muito bem elaborados manipulam o conteúdo do HTML e as propriedades do CSS de maneira a obter efeitos muito próximos aos de uma aplicação tradicional.

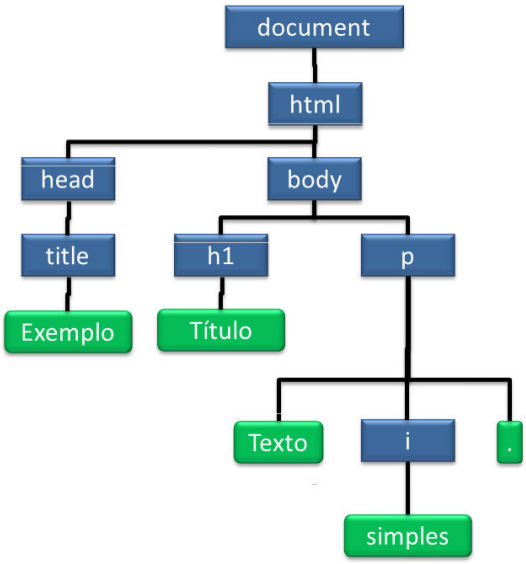
2. O QUE É O DOM?

Como já foi descrito, o Modelo de Objeto de Documento (DOM) é a essência do DHTML... mas o que é o DOM?

O DOM mapeia um documento HTML na forma de **uma árvore composta por nós**. Cada elemento marcado no HTML (parágrafo, lista, item de lista, imagem, div etc...) vira um nó no DOM. Cada nó pode ter outros nós dentro.

O nó raíz, que contém todas as nós do documento, é chamado criativamente de **document**. Observe um pequeno exemplo de uma estrutura em árvore construída pelo DOM para um pequeno HTML:

```
<html>
    <head>
        <title>Exemplo</title>
    </head>
    <body>
        <h1>Título</h1>
        <p>Texto <i>simples</i>.</p>
    </body>
</html>
```



Esses objetos podem ser **manipulados** pelo javascript. Por exemplo, se usarmos a seguinte linha JavaScript:

```
document.write("<p>Olá mundo!</p>");
```

Estaremos inserindo um ramo na árvore DOM. Onde exatamente? Aí depende... do ponto em que o código foi executado!

Considere os exemplos a seguir, que irão acrescentar esse texto **antes** e **depois** do <h1>, respectivamente.

Insere ANTES do h1:

```
<html>
	<head>
		<title>Exemplo</title>
	</head>
	<body>
		<script type="text/javascript"><!--
			document.write("<p>Olá mundo!</p>");
		--></script>
		<h1>Título</h1>
		<p>Texto <i>simples</i>.</p>
	</body>
</html>
```

Insere DEPOIS do h1:

```
<html>
	<head>
		<title>Exemplo</title>
	</head>
	<body>
		<h1>Título</h1>
		<script type="text/javascript"><!--
			document.write("<p>Olá mundo!</p>");
		--></script>
		<p>Texto <i>simples</i>.</p>
	</body>
</html>
```

O ponto onde o ramo é inserido depende do estágio da construção da árvore.

3. ACESSANDO O DOM

Manipular bem o DOM é a chava para uma aplicação interessante. Para manipular o DOM, primeiramente precisamos descobrir como **pegar uma referência para um elemento**. Por exemplo: se quiser mudar um texto de cor, preciso ter uma maneira de indicar para o navegador **qual** é esse elemento. Para pegar uma referência, o DOM nor fornece três funções principais, que serão vistas a seguir.

3.1. Pegando os elementos pelo ID

Observe o código abaixo:

```
<html>
	<head>
		<title>Exemplo</title>
	</head>
	<body>
		<h1 id="topo">Título</h1>
		<p>Texto <i>simples</i>.</p>
	</body>
</html>
```

Se quisermos escrever um JavaScript que modifica, por exemplo, a cor de fundo do <h1> cujo ID é "topo", podemos pegar uma referência para este elemento e modificá-lo assim:

```
var temp = document.getElementById("topo");	/* Pega a referência */
temp.style.backgroundColor = "red";
```

Como só pode existir um elemento com um dado **ID**, a função getElementById sempre retorna um único elemento. Observe que essa função é do objeto **document**.

3.2. Pegando os elementos pelo NOME

Observe o código abaixo:

```
<html>
	<head><title>Exemplo</title></head>
	<body>
		<form>
			<input type="text" name="cpf" />
		</form>
	</body>
</html>
```

Se quisermos escrever um JavaScript que modifica, por exemplo, a cor de fundo do campo <input> cujo nome é "cpf", podemos pegar uma referência para este elemento e modificá-lo assim:

```
var temp = document.getElementsByName("cpf");/* Pega as referências */
temp[0].style.backgroundColor = "red";
```

Observe que, como pode existir mais de um campo com o mesmo atributo **name**, a função getElementsByName sempre retorna **um vetor de elementos**, que podem ser acessados um a um pelo índice numérico.

Para saber quantos elementos foram retornados, basta acessar o **length**, assim:

```
var temp = document.getElementsByName("cpf");/* Pega as referências */
var num = temp.length;
window.alert(num);
```

Observe que a função **getElementsByName** é do objeto **document**.

3.3. Pegando os elementos pela TAG

Observe o código abaixo:

```
<html>
	<head>
		<title>Exemplo</title>
	</head>
	<body>
		<p>Texto <i>simples</i>.</p>
		<p>Outro texto.</p>
	</body>
</html>
```

Se quisermos escrever um JavaScript que modifica, por exemplo, a cor de fundo do segundo parágrafo, podemos pegar uma referência para este elemento e modificá-lo assim:

```
var temp = document.getElementsByTagName("p");	/* Pega as referências */
temp[1].style.backgroundColor = "red";
```

Observe que, como pode existir mais de um elemento marcado com a mesma **tag** a função getElementsByTagName sempre retorna **um vetor de elementos**, que podem ser acessados um a um pelo índice numérico.

Para saber quantos elementos foram retornados, basta acessar o **length**, assim:

```
var temp = document.getElementsByTagName("p");        /* Pega as referências */
var num = temp.length;
window.alert(num);
```

Observe que a função **getElementsByTagName** existe não apenas no **document**, mas também em qualquer elemento. Por essa razão, é possível fazer um código assim:

```
var temp = document.getElementsByTagName("p");        /* Pega as referências */
var elemento = temp[1].getElementsByTagname("i");
elemento.style.backgroundColor = "red";
```

Isso irá mudar apenas a cor de fundo da parte marcada com "i" dentro do segundo parágrafo encontrado.

3.4. Atributos que permite acessar outros elementos

Alguns atributos dos elementos permitem acessar outros elementos do XHTML:

parentNode	Referência para o nó pai
childNodes	Vetor dos nós filhos
firstChild	Referência para o primeiro nó filhote
lastChild	Referência para o último nó filhote
nodeValue	Valor do nó atual
nextSibling	Referência para o próximo elemento (inclui TextElements!)
previousSibling	Referência para o elemento anterior (inclui TextElements!)

4. MANIPULANDO O DOM

Podemos mudar os elementos do DOM de diversas formas. Uma delas é usando o atributo innerHTML, que permite modificar o conteúdo de um nó:

```
document.body.innerHTML = "<p>Olá!</p>"
```

É possível ainda usar funções para criar e remover elementos:

Nome	função	Quem tem
createElement("t")	Cria um elemento de tag	document
createTextNode("t")	Cria um elemento de texto	document
appendChild(e)	Acrecenta um nó filho ao elemento	todos os elementos
removeChild(e)	Remove um nó filho	todos os elementos
replaceChile(e1,e2)	Substitui um nó filho por outro	todos os elementos

5. MANIPULANDO O CSS PELO DOM

Podemos mudar os valores INLINE de CSS pelo DOM:

```
document.body.style.color = "blue";
```

Para mudar a classe de um elemento, basta usar:

```
document.body.className = "novaClasse";
```

Para acrescentar uma classe ao elemento:

```
document.body.className += " novaClasse";
```

A leitura do CSS, entretanto, não é tão óbvia. Isso não vai funcionar:

```
var cor = document.body.style.color;
```

Apenas alguns valores podem ser lidos diretamente, assim:

atributo	valor lido
offsetHeight	altura atual do elemento, em pixels
offsetWidth	largura atual do elemento, em pixels
offsetLeft	margem esquerda atual do elemento
offsetTop	margem superior atual do elemento

Outros valores precisam ser lidos de uma outra maneira, usando uma função:

```
function getStyle(oElm, strCssRule){
    var strValue = "";
    if(document.defaultView && document.defaultView.getComputedStyle) {
        strValue = document.defaultView.getComputedStyle(oElm, "").getPropertyValue(strCssRule);
        }
    else if(oElm.currentStyle) {
       strCssRule = strCssRule.replace(/\-(\w)/g, function (strMatch, p1) {
            return p1.toUpperCase();
            });
        strValue = oElm.currentStyle[strCssRule];
        }
    return strValue;
    }
```

Isso pode ser usado assim:

```
var el = document.getElementById("artigo");
var cor = getStyle(el,"background-color");
```

6. 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.

Aula 9: Fundamentos do JQuery

Fonte: Plano de Aula Oficial da Disciplina

Objetivo: Capacitar o aluno para o trabalho com o framework JQuery.

INTRODUÇÃO

JQuery é uma biblioteca (ou framework) "projetado para mudar sua maneira de escrever javascript"; É uma biblioteca JavaScript rápida e concisa que simplifica manipulação de eventos, animação e interações nos documentos HTML.

Foi lançada em Janeiro de 2006 por John Resig e é utilizada por mais de 31% dos 10.000 sites mais visitados, jQuery é atualmente a biblioteca JavaScript mais popular do mundo.

1. A BIBLIOTECA JQUERY

A biblioteca jQuery oferece uma camada de abstração de uso geral para scripts web, e por isso é útil em quase todas as situações que se utilizam scripts. Sua natureza extensível significa que plugins estão sendo constantemente desenvolvidos para acrescentar novas funcionalidades.

Caracteristicas

- Cross browser;
- Controle total sobre o DOM;
- Manipulação de eventos javascript;
- Manipulação de regras CSS;
- Aplicação de efeitos visuais;
- Uso de AJAX;

2. O QUE O JQUERY OFERECE?

2.1. Modificar a aparência de uma página web.

A CSS oferece um método poderoso de influenciar a forma como o documento é processado, mas fica prejudicado quando os navegadores web não dão todo o suporte para utilização de seus recursos. Com jQuery, os desenvolvedores não tem essa preocupação, pois fica a encargo do jQuery a equivalencia de padrões em todos os navegadores. Além disso, jQuery pode alterar as classes ou propriedades individuais de estilo aplicado para uma parte do documento, mesmo após a página ter sido carregada.

2.2. Alterar o conteúdo de um documento.

JQuery não se limita a simples mudanças na apresentação de um documento, jQuery pode modificar o conteúdo de um documento. O texto pode ser alterado, as imagens podem ser inseridos ou trocados, as listas podem ser reordenadas, ou toda a estrutura do HTML pode ser reescrita e ampliada, tudo isso com uma única API (Application Programming Interface).

2.3. Responder a interação do usuário.

Mesmo os mais elaborados e poderosos comportamentos não são úteis se não podemos controlar quando ocorrem. A biblioteca jQuery oferece uma maneira elegante para interceptar uma grande variedade de eventos, tais como clique em um link, sem necessidade de sobrecarregar o código HTML com manipuladores de eventos. Ao mesmo tempo, a API de manipulação de eventos remove as incoerências do navegador que normalmente afetam os desenvolvedores web.

2.4. Animações

Para implementar de forma eficaz esses comportamentos interativos, um designer também deve fornecer feedback visual para o usuário. A biblioteca jQuery facilita isso fornecendo uma série de efeitos.

2.5. Simplificar tarefas comuns de JavaScript

Além de todas essas características a biblioteca jQuery oferece aprimoramentos para construções básicas em JavaScript, como interação e manipulação de array.

3. USANDO O JQUERY

3.1. Download jQuery

O site da jQuery Offcial (http://jquery.com/) tem sempre a versão mais atualizada, além disso, possui recursos como códigos e notícias relacionadas com a biblioteca.

Para começar, precisamos de uma cópia da jQuery, que pode ser baixada na página inicial do site. Diversos versões da jQuery podem estar disponível a qualquer momento, a mais adequado para nós, como desenvolvedores de site vai ser a última versão descompactada da biblioteca. Nenhuma instalação é necessária. Para usar o jQuery, só precisamos colocá-lo em nosso site. Desde que o JavaScript é uma linguagem interpretada, não há compilação ou fase de construção para se preocupar. Sempre que precisamos de uma página para ter jQuery disponível, vamos simplesmente referir-se a localização do arquivo em nosso documento HTML.

A referência a biblioteca jQuery pode ser incluída em uma página web usando a seguinte marcação:

<script type="text/javascript" src="jquery.js"> </script>

Onde src e o endereço do arquivo **jquery.js.**

O jQuery também pode ser carregado de outras formas, como por meio do Google AJAX Libraries API , com a seguinte marcação:

<script type="text/javascript" src="http://www.google.com/jsapi"> </script>

ou

<script>google.load(jquery ", "1.4.2"); </script>

Ou, ainda, por HTTPS e HTTP:

<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.2/jquery.min.js"> </script>

A bilioteca pode ser carregada, ainda, usando o Microsoft hosts. O jQuery encontra-se em sua AJAX CDN (Content rede de distribuição), tornando mais fácil adicionar o suporte para a biblioteca jQuery. CDN serve bibliotecas JavaScript de um dos milhares servidores da Microsoft ao redor do mundo.

```
<script src="http://ajax.microsoft.com/ajax/jquery/jquery-1.4.2.min.js" type="text/javascript">
</script>
```

3.2. A Função de Busca $()

Não importa qual o tipo de seleção queremos utilizar em jQuery, sempre começamos com o cifrão e parênteses: $ (). Qualquer coisa que possa ser usada em um estilo também pode ser envolvida em aspas e colocados dentro dos parênteses, permitindo aplicar métodos jQuery para o conjunto de elementos combinados.

3.2.1. Método ready()

Como quase tudo o que fazemos quando estamos utilizando o jQuery lê ou manipula um modelo de objeto de documento (DOM), precisamos nos certificar que começamos adicionado eventos etc tão logo o DOM esteja pronto. Para fazer isso, nós registramos o evento ready (pronto) para o documento.

```
$(document).ready(init);
function init() {
    // faça alguma coisa quando o DOM estiver pronto
}
```

Esta notação é conhecida como **sintaxe formal** e também pode ser indicada de maneira "anônima":

```
$(document).ready( function() {
    // faça alguma coisa quando o DOM estiver pronto
});
```

O método jQuery ready() oferece duas vantagens sobre seu equivalente em JavaScript:

- O script está pronto para funcionar tão logo a árvore do documento tenha sido carregada. Não é necessário que tosdos os componentes da página, tais como imagens, folhas de estilo, animações e mídias em geral, tenham sido carregadas. Basta que a estrutura de marcação da págiuna esteja disponível e o script já poderá funcionar.

- Não há variações funcionais para o método e pelo fato de jQuery não admitir mistura de script com marcação, utiliza-se a sintaxe mostrada para servir de container aos scripts.

3.2.2. Seletores CSS

A biblioteca jQuery suporta quase todos os selecionadores incluídos nas especificações CSS 1 a 3, conforme descrito no site da World Wide Web Consortium: http://www.w3.org/Style/CSS/#. Este suporte permite que os desenvolvedores melhorem seus sites sem se preocupar com que os navegadores (especialmente Internet Explorer 6) podem compreender.

Para adicionar estilos em uma seleção jQuery podemos usar a seguinte sintaxe:

seletorjQuery.css('propriedade','valor')

3.2.3. Seleção de elementos

Existem três formas de selecionar elementos com jQuery:

- Nome da <tag>
- ID
- Class

Eles podem ser utilizados tanto sozinhas ou em combinação com outros selecionadores.

Sintaxe:

Seletor	Css	JQuery	Descrição
elemento	p	$('p')	Seleciona todos os paragrafos no documento
ID	#meuID	$('#meuID')	Seleciona o único elemento no documento lque possui o id "meuID"
class	.minhaClass	$('.minhaClass')	Seleciona todos os elementos no documento que tem a classe "minhaClass"

$('#id')

O seletor id acessa o elemento cujo valor do atributo id tenha sido especificado no argumento. Exemplo:

```
$(document).ready(function(){
    $('button').click(function(){
        $('#diferente').css('background','red');
        });
});
```

$('.class')

O seletor id acessa o elemento cujo valor da atributo class tenha sido especificado no argumento. Exemplo:

```
$(document).ready(function(){
    $('button').click(function(){
        $('.verde').css('background','green');
        });
});
```

$('elemento')

O seletor de elementos acessa todos os elementos especificados no argumento. Exemplo:

```
$(document).ready(function(){
    $('button').click(function(){
        $('div').css('background','yellow');
        });
});
```

4. SINTAXE AVANÇADA

4.1. Encadeamento

Denomina-se encadeamento a caracteristica de se poder encadear diversos métodos em uma declaração. Isso é possível porque se criou o jQuery de modo a que cada método em uma declaração retorne um objeto pronto para receber outro método, que, por sua vez, retornará outro objeto, e assim por diante, permitindo ao desenvolvedor construir uma cadeia de obejetos e métodos.

Exemplo:

```
$('div').children('p').fadeOut().addClass('xpto')
```

4.2. Grupamento de Seletores

O grupamento de seletores permite acessar um grupo de seletores. O argumento é a lista de seletores que se deseja acessar separados por virgula. Sintaxe:

```
$(seletor1, seletor2, seletor3,  seletorN)
```

Exemplo:

```
$(document).ready(function(){
    $('button').click(function(){
        $('p, #diferente, .verde, li').css('background','gray');
        });
});
```

4.3. Seletores compostos

Os seletores compostos são aqueles constituídos pela combinação de dois ou mais seletores simples. A combinação entre seletores simples para criar um seletor composto segue uma sintaxe própria e é feita com o emprego de três sinais de combinação.

Sinal de combinação	Exemplo	Nota
Espaço em branco	div p em	Obrigatório usar um ou mais espaços em branco entre os seletores
Sinal maior que ">"	div > p ou div>p	Espaço facultativo entre os seletores
Sinal de adição "+"	p + a ou p+a	Espaço facultativo entre os seletores

5. FILTROS

Denominam-se seletores filtros os seletores do tipo pseudosseletores que se destinam a filtrar uma condição particular de um seletor simples ou composto. Os filtros simplesmente filtram um seletor, para que ele atinja somente determinados elementos.

$('elemento:first')

Seleciona a primeiro elemento do documento

$('elemento:last')

Seleciona a último elemento do documento

$('elemento:even')

Seleciona os elementos com índice par do documento.

$('elemento:odd')

Seleciona os elementos com índice ímpar do documento.

$('elemento:eq(4)')

Seleciona o elemento com índice 4 do documento

$('elemento:contains(texto)')

Esse seletor é exclusivo da biblioteca jQuery. Acessa o texto definido no parâmetro. No exemplo seleciona todos os elementos que tem a string "texto" em seu conteúdo

$('elemento:hidden')

Seleciona todos os elementos ocultos do documento

$(':visible')

Seleciona tudo que está visível no documento

$('input:empty')

Seleciona todos os inputs vazios do documento

$('elemento:has(p)')

Seleciona todas as elementos que contenham parágrafos em seu conteúdo

5.1. Seletores de atributo

São os seletores que utilizam de atributos (href, id, class, etc.) para encontrar seus alvos.

$('elemento[id]')

Seleciona todos os elementos que tenham um atributo "id"

$('elemento[id = "valor"]')

Seleciona todos os elementos com id = "valor"

$('elemento[id != "valor"]')

Seleciona todos os elementos com id diferente de "valor". Você também pode acumular atributos, assim:

$('elementos[id = "valor"][href = "link"]')

5.2. Seletores para formulários

São seletores criados especialmente para uso em formulários.

$(':input')

Seleciona todos os inputs input, textarea, button e select do documento

$(':text')

Seleciona todos os inputs com type = 'text' do documento

$(':password')

Seleciona todos os inputs type = 'password' do documento

$(':checkbox')

Seleciona todos os checkboxes; o mesmo serve para inputs do tipo radio, button, submit, reset, image e file.

$(':enabled')

Seleciona todos os elementos do formulário que estejam abilitados

$(':disabled')

Seleciona todos os elementos do formulário que estejam desabilitados

$(':checked')

Seleciona os elementos selecionados em campos do tipo radio e checkbox

$(':selected')

Seleciona os elementos selecionados nos selects do formulário

6. BIBLIOGRAFIA

Plano de Aula Oficial da Disciplina "CCT0081 - Programação para Internet Rica", Centro Universitário Estácio-Radial.

Aula 10: JQuery UI

Fonte: Plano de Aula Oficial da Disciplina

Objetivo: Capacitar o aluno para o trabalho com a biblioteca Jquery UI.

INTRODUÇÃO

Desde 1995, quando Brendan Eich criou a linguagem JavaScript, seu intuito já era proporcionar interatividade as páginas HTML. Para tornar mais fácil o desenvolvimento web, surgiram diversos frameworks e bibliotecas JavaScript. Atualmente, os desenvolvedores podem escolher entre mais de 40 delas, de acordo com os recursos e suas finalidades.

A JQuery UI é uma biblioteca complementar ao framework JQuery e que tem o objetivo de permitir uma maior facilidade no desenvolvimento de aplicativos e sites profissionais fornecendo um conjunto de elementos e funcionalidades prontos para serem usados.

1. DIFERENCIANDO JQUERY DA JQUERY UI

Como fui dito na aula passada, o JQuery é um framework para facilitar o acesso a recursos já disponíveis, uma tarefa que normalmente é especificada pelo nome **framework**.

O JQuery UI, por outro lado, é uma biblioteca de recursos úteis prontos para serem usados e que é baseada no framework JQuery. O objetivo da JQuery UI é "componentizar" interações de uma forma acessível e em camadas, de modo que o conteúdo tenha maior portabilidade.

2. DOWLOAD E ORGANIZAÇÃO

Pode-se baixar um pacote de download rápido através do botão "quick download" do site, http://jqueryui.com/ . Por ouro lado, a biblioteca é relativamente grande (adiciona cerca de 50KB ao tamanho do JQuery), o que pode levar a alguns preferir selecionar alguns pacotes específicos na parte de download do site. Através desta opção é possível customizar de acordo com o objetivo que o desenvolvedor deseja alcançar, além de poder escolher um tema mais apropriado as cores de sua aplicação.

A vantagem do download personalizado é a redução de tempo do carregamento dos scripts da biblioteca, mas requer uma boa escolha dos componentes. Para entender a organização, é preciso apresentar alguma nomenclatura.

Os componentes da JQuery UI são divididos em quatro módulos: core, interações, widgets e efeitos.

Core - elementos básicos do JQuery UI, e são meio que obrigatórios
Interações - adicionam comportamentos básicos a qualquer elemento, como arrastar, expandir, selecionar, etc.
Widgets - oferece os principais recursos de navegação utilizados, como abas, caixas de diálogo, menu expansivo...
Efeitos - Elementos de animação, trás uma série de funcionalidades para tornar websites mais profissionais, interativos e completos.

3. O QUE A JQUERY UI OFERECE?

3.1. Componentes de Interação.

*** Draggable** http://jqueryui.com/demos/draggable/
Permite funcionalidades arrastáveis em qualquer elemento DOM (Document Object Model) dentro da janela de exibição.

*** Resizable** http://jqueryui.com/demos/resizable/
Permite que qualquer elemento DOM seja redimensionado, arrastando sua borda, aumentando assim para a largura e a altura desejada de um box, por exemplo. Pode ser controlado por scripts que limitam a largura e altura máxima permitida, que aplicam efeitos entre outras funcionalidades.

*** Selectable** http://jqueryui.com/demos/selectable/
Permite que um elemento ou um grupo de elementos sejam selecionáveis ao clique do mouse, com a possibilidade da utilização da tecla CTRL para fazer múltiplas seleções.

*** Sortable** http://jqueryui.com/demos/sortable/
Permite que os elementos sejam arrastados para um novo local em uma lista de elementos, enquanto os outros se ajustam sobre a nova organização de elementos, por meio de um seletor connectwith.

*** Doppable** http://jqueryui.com/demos/doppable/
Permite que qualquer elemento DOM seja alvo para os elementos arrastáveis. É possível controlar se o alvo aceita ou não o elemento, alterar a aparência do alvo quando o elemento é lançado sobre ele, ou até criar um gerenciador de fotografias, com funcionalidades

como: apagar uma imagem da galeria arrastando a para a lixeira ou clicando no ícone da lixeira, retorná-la da lixeira para a galeria ou visualizá-la em tamanho maior pelo ícone zoom.

3.2. Componentes para Criação de Widgets.

*** Datepicker** http://jqueryui.com/demos/datepicker/

Campo de formulário que traz um calendário interativo. Basta escolher a data e clicar para que seja preenchido. Pode-se adicionar funcionalidades como a escolha do formato de data, de acordo com a língua utilizada, menus para facilitar a escolha do mês e do ano, entre outras.

*** Accordion** http://jqueryui.com/demos/accordion/

Permite a construção de cabeçalhos clicáveis que expande em seções lógicas a área de conteúdo, como guias. Mais conhecido como efeito sanfona.

*** Dialog** http://jqueryui.com/demos/dialog/

Permite a criação de uma Dialog, que é uma janela flutuante que contém uma barra de título e uma área de conteúdo. A janela de diálogo pode ser movida, redimensionada e fechada através do ícone 'x'. É possível implementá-la para que se o comprimento do conteúdo exceder a altura máxima, uma barra de rolagem aparecerá automaticamente. O formato Modal impede que o usuário interaja com o restante da página até que o diálogo seja fechado, podendo adicionar uma camada semitransparente que escurece o conteúdo da página por trás do diálogo.

*** Progressbar** http://jqueryui.com/demos/progressbar/

A barra de progresso da jQuery UI mostra a porcentagem de processamento da aplicação. Ela pode ser animada e redimencionável através de comandos CSS para se ajustar ao tamanho da tela.

*** Slider** http://jqueryui.com/demos/slider/

O pluguin Slider pode ser usado para que o usuário, de forma dinâmica, mova alças com o mouse para definirem o resultado desejado.

*** Tabs** http://jqueryui.com/demos/tabs/

As tabelas são guias, geralmente utilizadas para dividir o conteúdo em seções. O usuário pode alternar entre as abas através de clique (onClick) ou ao passar do mouse (onHover). Há também a opção Sortable para que o usuário organiza as abas de acordo com sua preferência, apenas as arrastando para a posição desejada.

3.3. Componentes de Efeitos

*** AddClass** http://jqueryui.com/demos/addclass/
Adiciona a classe específica para cada conjunto de elementos com transições opcionais entre os estados.

*** RemoveClass** http://jqueryui.com/demos/removeclass/
Remove todas as classes ou uma classe específica de um elemento, com transições opcionais entre os estados.

*** Hide** http://jqueryui.com/demos/hide/
Diversos efeitos para esconder um elemento.

*** Animate** http://jqueryui.com/demos/animate/
Esta função permite animações com cores. É muito usada pelo recurso de transição de classe, podendo-se alterar as seguintes propriedades: backgroundColor, borderBottomColor, borderLeftColor, borderRightColor, borderTopColor, color, outlineColor.

*** Effect** http://jqueryui.com/demos/effect/
Permite que diversos tipos de efeitos sejam adicionados a um elemento, sem a lógica de show e hide.

*** Show** http://jqueryui.com/demos/show/
Diversos efeitos para mostrar um elemento

*** SwiteClass** http://jqueryui.com/demos/switeclass/
Muda de uma classe definida para outra classe com a opção de utilizar uma transição.

*** Toggle** http://jqueryui.com/demos/toggle/
Método de alternância de um elemento para outro que aceita efeitos avançados da JQuery UI.

*** ToggleClass** http://jqueryui.com/demos/toggleclass/
Adiciona a classe se ela não estiver presente e a remove se ela estiver presente com a opção de utilizar uma transição.

4. BIBLIOGRAFIA

Plano de Aula Oficial da Disciplina "CCT0081 - Programação para Internet Rica", Centro Universitário Estácio-Radial.

Guia de Referência CSS Volume 1: Parâmetros Básicos

Prof. Daniel Caetano

Objetivo: Apresentar os parâmetros básicos que podem ser modificados com a tecnologia CSS.

Bibliografia: W3, 2009; CASCADE, 2006; RAMALHO, 1999.

INTRODUÇÃO

O Objetivo deste texto é servir de apoio na criação de sites web, indicando todas as características básicas de um documento HTML que podem ser modificadas com o uso da tecnologia CSS.

Não se pretende com este documento explicar o uso da tecnologia CSS, visto que isso foi feito em aula específica, mas sim apresentar uma grande variedade de propriedades cuja discussão não é possível em aula, devido a restrições de tempo.

1. ESTILOS PARA PLANO DE FUNDO (Background)

O plano de fundo é uma propriedade clássica, que serve para modificar a cor ou imagem do fundo de qualquer elemento gráfico do HTML. Isso inclui o fundo da página, as barras HR, as tabelas, dentre outros. As propriedades principais de mudança de plano de fundo são:

Nome	**Função**
background-color	Muda a cor do fundo do elemento rgb(vermelho, verde, azul) / #RRGGBB transparent
background-image	Muda a imagem de fundo do elemento url("nome.jpg") / none
background-repeat	Indica se a imagem de fundo deve se repetir ou não repeat / repeat-x repeat-y / no-repeat
background-attachment	Indica se a imagem de fundo é fixa ou não scroll / fixed
background-position	Indica posição inicial da imagem de fundo xpos ypos / x% y% top/center/bottom left/center/right

Exemplo: faz o fundo da página ter a cor azul, e apresenta uma imagem que serve de fundo para uma barra no topo da página, repetida horizontalmente:

```
body {
        background-color: rgb(0,0,200);
        background-image: url("bar.jpg");
        background-position: top left;
        background-repeat: repeat-x;
        }
```

2. ESTILOS PARA TEXTO (text)

Todo elemento de texto usado no HTML pode ser modificado visualmente de diversas maneiras, sendo as propriedades de texto justamente as mais comumente modificadas. As propriedades de mudança de texto são:

Nome	Função
color	Muda a cor do texto rgb(vermelho, verde, azul) / #RRGGBB
direction	Muda a direção do texto (esq->dir / dir->esq) ltr / rtl
line-height	Muda a distância entre linhas normal / *número* *comprimento* / *%*
letter-spacing	Aumenta ou diminui o espaço entre caracteres normal / *comprimento*
text-align	Alinhamento de texto no elemento left / right center / justify
text-decoration	Adiciona uma decoração no texto none / underline overline / line-through / blink
text-indent	Adiciona indentação à primeira linha de texto em um elemento *comprimento* / *%*
text-shadow	Adiciona sombra no texto none / *cor* / *comprimento*
text-transform	Controla as letras em um elemento none / capitalize uppercase / lowercase
unicode-bidi	Controle de display bidirecional do Unicode normal / embed bidi-override
white-space	Define como os espaços serão processados no elemento normal / pre nowrap
word-spacing	Aumenta ou diminui o espaçamento entre palavras normal / *comprimento*

Exemplo: faz os parágrafos possuírem texto em amarelo, justificado, colocando as primeiras letras de cada palavra em maiúsculas:

```
p {
        color: rgb(255,255,0);
        text-align: justify;
        text-transform: capitalize;
        }
```

3. ESTILOS PARA A FONTE (font)

O tipo de letra usada nos textos também pode ser modificada de uma grande maneira de formas. As propriedades principais de mudança de fonte são:

Nome	Função
font-family	Muda o tipo de caractere *nome-da-fonte* / *fonte-genérica*
font-size	Muda o tamanho da fonte xx-small / x-small small / medium large / x-large xx-large / smaller larger / *comprimento* *%*
font-size-adjust	Seleciona um tamanho para que a fonte alternativa tenha o mesmo tamanho da fonte principal none / *number*
font-stretch	Comprime ou expande a fonte atual normal / wider narrower / ultra-condensed extra-condensed / condensed semi-condensed / semi-expanded expanded / extra-expanded ultra-expanded /
font-style	Muda o estilo da fonte normal / italic oblique
font-variant	Mostrar um texto em letras pequenas ou normais normal / small-caps
font-weitght	Muda o "peso" de uma fonte normal / bold bolder / lighter 100 *a* 900

Exemplo: modifica a fonte de título para o tipo Verdana, Arial ou o genérico "sans-serif", com tamanho 28 pixels, em itálico e bold, e com todas as letras em maiúsculas:

```
h1 {
        font-family: verdana, arial, sans-serif;
        font-size: 28px;
        font-style: italic;
        font-weight: bold;
        font-variant: small-caps;
        }
```

4. ESTILOS DE BORDAS (border)

Todo elemento HTML pode ter uma borda, mesmo que ela não seja definida por padrão. A borda é como uma moldura que fica ao redor do elemento e é comum que o designer queira modificar suas propriedades. As propriedades de mudança de borda são:

Nome	Função
border-color	Muda cor das bordas
	rgb (vermelho, verde, azul) / *#RRGGBB*
border-style	Muda o estilo das bordas
	none / hidden
	dotted / dashed
	solid / double
	groove / ridge
	inset / outset
border-width	Muda a espessura da borda
	thin / medium
	thick / *comprimento*
border-bottom-___	Muda a propriedade só para a borda inferior
border-left-___	Muda a propriedade só para a borda esquerda
border-right-___	Muda a propriedade só para a borda direita
border-top-___	Muda a propriedade só para a borda superior

Exemplo 1: liga toda a borda de um parágrafo, em cinza escuro, com estilo "afundado", de espessura fina:

```
p {
        border-color: rgb(60,60,60);
        border-style: inset;
        border-width: thin;
        }
```

Exemplo 2: apresenta as bordas direita e inferior de um parágrafo, na cor cinza escura, em formato arredondado e espessura grossa na inferior e média na direita:

```
p {
        border-right-color: rgb(60,60,60);
        border-right-style: ridge;
        border-right-width: medium;
        border-bottom-color: rgb(60,60,60);
        border-bottom-style: ridge;
        border-bottom-width: thick;
        }
```

5. ESTILOS DE LINHA DE CONTORNO (outline)

Algumas vezes é desejável modificar as propriedades das linhas de contorno de algum elemento. A linha de contorno fica no exterior da borda do elemento. Isso é feito com estas propriedades, que funcionam de forma muito similar às propriedades de borda, com a diferença de não ser possível especificar separadamente cada uma das faces da linha de contorno. As propriedades principais de linha de contorno são:

Nome	Função
outline-color	Muda cor do contorno *rgb(vermelho, verde, azul)* / *#RRGGBB* invert
outline-style	Muda o estilo do contorno none / dotted dashed / solid double / groove ridge / inset outset
outline-width	Muda a espessura do contorno thin / medium thick / *comprimento*

Exemplo: coloca uma linha de contorno vermelha sólida e fina em um parágrafo

```
p {
        outline-color: rgb(255,0,0);
        outline-style: solid;
        outline-width: thin;
        }
```

6. ESTILOS DE MARGEM (margin)

A margem é a distância mínima que deve existir entre o contorno (outline) e os outros elementos da página (externos). É bastante comum o desejo de mudar as margens de um elemento para afastar outros que estejam ao redor dele. As propriedades principais de margem são:

Nome	Função
margin	Muda a propriedade de todas as margens auto / *comprimento* / *%*
margin-bottom	Muda apenas a propriedade da margem inferior
margin-left	Muda apenas a propriedade da margem da esquerda
margin-right	Muda apenas a propriedade da margem da direita
margin-top	Muda apenas a propriedade da margem do topo

Exemplo: define uma margem de 0 pixels (nenhuma) ao redor do parágrafo, para eliminar o espaçamento vertical entre os parágrafos:

```
p {
	margin: 0px;
	}
```

7. ESTILOS DO ESPAÇO INTERNO (padding)

Algumas vezes se deseja modificar o espaçamento entre a borda de um elemento e seu conteúdo interno. Isso é feito modificando o "padding", cujas propriedades são:

Nome	Função
padding	Muda a propriedade de espaçamento em toda a volta *comprimento* / *%*
padding-bottom	Muda apenas a propriedade da espaçamento inferior
padding-left	Muda apenas a propriedade da espaçamento da esquerda
padding-right	Muda apenas a propriedade da espaçamento da direita
padding-top	Muda apenas a propriedade da espaçamento do topo

Exemplo: define uma borda vermelha sólida e fina para o H1 e coloca um espaçamento interno de 15 pixels entre ela e o texto H1.

```
H1 {
	border-color: rgb(255,0,0);
	border-style: solid;
	border-width: thin;
	padding: 15px;
	}
```

7. ESTILOS DE MARCADORES DE LISTA (List)

As listas padrão do HTML podem ser muito feias. Para modificar o estilo das listas, existem os seguintes parâmetros:

Nome	Função
list-image	Muda a imagem do marcador da lista
	url("image.jpg") / none
list-style-position	Indica a posição do marcador na lista
	inside / outside
list-style-type	Indica o tipo de marcador de lista
	none / disc
	circle / square
	decimal / decimal-leading-zero
	lower-roman / upper-roman
	lower-alpha / upper-alpha
	lower-greek / lower-latin
	upper-latin / hebrew
	armenian / georgian
	cjk-ideographic / hiragana
	katakana / hiragana-iroha
	katakana-iroha
marker-offset	Modifica o distanciamento entre marcador e texto
	auto / *comprimento*

Exemplo: Muda o elemento de lista para que a imagem do marcador de lista como sendo a imagem *seta.gif*:

```
li {
        list-image: url("seta.gif");
        }
```

Exemplo 2: Muda o elemento de lista para que fique sem marcador de lista:

```
li {
        list-style-type: none;
        }
```

8. ESTILOS DOS ELEMENTOS DAS TABELAS (Table)

Algumas propriedades de tabelas podem ser mudadas também, com os seguintes parâmetros:

Nome	Função
border-colapse	Indica se as bordas de tabelas são sobrepostas ou não (se collapse, mostra apenas uma linha da tabela) collapse / separate
border-spacing	Distância que separa células (apenas para "separate") *comprimento comprimento*
caption-side	Posição da legenda da tabela top / bottom left / right
empty-cells	Indica se as células vazias devem ser mostradas show / hide
table-layout	Muda o algoritmo para mostrar as células, linhas e colunas auto / fixed

Exemplo: definir para que uma tabela tenha legenda na parte superior, exibindo as células vazias:

```
table {
        caption-side: top;
        empty-cells: show;
        }
```

9. BIBLIOGRAFIA

CASCADE Style Sheets, level 2 revision 1: CSS 2.1 Specification - W3C Working Draft 06 November 2006. Disponível em < http://www.w3.org/TR/CSS21/ >. Visitado em 21 de Dezembro de 2006.

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

RAMALHO, J.A. *HTML 4 Prático e Rápido*. Editora Berkeley, 1999.

Guia de Referência CSS Volume 2:
Parâmetros de Posicionamento

Prof. Daniel Caetano

Objetivo: Apresentar os parâmetros de posicionamento que podem ser modificados com a tecnologia CSS.

Bibliografia: W3, 2009; CASCADE, 2006; RAMALHO, 1999; NIELSEN, 2000.

INTRODUÇÃO

O Objetivo deste texto é servir de apoio na criação de sites web, indicando todas as características de posicionamento que podem ser modificadas com o apoio da tecnologia CSS.

Não se pretende com este documento explicar o uso da tecnologia CSS, visto que isso foi feito em aula específica, mas sim apresentar uma grande variedade de propriedades cuja discussão não é possível em aula, devido a restrições de tempo.

1. ATRIBUTOS DE DIMENSÃO (dimensions)

É muito frequente o desejo de mudar as dimensões e espaçamentos de alguns elementos. Para isso são usadas as propriedades de dimensão:

Nome	**Função**
height	Muda a altura de um elemento
	auto / *comprimento* / *%*
line-height	Muda a distância entre linhas
	normal / *comprimento*
	número / *%*
max-height	Define a altura máxima de um elemento
	none / *comprimento* / *%*
max-width	Define a largura máxima de um elemento
	none / *comprimento* / *%*
min-height	Define a altura mínima de um elemento
	none / *comprimento* / *%*
min-width	Define a largura mínima de um elemento
	none / *comprimento* / *%*
width	Muda a largura de um elemento
	auto / *comprimento* / *%*

2. ATRIBUTOS DE CLASSIFICAÇÃO (classification)

As propriedades de classificação são aquelas que permitem indicar como algum elemento será apresentado, se sobrepõe outro etc. Um layout perfeito depende de um bom domínio sobre estas propriedades:

Nome	Função
clear	Define em que lados não podem existir "floats"
	left / right
	both / none
cursor	Especifica o tipo de cursor a ser apresentado (mouse)
	url / auto
	crosshair / default
	pointer / move
	e-resize / ne-resize
	nw-resize / n-resize
	se-resize / sw-resize
	s-resize / w-resize
	text / wait
	help
display	Define como e se um elemento é apresentado
	none / inline
	block / list-item
	run-in / compact
	marker / table
	inline-table / table-row-group
	table-header-group / table-footer-group
	table-row / table-column-group
	table-column / table-cell
	table-caption
float	Define onde uma imagem ou texto irá aparecer
	left / right
	none
position	Define o tipo de posicionamento de elementos
	static / relative
	absolute / fixed
visibility	Define se um elemento deve ser visível ou não
	visible / hidden
	colapse

3. ATRIBUTOS DE POSICIONAMENTO (positioning)

As propriedades de posicionamento são aquelas que permitem indicar onde algum elemento será apresentado. Um layout bom depende de domínio sobre estas propriedades, lembrando que seu comportamento pode mudar de acordo com o tipo de posicionamento (absoluto, estático, relativo e fixo):

Nome	**Função**
bottom	Define a distância entre a parte inferior de um elemento e seu "elemento-pai" auto / *%* *comprimento*
clip	Define o formato de um elemento *shape* / auto
left	Define a distância entre a parte esquerda de um elemento e seu "elemento-pai" auto / *%* *comprimento*
overflow	Define o que ocorre se elemento não couber em sua área visible / hidden scroll / auto
position	Define o tipo de posicionamento de elementos static / relative absolute / fixed
right	Define a distância entre a parte direita de um elemento e seu "elemento-pai" auto / *%* *comprimento*
top	Define a distância entre a parte superior de um elemento e seu "elemento-pai" auto / *%* *comprimento*
vertical-align	Define o alinhamento vertical de um elemento baseline / sub super / top text-top / middle bottom / text-bottom *comprimento* / *%*
z-index	Define a ordem Z (saindo da tela) de um elemento auto / *número*

4. BIBLIOGRAFIA

CASCADE Style Sheets, level 2 revision 1: CSS 2.1 Specification - W3C Working Draft 06 November 2006. Disponível em < http://www.w3.org/TR/CSS21/ >. Visitado em 21 de Dezembro de 2006.

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

Guia de Referência JavaScript: Interagindo com o Navegador

Prof. Daniel Caetano

Objetivo: Apresentar os elementos mais comuns pelos quais o JavaScript pode interagir com o navegador.

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

INTRODUÇÃO

O Objetivo deste texto é servir de apoio na criação de sites web, indicando os elementos comuns de um navegador que podem ser modificados com o uso do JavaScript.

Não se pretende com este documento explicar o uso da tecnologia JavaScript, visto que isso foi feito em aula específica, mas sim apresentar uma grande variedade de elementos cuja discussão não é possível em aula, devido a restrições de tempo.

1. 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.

BODY e FRAMESET

onload	Quando um documento **inicia** seu carregamento
onunload	Quando um documento **inicia** seu "descarregamento"

Elementos de FORM

onblur	Quando o elemento perde o foco
onchange	Quando o conteúdo de um elemento for alterado
onfocus	Quando o elemento receber foco
onreset	Quando o form for resetado
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
ondblclick	Quando o elemento for duplamente clicado
onmousedown	Quando o botão do mouse for apertado sobre o elemento
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

2. ESTILOS VISUAIS QUE PODEM SER ALTERADOS

As propriedades visuais dos elementos podem ser acessadas com nomes específicos, permitindo a alteração de estilos em tempo de execução. A lista de estilos mais comuns está apresentada a seguir.

Plano de Fundo

backgroundAttachment	Indica se a imagem de fundo é fixa ou rola
backgroundColor	Muda cor de fundo de um elemento.
backgroundImage	Muda a imagem de fundo de um elemento
backgroundPosition	Muda a posição de uma imagem de fundo
backgroundPositionX	Muda a posição "x" da imagem de fundo
backgroundPositionY	Muda a posição "y" da imagem de fundo
backgroundRepeat	Define se e como a imagem de fundo será repetida

Textos

color	Muda a cor do texto
fontFamily	Muda o tipo de fonte
fontSize	Muda o tamanho da fonte
fontSizeAdjust	Muda / ajusta o tamanho de um texto
fontStretch	Estica ou aperta uma fonte
fontStyle	Muda o estilo da fonte
fontVariant	Texto em "mini-maiúsculas"
fontWeight	Muda a espessura da fonte
letterSpacing	Muda o espaçamento entre letras
lineHeight	Muda a altura de uma linha
quotes	Muda o tipo de "aspas" do texto.
textAlign	Muda o alinhamento do texto
textDecoration	Muda a "decoração" de um texto
textIndent	Muda a intendação da 1a. linha de texto
textShadow	Muda a sombra do texto
textTransform	Texto com todas as iniciais em maiúsculas
whiteSpace	Muda o comportamento de quebra de linha em espaços
wordSpacing	Muda o comp. de quebra de linha em palavras

Bordas e Margens

borderBottomColor	Muda cor da borda de baixo
borderBottomStyle	Muda estilo da borda de baixo
borderBottomWidth	Muda largura da borda de baixo
borderColor	Muda a cor das bordas todas
borderLeftColor	Muda cor da borda esquerda
borderLeftStyle	Muda estilo da borda esquerda
borderLeftWidth	Muda largura da borda esquerda
borderRightColor	Muda cor da borda direita
borderRightStyle	Muda estilo da borda direira
borderRightWidth	Muda largura da borda direita
borderStyle	Muda estilo de todas as bordas
borderTopColor	Muda cor da borda superior
borderTopStyle	Muda estilo da borda superior
borderTopWidth	Muda largura da borda superior
borderWidth	Muda largura de todas as bordas
margin	Muda todas as margens
marginBottom	Muda a margem inferior
marginLeft	Muda a margem esquerda
marginRight	Muda a margem direita
marginTop	Muda a margem superior
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
paddingBottom	Muda espaçamento interno inferior
paddingLeft	Muda espaçamento interno esquerdo
paddingRight	Muda espaçamento interno direito
paddingTop	Muda espaçamento interno superior

Layout

clear	Define em qual lado do elemento não há "float"
counterIncrement	Incrementa elementos de contagem
counterReset	Inicializa os elementos de contagem
cssFloat	Define quando uma imagem ou texto fica "float"
cursor	Muda o cursor a ser apresentado
direction	Muda a direção do texto de um elemento
display	Muda a maneira que o elemento será apresentado
height	Muda a altura de um elemento
markerOffset	Muda a distância entre marcadores de caixas
marks	Indica se os marcadores de impressão serão impressos
maxHeight	Muda a máxima altura de um elemento
maxWidth	Muda a máxima largura de um elemento
minHeight	Muda a mínima altura de um elemento
minWidth	Muda a mínima largura de um elemento
overflow	O que fazer com conteúdo que não cabem no elemento.
verticalAlign	Muda o alinhamento vertical de um elemento
visibility	Muda a visibilidade de um elemento
width	Muda a largura de um elemento

Listas

listStyleImage	Muda a imagem de marcador de lista
listStylePosition	Muda a posição do marcador de lista
listStyleType	Muda o tipo de marcador de lista

Posicionamento

bottom	Define a dist. inferior entre elemento atual e outros
left	Define a dist. esquerda entre elemento atual e outros
position	Define o tipo de posicionamento (absoluto, relativo...)
right	Define a dist. direita entre elemento atual e outros
top	Define a dist. superior entre elemento atual e outros
zIndex	Define a ordem vertical de um elemento

Impressão

orphans	Mínimo de linhas (do parág.) no inferior da página.
page	Muda o tipo ode página para apresentar um elemento
pageBreakAfter	Comportamento do "page break" depois do elemento
pageBreakBefore	Comportamento do "page break" antes do elemento
pageBreakInside	Define o comportamento do "page break" no elemento
size	Orientação e tamanho da página
widows	Mínimo de linhas (do parág.) no topo da página

Tabelas

borderCollapse	Define se as bordas se sobrepõem ou não
borderSpacing	Define o espaçamento entre bordas de células
captionSide	Muda a posição da legenda
emptyCells	Define se células vazias serão apresentadas
tableLayout	Muda o algoritmo de apresentação da tabela

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

dir	Muda ou retorna a direção de um texto
lang	Muda ou retorna o código de língua de um elemento
title	Muda ou retorna o título de um elemento.

3. 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:

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. focus()	Muda o foco para a janela atual
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. prompt()	Abre uma janela que pede informações para o usuário
window. resizeBy()	Muda o tamanho da janela de maneira relativa
window. resizeTo()	Muda o tamanho da janela de maneira absoluta
window. scrollBy()	Rola o conteúdo de maneira relativa
window. scrollTo()	Tola o conteúdo de maneira fixa

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

window.onload
window.onfocus
window.onblur

4. 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:

window.location.host	Indica o servidor e porta da URL
window.location.hostname	Indica o servidor da URL
window.location.href	Indica a URL inteira
window.location.pathname	Indica o caminho da URL
window.location.port	Indica a porta da URL
window.location.protocol	Indica o protocolo da URL
window.location.search	Indica os dados após a ? na URL
screen.availHeight	Altura da tela (menos a barra de tarefas)
screen.bufferDepth	Profundidade de cores do buffer (só IE)
screen.colorDepth	Profundidade de cores da tela
screen.deviceXDPI	Número de pontos por polegada horz. (só IE)
screen.deviceYDPI	Número de pontos por polegada vert. (só IE)
screen.fontSmoothingEnabled	Se suavizamento de fontes está ligado (só IE)
screen.height	Altura da tela
screen.logicalXDPI	Pontos por polegada normais horz. (só IE)
screen.logicalYDPI	Pontos por polegada normais vert. (só IE)
screen.pixelDepth	Resolução em cores da tela (menos IE)
screen.updateInterval	Refresh da tela (só IE)
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