Unidade 1: Introdução à World Wide Web

Prof. Daniel Caetano

Objetivo: Apresentar a história da Web, as "fases" da Web e diferenciá-la das GUIs.

Bibliografia: DOMINGUES, 2001; NIELSEN, 2000; FERREIRA, 2003; TANENBAUM, 2003; DUNCAN, 2003.

INTRODUÇÃO

Conceitos Chave:

- Problema: Desenvolver um site. Que ferramentas usar?
- Desenvolvimento de site
  - * Área enorme para profissional de TI
  - * Várias tecnologias, metodologias e ferramentas
    - + Quais usar?
- Retrospectiva histórica
  - * Contexto de cada tecnologia.

De uma forma ou de outra, a grande maioria das pessoas já está habituada à World Wide Web, WWW ou simplesmente "web". Tão habituada que a maioria de nós a toma por um recurso garantido, sem o qual é praticamente impossível trabalhar.

Sendo assim, um dos grandes ramos no qual o profissional de TI pode desenvolver seu trabalho é exatamente na elaboração de *sites web*, sejam eles institucionais ou comerciais.

As possibilidades na web são inúmeras: existem milhares de maneiras de se desenvolver um site web, cada uma delas mais adequada para um tipo de situação. Cabe a nós, desenvolvedores, decidirmos quais ferramentas, quais tecnologias e quais metodologias iremos adotar e seguir para transformar a idéia de um site em realidade.

Por outro lado, as coisas nem sempre foram desta forma. Houve épocas em que os recursos disponíveis eram bastante limitados; isso fazia com que o desenvolvimento de bons sites fosse mais difícil, mas também fazia com que as escolhas do desenvolvedor fossem mais simples: se só há um caminho, só há um caminho a seguir.

Para facilitar a compreensão das diversas tecnologias que estão disponíveis e seu correto uso, convém fazer uma análise histórica da evolução da web e quais foram as razões que levaram algumas das tecnologias hoje disponíveis a serem criadas.

1. HISTÓRICO DA WORLD WIDE WEB

Conceitos Chave:

- Internet => quase 30 anos
- No início: uso por aplicativos pouco amigáveis
- 1990: Tim Berners-Lee cria a WWW
  - * Conceito de Hipertexto aplicado
- Navegadores
  - 1990: ViolaWWW
  - 1992: Mosaic
  - 1994: Netscape Navigator
  - 1995: Internet Explorer - Guerra de Navegadores
  - 1996: Opera
  - 1998: Fim da Netscape
  - 2004: Mozilla Firefox
  - Outros... baseados no core do Mozilla (Gekko)
- Internet Ubíqua: celulares e videogames

Como muitos sabem, a Internet não é uma novidade; e ela já possui a forma que conhecemos há quase 30 anos, embora não fosse de amplo acesso ao público. A primeira rede de maior porte usando o TCP/IP entrou em operação em 1985, mas a idéia era mais antiga, tendo o termo "Internet" sido definido por volta de 10 anos antes disso.

A Internet não era, entretanto, de uso amigável. Os programas disponíveis eram bastante simples, para transferência de arquivos (FTP), terminal (TELNET) e outros do mesmo estilo. Um dos primeiros sistemas que permitiu uma "navegação" mais amigável era o sistema GOPHER, que permitia alguma navegação (menus) e texto.

Entretanto, foi apenas no início da década de 1990 que o conceito de hipertexto foi aplicado à navegação, quando a World Wide Web foi criada por Tim Berners Lee, cientista do CERN (Organização Européia para a Pesquisa Nuclear).

Hipertexto é um tipo de documento texto em que, se for selecionada uma palavra, esta seleção leva o leitor a uma outra parte do documento (ou a um outro documento), onde um novo texto, relacionado à palavra selecionada anteriormente, será apresentado.

O conceito de hipertexto foi criado há muito tempo, no princípio do século XX, por Paul Otlet, tendo sido ligeiramente aprimorado por H. G. Wells, no início da década de 1930. Foi apenas na década de 1940, entretanto, que o conceito foi melhor formalizado por Ted Nelson e Douglas Engelbart e o termo "hipertexto" foi criado apenas em 1965.

O primeiro editor hipertexto foi criado em 1968 por Andries van Dam e muitas outras aplicações utilizaram este tipo de tecnologia, mas seu uso só se tornou realmente expressivo com o advento da World Wide Web.

Um dos primeiros softwares para "navegação" pelo hipertexto da World Wide Web foi o **ViolaWWW**, criado no início da década de 1990. Pouco depois, entre 1992 e 1993, foi desenvolvido o **Mosaic**, da NCSA (National Center for Supercomputing Applications - Centro Nacional para Aplicações de Supercomputadores), que rapidamente tomou conta do mercado.

O domínio do Mosaic durou até o lançamento do **Netscape Navigator**, em 1994, que tornou-se o padrão de navegador web em praticamente todas as plataformas existentes. A empresa Netscape Communications Inc. começou a imaginar novas aplicações para seu "*suite*", independentes de plataforma, que permitiriam o usuário editar e manusear documentos em qualquer equipamento conectado, independente de seu sistema operacional... uma realidade que só tomaria corpo muitos anos depois, sem a participação direta da Netscape.

Observando o plano pretensioso da Netscape, a Microsoft temeu por seu já consagrado domínio do mercado de sistemas operacionais caseiros. Rapidamente tratou de desenvolver seu próprio navegador Web, o qual foi incluído pela primeira vez no Windows 95 OSR2, ainda no fim de 1995, chamado de **Internet Explorer**, como uma referência à ferramenta Windows Explorer.

De 1995 a 1998 a "guerra dos navegadores" foi ferrenha. O uso de táticas questionáveis de marketing por parte da Microsoft tiveram como resultado o surgimento do navegador **Opera** (1996), a extinção da Netscape Communications e o completo domínio do Internet Explorer a partir de 1999. Entretanto, a necessidade de um novo navegador para outros sistemas operacionais (o Internet Explorer só existia para Windows e MacOS), aliada à falta de segurança e problemas inerentes ao navegador da Microsoft, fizeram com que a comunidade OpenSource se movimentasse.

Aproveitando-se da liberação pública do código do antigo Netscape Communicator, formou-se a Mozilla Foundation, em 1998, para criar um novo navegador web gratuito, de código aberto e multiplataforma. Várias versões "***Mozilla Internet Suite***" foram lançadas de 1999 a 2003, até que em 2004 foi liberada a primeira versão oficial do novo navegador: **Mozilla Firefox**. Uma equipe resolver dar continuidade ao Mozilla Internet Suite, mais parecido com o antigo Netscape Navigator, com o nome de **Sea Monkey**.

Desde o lançamento do Mozilla Firefox até os dias atuais, tem havido uma nova onda de migração do Internet Explorer 6 para Firefox 1.x. No fim de 2006 foram lançadas as novas versões de ambos navegadores: Internet Explorer 7 e Firefox 2, iniciando uma nova rodada na disputa pela liderança do mercado.

No momento atual, a versão corrente do Firefox é a 3.0, lançada em 2008, já estando disponíveis os primeiros betas da versão 3.1 e já estão sendo definidas as diretrizes para a versão 4.0, ainda sem data de lançamento.

Neste meio tempo, a Internet ganhou os videogames, celulares e demais dispositivos móveis, tendo sido o Internet Explorer portado para diversos equipamentos que usem sistema operacional baseado no Windows CE e o Opera para a maioria dos equipamentos que não usam sistemas operacionais baseados no sistema da Microsoft.

2. EVOLUÇÃO DA WEB

Conceitos Chave:
- Tecnologias: evoluíram com navegadores
  * Disponíveis x Desejáveis
  * Cada tecnologia pode ser associada a uma das fases
- Fases da Web (com relação ao conteúdo)
  * Web Estática
  * Web Dinâmica / Web Ativa => Web 2.0
  * Web Semântica => Web 3.0
- Exigências
  * Web Dinâmica => Servidor
  * Web Ativa => Cliente
  * Recursos: usar ou não usar?

Desde o surgimento dos navegadores até as versões mais recentes, eles passaram por muitas mudanças tecnológicas... e como conseqüência, muito mudou também no desenvolvimento de web sites. Assim, antes de sentar e projetar um site web, é interessante avaliar as tecnologias que estão disponíveis e quais serão usadas.

É possível classificar as evoluções da Web e suas tecnologias em fases:

Web Estática: Primeira fase da web, em que os conteúdos eram estáticos, ou seja, eram criados todos manualmente e quase nunca atualizados. Esta fase passou por dois estágios: um primeiro onde as formatações visuais eram mínimas e uma segunda, em que se introduziram tabelas, mudanças de cores e fontes, etc.

Web Dinâmica: Segunda fase da web, em que os conteúdos passaram a ser dinâmicos, ou seja, modificam-se sozinhos com o tempo. As tecnologias acrescentadas, como forms, uso de linguagens script (Javascript, PHP, ASP etc) em conjunto com bancos de dados tornaram possível a criação de páginas personalizadas e com características muito mais atrativas aos usuários.

Web Ativa: Fase atual, em que foram acrescentados elementos que são executados no computador do usuário, com informações sendo apresentadas e escolhidas sem a intervenção do servidor, como nas tecnologias Java e Flash, ou com participação do servidor, como na tecnologia AJAX.

Web Semântica: Uma fase que ainda está em desenvolvimento, com impactos maiores ainda para o futuro, em que as páginas web passam a ser desenvolvidas não apenas para a leitura humana, mas também para a leitura de computadores, visando o uso concreto de agentes web (*web agents*).

> Agentes Web são programas que navegam pela Internet buscando informações de interesse para seu usuário. A idéia por trás dos "agentes web" é a mecanização de trabalho aborrecido, como buscar informações sobre um derterminado assunto, fazer pesquisa de preços, negociar horários de serviços que se encaixem na agenda do usuário etc.
>
> Já existem alguns sistemas que funcionam com base em agentes web, embora não tenhamos contato direto com os agentes. Exemplos destes sistemas são o Buscapé ou o próprio Google.

É importante lembrar que usar recursos de Web Dinâmica implicam em exigências específicas com relação ao servidor onde a página estará hospedada (como a existência de um interpretador PHP de uma determinada versão, por exemplo). Uma vez que esta exigência seja resolvida, todos os usuários terão acesso ao conteúdo.

No caso da Web Ativa, entretanto, a exigência é com relação à existência de um determinado produto (Flash, por exemplo) de uma determinada versão no micro do cliente. A vantagem quando os requisitos são atendidos é que o usuário tem uma experiência muito mais rica e rápida. A desvantagem é que o usuário pode não querer instalar o software em sua máquina (ou mesmo não poder instalá-lo, por uma série de razões) e seu site acabar ficando inacessível para estes usuários.

A dica aqui é sempre usar o menor número de recursos necessário para que o web site seja atrativo e, sempre que for usado algum recurso extra de Web Dinâmica ou Ativa, procurar requisitar sempre a versão mais antiga possível de softwares a serem instalados no servidor ou computador do cliente, dado que a probabilidade de uma versão antiga ou mais nova já estar instalada é maior a cada versão do produto que entra no mercado.

3. DAS APLICAÇÕES TRADICIONAIS AOS SITES WEB

Conceitos Chave:

- GUI x WUI: o que são
- Aplicações Web
- Foco: tarefas x informações
- Pago x Gratuito
- Usuários:
    * Desejos: conhecidos x desconhecidos
    * Adesão: cativo x nômade
    * Navegação: totalmente padronizada x levemente padronizada

> Paradigma é um modelo ou uma filosofia a ser seguida. Segundo Thomas Khun, o termo "paradigma" refere-se à teoria dominante num determinado período de tempo por uma comunidade.

A maioria das pessoas que usam computadores já lidaram com dois tipos de interfaces gráficas: aquelas dos aplicativos tradicionais, chamadas de GUIs (Graphical User Interface) e aquela encontrada na Web, com hiperlinks, chamda de WUI (Web User Interface), tendo este último nome se tornado uma infeliz escolha, vez que cada vez mais encontramos aplicações tradicionais implementadas num contexto Web, as chamdas aplicações Web.

De qualquer maneira, existem diferenças na maneira com que o usuário encara um aplicativo tradicional (o famoso "programa") e um site web, seja ele meramente informativo ou aplicativo. Estas diferenças são muito importantes porque são elas que irão guiar as decisões do desenvolvedor de um site Web. Em linhas gerais, o que se pode dizer é que um site Web não pode ser projetado como se fosse um aplicativo qualquer.

A primeira coisa que se precisa compreender é o *foco* do que se está desenvolvendo. Enquanto as páginas web clássicas (ex.: site de uma empresa) tem o objetivo de facilitar a obteção de informações, os aplicativos (e os aplicativos web, ex.: gMail) tem o objetivo de facilitar o processamento e a troca de informações. A segunda característica que é necessário perceber é que, em geral, o usuário tem expectativas distintas quanto àquilo que ele paga ou recebe de graça.

São estas duas características básicas que norteiam as expectativas do usuário, sendo possível definir dois tipos clássicos de usuário: o usuário do aplicativo comercial tradicional, e o usuário que busca informações públicas na Internet.

O usuário de aplicativo comercial tradicional

Pode-se dizer que o usuário de aplicativo comercial tradicional é um usuário conhecido, no sentido que é possível definir quais são seus anseios acerca da aplicação. Um usuário que procure um editor de imagens, como o Corel PhotoPaint ou o Adobe PhotoShop, certamente espera que estes aplicativos ofereçam recursos de edição de imagem dos mais diversos tipos. Também é razoável concluir que eles não esperam que estes aplicativos editem músicas ou filmes.

Uma outra característica interessante deste usuário é que ele é muito mais seletivo antes de comprar um software, pois envolve um custo, mas, depois que ele pago o software, ele pode ser considerado "cativo", isto é, não vai mudar para o software concorrente se não tiver uma razão muito forte para isso, dado que implicará um novo desembolso.

Finalmente, estes usuários esperam uma navegação padronizada, isto é, o menu "ajuda" sempre no mesmo lugar e funcionando da mesma forma, assim como o menu "arquivo" e "editar". Ele também espera que as janelas de configuração tenham uma dada aparência e assim por diante.

O usuário que busca informações públicas na Internet

Em oposição ao usuário anteriormente apresentado, o usuário que busca informações públicas na Internet é, via de regra, quase totalmente desconhecido. Um site de carros, por exemplo, pode contar com visitantes aficcionados em carros, em alguém que está pensando em comprar um modelo específico e aqueles que simplesmente cairam no site por acaso, por causa de alguma imagem indexada pelo Google ou Bing, por exemplo. Não se sabe o que estes usuários esperam do site, sendo este um ponto de alta relevância na elaboração do site.

Este usuário é, também, pouco seletivo inicialmente: como ele busca coisas gratuitas, ele testa tudo que for grátis; entretanto, ainda que ele tenha optado por testar um site específico, ele não pode ser considerado "cativo". É mais razoável considerá-lo como "nômade" pois, se ele achar o site ruim, ele certamente migrará. Aliás, ainda que ele ache o site bom, ainda assim ele irá consultar outros também: a diferença é que se ele achar um site bom, ele pode voltar, no futuro; se achar ruim, quase que certamente ele nunca mais voltará.

Finalmente, estes usuários esperam uma navegação menos padronizada, isto é, esperam que exista um menu de navegação ou algo semelhante, mas o local onde isso vai estar, quais as opções... dependem totalmente do site. É claro que o usuário espera que a organização faça sentido, mas ele não tem em mente nenhum padrão "prévio", o que deixam nas mãos do desenvolvedor toda a responsabilidade.

E o usuário de aplicativos web?

O usuário de aplicativos Web situa-se num "meio termo" entre os dois apresentados acima, mesclando as características de ambos, em especial nos casos em que o uso do aplicativo web envolve o pagamento de uma "mensalidade".

4. SITES-EXEMPLO

WUI:

Wikipedia:	http://www.wikipedia.org/
Governo Federal:	http://www.brasil.gov.br/
Univ. de São Paulo:	http://www.usp.br/

GUI:

Orkut:	http://www.orkut.com/
gMail:	http://www.gmail.com/
Google Maps:	http://maps.google.com/

5. TECNOLOGIAS DAS FASES DA WEB (OPCIONAL)

Conceitos Chave:
- WUI são mais jovens que as GUIs
- Ambas passaram por diversos estágios
- WUIs
    * Início: apenas apresentar informações
    * Atual: também realizar comunicação e aplicações
        + Comércio eletrônico
- Web Estática x Dinâmica x Ativa x Semântica
    * "Transição" gradual
    * Transição? => convivência de diversas fases

Apesar de as Interfaces Web com o Usuário (WUIs) serem relativamente mais jovens que as Interfaces Gráficas com o Usuário (GUIs), as WUIs também já passaram por diversos estágios de desenvolvimento, amadurecendo a cada geração em direção à sua vocação atual, que combina a transmissão de informações, presente na idéia original da Web, com o comércio eletrônico, uma área em larga expansão já há vários anos.

Pode-se dizer que a web passou pelos estágios conhecidos como "Web Estática", "Web Dinâmica" e "Web Ativa", além do conceito ainda um pouco vago da "Web Semântica". Ainda que as mudanças sejam apresentadas como gerações distintas da web, é importante ter em mente que elas ocorreram de forma contínua e suave ao longo dos anos. Além disso, uma "geração" não substituiu a outra: pelo contrário, sites com características das mais diferentes etapas de cada uma destas fases estão ainda no ar e convivem pacificamente.

5.1. Páginas Web Estáticas

- Web Estática: primeira fase
    * Básica: imagens, tabelas, listas e links
    * Dificuldade de atualização
        + Uso limitado: difusão de informação "imutável"
        + Páginas fantasma
        + Descrição de empresas e produtos (?)
    * Dificuldade de Gerenciamento
        + Expansão? Consulta pelo cliente?
    * Evolução ao longo dos anos: Cascade Style Sheets (CSS)
        + Separação do conteúdo da forma da página
        + Forma definida em um arquivo separado ("único")
        + Facilita remodelagem visual
        + Não resolve problemas técnicos de interação e personalização

A primeira fase da Web, como idealizada por Tim Berners Lee, é chamada hoje de "Web Estática". As páginas Web Estáticas são compostas apenas por elementos básicos como tabelas, imagens, listas e links.

Seu uso é bastante limitado, servindo basicamente à difusão de conteúdo que não muda (ou muda muito pouco) ao longo do tempo, com pouca capacidade de busca (limitada à capacidade do navegador do usuário). A questão do conteúdo de pouca variação (daí o nome: estático), oriunda da dificuldade de atualização de páginas construídas neste formato, foi ignorada no começo e, como resultado, há uma infinidade de "páginas fantasma" na Web, com conteúdo bastante desatualizado e que não verão jamais uma atualização por todo o resto de sua existência.

Seu uso acertado foi basicamente para a descrição dos produtos ou serviços de uma empresa (coisas que, espera-se, não sofram tanta variação ao longo do tempo), além de informações sobre objetivos de empresa (que costumam mudar apenas em intervalos de alguns anos, quando a empresa tem sua diretoria ou presidência alterada).

Entretanto, a dificuldade de atualização e gerenciamento (incluindo adição de novas informações), dificuldades para o cliente encontrar as informações que procura (pela ausência de um sistema de busca adequado) e a dificuldade inerente modificar o layout de um site (na primeira fase, as informações de visualização eram inseridas dentro das páginas) fizeram com que este formato fosse evoluído em poucos anos. A viabilidade comercial das páginas puramente estáticas teve uma vida de menos de dois anos.

Uma das inovações mais recentes da Web que poderiam ter trazido sobrevida à Web Estática foi o advento das Cascade Style Sheets, ou CSS.

A criação das Cascade Style Sheets trouxe um novo conceito ao projeto de páginas Web, em que o conteúdo é separado da definição das características visuais de uma página. Isso significa que a página descrita em HTML não deve conter qualquer informação sobre a "forma" da página: deve apenas especificar o significado do conteúdo (o que é título, o que é texto, qual é a figura a mostrar, o que é legenda etc). Toda a especificação da forma, que é atrelada ao significado do conteúdo, é definida externamente em um arquivo do tipo CSS.

Embora esta tecnologia não tire as limitações de uso das páginas web estáticas, seu uso traz maior flexibilidade visual, permitindo a criação de layouts muito mais elaborados, bonitos e - muito importante - de carregamento bastante rápido. Além disso, a separação dos aspectos visuais do aspecto de conteúdo - algo que vem de encontro à arquitetura MVC, amplamente utilizada na Engenharia de Software - facilita muito o gerenciamento de páginas e seu conteúdo, bem como a remodelagem visual de sites inteiros.

Infelizmente, no que concerne a negócios, os problemas das páginas Web Estáticas vão além dos já citados, que são basicamente aspectos técnicos.

Primeiramente, as páginas web estáticas não são capazes de receber entradas dos usuários. Os contatos com as empresas (para adquirir produtos, por exemplo) precisam ocorrer por e-mail e, em geral, não podem ser automatizados. Além disso, elas são pouco personalizáveis, comprometendo a usabilidade e dificultando a criação de usuários cativos.

5.2. Páginas Web Dinâmicas

- Web Dinâmica: segunda fase
    * Necessidades:
        + permitir buscas
        + possibilitar negócios eletrônicos
    * Ponto fundamental: formulários
        + Elementos de GUI, mas devem respeitar regras de WUI
- Paradigmas de Web Dinâmica
    * Server-Side desacoplado: CGIs, Servlets
        + Alta velocidade
        + Alta compatibilidade com clientes
    * Server-Side integrado: ASP, PHP, JSP
        + Facilidade de desenvolvimento
        + Alta compatibilidade com clientes
    * Client-Side: JavaScript, VBScript
        + Reduz demanda sobre servidor
        + Compatibilidade limitada com clientes
- Tecnologias adicionais usadas
    * Banco de Dados: dados permanentes ou de segurança
    * Cookies: dados temporários
- Vantagens
    * Personalização de Páginas
        + Páginas mais amigáveis
        + Mudanças de layout e língua
        + Apresentar informações do interesse do cliente
    * Permitir negócios
    * Fidelização do cliente
- Desvantagens
    * Carga no servidor
    * Lentidão da interface
    * Limitações na interação

A primeira evolução da Web veio basicamente para atender a uma onipresente necessidade de ferramentas de busca mais poderosas do que as fornecidas pelos navegadores, além de possibilitar o uso da web para negócios.

O ponto fundamental desta primeira evolução foi a criação dos "forms", os populares formulários, que nada mais são do que uma adaptação de elementos de entrada de dados de

uma GUI para uso em uma WUI. Os formulários devem ser usados com cuidado, seguindo basicamente as regras prescritas para as GUIs, ao mesmo tempo em que não deve ignorar as regras de desenho das WUIs.

As páginas Web Dinâmicas podem ser construídas de acordo com 3 conceitos distintos, dependendo dos requisitos da aplicação/página sendo desenvolvida. Cada um destes conceitos possui tecnologias próprias, vantagens e defeitos:

- **Server Side**, externo à página: CGIs, Servlets. Usado quando se deseja uma aplicação veloz e alta compatibilidade com clientes.
- **Server Side**, interno à página: ASP, JSP, PHP. Usado quando se deseja praticidade de desenvolvimento, além da alta compatibilidade com clientes.
- **Client Side**: JavaScript, VBScript. Usando quando se deseja reduzir a carga no servidor, sob pena de uma compatibilidade limitada.

Adicionalmente, algumas outras tecnologias são utilizadas:

- **Banco de Dados**, para armazenar perfil de usuário, conteúdos em diversas línguas, últimos lançamentos, etc.
- **Cookies**, para armazenar dados temporários como página visualizada atualmente, variáveis de sessão, etc.

Talvez a principal vantagem das páginas web dinâmicas sobre as estáticas seja que as dinâmicas permitem a realização de negócios por meio eletrônico. Mas isso não é tudo, já que elas também podem ser usadas para melhorar o tratamento do cliente por parte da empresa, através de tratamento personalizado. Isso inclui não apenas chamar o usuário pelo nome, mas também adaptar as informações ao gosto do usuário: listar os últimos produtos comprados pelo usuário, apresentar novos produtos que podem ser de interesse dele, propor promoções especiais de acordo com seu perfil, etc.

Além disso, as tecnologias por trás da web dinâmica permitem uma personalização das páginas por parte do usuário, que pode modificar o layout de uma página, escolher quais informações serão apresentadas em sua página principal, mudar o idioma quando desejado, etc.

É importante lembrar que este tipo de característica facilita a fidelização do cliente, através de uma melhor qualidade de serviço, por resolver os problemas do cliente de forma mais simples e rápida, por criar condições especiais para bons compradores, etc.

Infelizmente a Web Dinâmica também tem seus problemas. As novas tecnologias e mecanismos de funcionamento fazem com que as cargas no servidor sejam bastante maiores que na web estática. Além disso, entre o comando do usuário e a resposta do servidor existe um tempo de tráfego das informações na rede, o que se traduz em uma lentidão geral da interface, perante o usuário. Para finalizar, as interações são bastante limitadas, resumindo-se ao preenchimento de formulários e seleções em mapas.

5.3. Páginas Web Ativas

- Web Ativa: segunda fase
    * Necessidades:
        + reduzir tempos de resposta
        + melhorar a interação
    * Ponto fundamental: plugins
        + Máquinas virtuais
        + Regras dependentes da página/aplicação
- Paradigmas de Web Ativa
    * Página quase totalmente Client Side
        + Flash, Java...
        + AJAX
- Vantagens
    * Rapidez de resposta
    * Flexibilidade de interação
- Desvantagens
    * Compatibilidade limitada pelo cliente
    * Falta de padronização: GUI com regras de WUI
        + Complica aprendizado do usuário
- Sugestão:
    * Dinâmico + Ativo
    * AJAX

A segunda evolução da Web veio basicamente para solucionar alguns problemas da web dinâmica: reduzir os tempos de resposta e aumentar as possibilidades de interação.

O ponto fundamental desta segunda evolução foi o uso dos "plugins", em especial os de máquina virtual, que possibilitaram a execução de programas na máquina do usuário, quase que de maneira independente do equipamento e sistema operacional usados pelo usuário. As regras de uso e design para estes casos são as mais variadas e dependem do tipo de site e aplicação que se deseja criar.

As páginas Web Ativas são quase totalmente "client-side", ou seja, rodam na máquina cliente. Isso significa que as respostas às interações não dependem mais do tempo da rede, tornando-as muito mais rápidas, além de reduzir a carga no servidor. As tecnologias básicas do desenvolvimento atual de páginas ativas são o Java e Flash. Bancos de Dados também são usados, embora de forma menos extensiva (sob pena de aumentar muito a carga no servidor); são bastante importantes no caso do uso de AJAX.

As vantagens das páginas Web Dinâmicas estão praticamente todas presentes nas páginas Web Ativas. Além disso, são vantagens adicionais a total flexibilidade de interação e

layout, o que combinado à velocidade de resposta às ações do usuário e uso extensivo de áudio e vídeo podem melhorar muito a experiência do usuário.

Estas tecnologias têm sido muito usadas na criação de aplicativos sob demanda, em que se paga pelo tempo de uso do aplicativo, por exemplo (ou não se paga nada, como nos milhares de jogos em flash disponíveis na internet).

Infelizmente a Web Ativa também apresenta alguns problemas. O principal deles, sob a óptica do usuário, é a falta de padronização das páginas feitas usando as tecnologias de Web Ativa. Cada página tem um tipo de interação, um tipo de visual, etc... obrigando que o usuário tenha de "aprender" a usar cada nova página que encontrar.

Além disso, como as aplicações rodam em "máquinas virtuais" no lado do computador cliente, é necessário que a máquina virtual adequada, da versão correta, esteja instalada no equipamento do usuário, o que nem sempre é possível, gerando um problema de compatibilidade. Por esta razão, é comum - e até mesmo desejável - que sites do tipo "ativo" possuam também uma versão do tipo "dinâmico".

5.4. Páginas Web "Semânticas"

- Web Semântica: ainda não é exatamente uma fase
  - * Necessidades:
    - + possibilitar compreensão de páginas por agentes eletrônicos
    - + automatizar processos tediosos
    - + aumentar a qualidade da informação disponível
  - * Web Services: solução "parcial"
    - + Não estão disponíveis pelo navegador
    - + Voltados apenas aos agentes de software
- Web Semântica possibilita leitura por pessoas **e** agentes, simultaneamente
  - * Uso de XHTML, com tags estendidas para especificar significado

Apesar do advento das CSS e a conseqüente separação entre conteúdo e características visuais de uma página, a parcela de conteúdo continua sendo escrita como sempre foi feito: uma descrição de informações focada apenas na leitura humana.

Entretanto, a mecanização das tarefas do dia-a-dia tem sido quase que uma necessidade na sociedade moderna e, no que se refere à atividades realizadas na web, muito pouco se evoluiu neste sentido. Em outras palavras, é quase impossível mecanizar a maioria das tarefas que as pessoas realizam na web, como procurar um preço mais baixo de um produto ou mesmo agendar uma viagem, incluindo hotel e vôo automaticamente... é preciso que alguém realize esse processo manualmente. Quando ele é feito por softwares, na forma como a Web é apresentada hoje, o resultado é quase sempre aquém do esperado pelo usuário.

Como uma proposta para este tipo de problema, foi desenvolvido pela IBM e Microsoft uma tecnologia chamada "Web Services", que é baseada nas tecnologias da web, compartilha recursos com a web, mas está fora da World Wide Web, no sentido que não se acessa um Web Service pelo navegador!

> Especificamente, as tecnologias padrão usadas pelos Web Services são XML e HTTP, sobre os quais foram construídas tecnologias como SOAP, WSDL e UDDI. O que essa tecnologia implementa de diferente do proposto pelo HTML? A diferença básica é que são tecnologias voltadas à compreensão dos dados por parte do computador. A rede constituída pelos "Web Services" é como se fosse uma espécie de World Wide Web dos chamados "agentes de software", para que esses agentes possam realizar tarefas pelos seus "donos" de forma mecânica.

Como é possível observar, vivemos hoje num mundo segmentado: temos dados disponíveis na forma para a leitura humana e dados disponíveis na forma para leitura dos equipamentos. A proposta da Web Semântica é exatamente a unificação destas duas categorias de dados em uma única: documentos web que possam ser lidos facilmente tanto por humanos quanto por agentes de software.

O mecanismo de implementação é um documento Web normal, acrescido de "tags" que indiquem o significado semântico das palavras, para que eles possam ser compreendidos por humanos e máquinas. Por exemplo:

```
<USUARIO>
        <NOME>Fulano DeTal</NOME>
        <IDADE>18</IDADE>
        <ENDERECO>
                <RUA>Rua das Garças</RUA>
                <NUMERO>17</NUMERO>
                <BAIRRO>Bairro do Limoeiro</BAIRRO>
        </ENDERECO>
</USUARIO>
```

Os "agentes de software" são, então, programas capazes de ler páginas web descritas desta forma, que entram na mesma página web do hotel e do aeroporto que um ser humano entraria, encontram as informações que interessam ao seu "dono" e tomam atitudes necessárias, como informar ao usuário um resumo das melhores opções ou ainda tomando decisões ativas, como marcando uma viagem ou uma consulta com o médico.

6. SITES-EXEMPLO

Web Estática (em extinção):

http://macarlo.com/noticias.htm

Web Dinâmica:

http://www.ibm.com/br/pt/

http://www.microsoft.com.br/

Web Ativa:

http://www.samsung.com.br/
http://www.youtube.com/

7. BIBLIOGRAFIA

CAETANO, D. J. *Grids e Web Services: Uma combinação de futuro*. Trabalho de disciplina EPUSP, 2006.

DOMINGUES, D. G. O uso de metáforas na computação. Dissertação - Escola de Comunicações e Artes da Universidade de São Paulo. São Paulo, 2001.

DUNCAN, C.B. *Flash MX: Criação e Desenvolvimento de Web Sites*. Editora Futura, 2003.

FERREIRA, M.A.G.V. *Notas de Aula - Tópicos de Comunicação Homem-Máquina*, EPUSP, 2003.

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

TANENBAUM, A. S. *Redes de Computadores*. Editora Campus, 2003.

Unidade 2: Estrutura Básica de Documentos HTML

Prof. Daniel Caetano

Objetivo: Introduzir a linguagem HTML, estruturação de uma página e algumas tags.

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

INTRODUÇÃO

Conceitos Chave:
- Problema: Desenvolver um site web profissional e correto
- Evolução da Web => Evolução das Tecnologias/Linguagens Web
    * Ganharam novos recursos
    * Permanecem compatíveis
    * Seu uso correto mudou: evitar tecnologias "depreciadas" pelo W3C

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.

2. A ESTRUTURA DE UM DOCUMENTO HTML

Conceitos Chave:

- Leitura automatizada => Exige estrutura de documento
  - * HTML => Tem estrutura fixa
    - + Sempre se repete
- Todo o conteúdo em HTML 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 HTML, é importante apresentar a estrutura de um documento HTML.

É importante observarmos esta estrutura com atenção - e memorizá-la, porque ela será usada com freqüência e, se a estrutura de um documento HTML 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 HMTL tem 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 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

```
<HTML>
    [ 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

```
<HTML>
    <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

```
<HTML>
    <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 modificador LANG na

tag <HTML>, com o valor "pt-BR" (português do Brasil). A página ficará como apresentado a seguir.

pagina.html

```
<HTML 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:

- HTML => 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 HTML, 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 HTML, identificamos o título da página, dentro do cabeçalho, pelas tags <TITLE> e </TITLE>. Por exemplo:

pagina.html

```
<HTML LANG="pt-BR">
    <HEAD>
        <TITLE>Título da página Web!</TITLE>
    </HEAD>
    <BODY>
        Texto da página Web!
    </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
- ...

Por enquanto vamos nos limitar à informação do título que, de qualquer forma, é a mais importante 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:
- HTML => Corpo => <BODY> ... </BODY>
  * Conteúdo apresentado ao usuário pelo navegador
  * Parágrafo: <P> ... </P>
    + Uma das tags principais do HTML
- Nota: Em HTML, quebras de linha 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 HTML 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 HTML é a tag de Parágrafo**: <P>**. Como um parágrafo tem um início e um fim, então existe também uma tag de fechamento de parágrafo: **</P>**.

Assim, no exemplo a seguir por ser verificado como é possível indicar um parágrafo da maneira correta em uma página HTML.

pagina.html

```
<HTML LANG="pt-BR">
    <HEAD>
        <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 HTML, 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 HTML.

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 HTML 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 HTML usada apenas em casos muito especiais em que se deseja forçar a quebra de uma linha em uma determinada posição. Esta tag é
, de Break Row ou line BReak (quebra de linha, em inglês). Por se tratar de uma intervenção pontual, não existe tag de fechamento para
. Convém lembrar que **é incorreto** o uso da quebra de linha em substituição à tag de parágrafo.

**<P>Este parágrafo será quebrado no meio, bem aqui
e continua depois</P>**

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 (OPCIONAL)

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 HTML
- Indicação de Codificação no Cabeçalho
  - * Tag <META>
    - + Modificador HTTP-EQUIV="Content-Type"
    - + Modificador CONTENT="..."
      - = Codificação: iso-8859-1
      - = Codificação: utf-8
    - + Não tem fechamento </META>
- 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 HTML

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> não precisa de fechamento (</META>), já que ela especifica uma instrução, e não um bloco de informações.

4.2. Tags de Acentuação HTML

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 **HTML 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. TUTORIAL / EXERCÍCIO

1. Abra o "Bloco de Notas" do Windows.
2. Copie o trecho a seguir no bloco de notas, e grave com o nome de pagina.html:

pagina.html

```
<HTML LANG="pt-BR">
	<HEAD>
		<TITLE></TITLE>
	</HEAD>
	<BODY>
		<P></P>
	</BODY>
</HTML>
```

3. Sem fechar o Bloco de Notas, abra o navegador e arraste este arquivo "pagina.html" sobre a área cliente do navegador. Analise o resultado. Por que o resultado foi esse?

4. Modifique o arquivo pagina.html pelo bloco de notas, adicionando o título "Estrutura de HTML" na página, como indicado abaixo:

pagina.html

```
<HTML>
	<HEAD>
		<TITLE>Estrutura HTML</TITLE>
	</HEAD>
	<BODY>
		<P></P>
	</BODY>
</HTML>
```

5. Salve o arquivo (não feche o Bloco de Notas!) e indique ao navegador para que ele recarregue a página.

6. Observe o resultado. Onde foi parar o texto do título?

7. Adicione esta página aos "favoritos". Qual o nome que é indicado lá?

8. Acrescente a frase "Aqui temos um exemplo de estrutura em HTML." no parágrafo existente na página, e salve novamente o arquivo.

pagina.html

```
<HTML LANG="pt-BR">
	<HEAD>
		<TITLE>Estrutura HTML</TITLE>
	</HEAD>
	<BODY>
		<P>Aqui temos um exemplo de estrutura em HTML.</P>
	</BODY>
</HTML>
```

9. Recarregue a página no navegador (dica: use a tecla F5!) e analise o resultado. Bem diferentes os resultados, não?

10. Agora use o que você aprendeu para construir uma página que tenha título "**Meu Primeiro HTML**" e seja similar ao indicado a seguir (incluindo os acentos):

> Esta pode ser considerada uma página HTML muito básica, contendo alguns poucos parágrafos.
>
> Como é possível ver, não há muito mistério em se criar conteúdo para a web.

11. Recarregue a página em seu navegador. Observe o resultado. Em alguns navegadores a página aparecerá com a acentuação correta, mas em outros, não. Qual é o seu caso?

12. Se a acentuação apareceu correta, experimente modificar a codificação do navegador para "Unicode UTF-8". Isso pode ser feito pelo menu "Exibir => Codificação" ou "Visualizar => Codificação de Caracteres" (depende da versão). O que aconteceu agora?

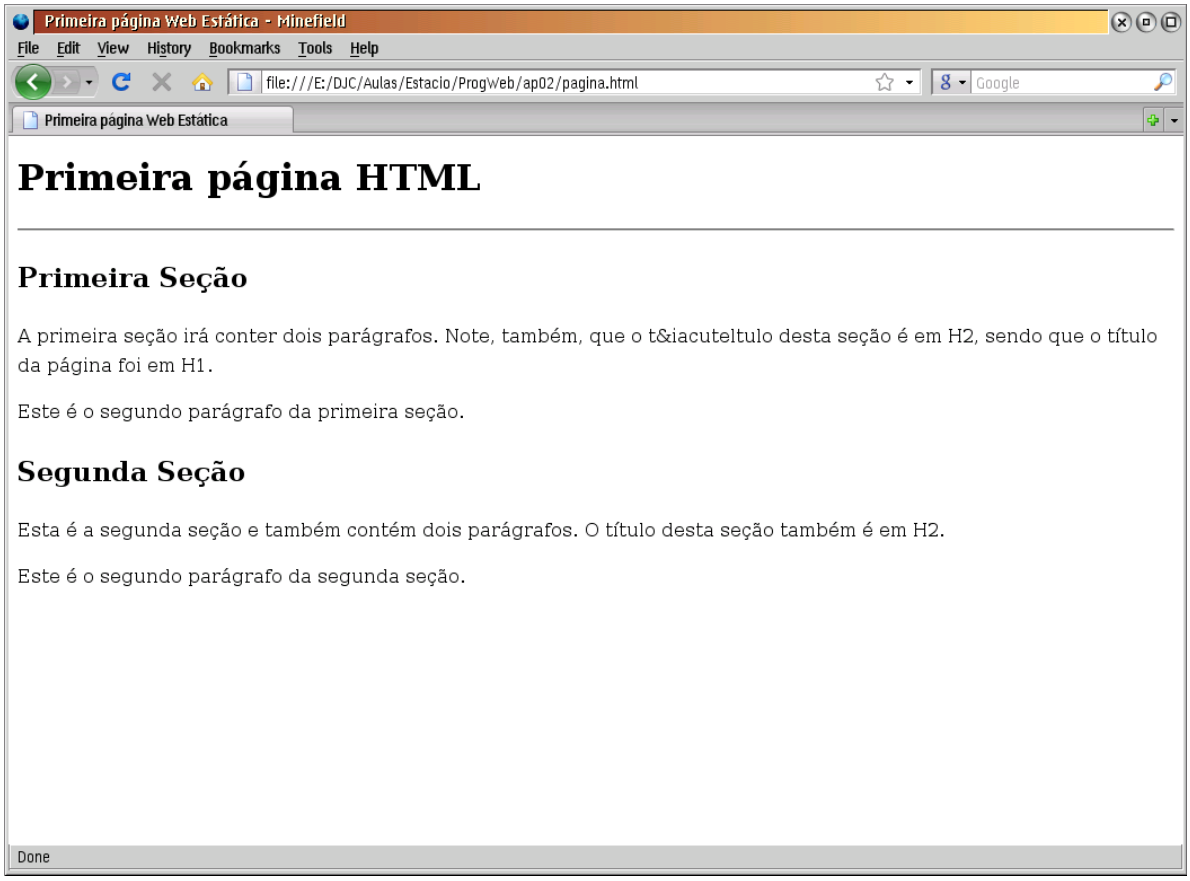
A codificação pode ter aparecido corretamente, em princípio, porque seu navegador está configurado, por padrão, para mostrar a codificação de caracteres que o seu computador usa. Entretanto, se sua página for aberta por um navegador em outro país (ou em outra língua... ou mesmo em outro sistema operacional), haverá uma boa chance de que os caracteres de acentuação **não** sejam apresentados corretamente.

Como fazer com que nossa página seja visualizada corretamente por qualquer usuário, de qualquer sistema operacional, em qualquer parte do globo? Basta usar acentuação HTML ou, pelo menos, indicar a codificação utilizada com a tag <META>!

6. EXERCÍCIOS

Nesta aula foram apresentados os usos das seguintes tags: <HTML>, <HEAD>, <TITLE>, <BODY>, <P>, <H1> a <H6>,
 e <HR>, além do uso da tag <META> para indicar a codificação de caracteres e códigos HTML de acentuação.

1. Usemos estas tags para criar a página abaixo. Ao criar a página, lembre-se de garantir a correta apresentação da acentuação, indicar a codificação de caracteres e língua base da página. **Não use
. **Não se preocupe com a formatação visual!



2. Crie uma página com algum conteúdo sobre você, sobre sua experiência e conhecimentos. Separe cada parte da página (sua vida escolar, cursos que você fez, seus conhecimentos específicos, assuntos de que gosta, lugares em que trabalhou...) com uma estrutura de títulos e também usando linhas de separação. **Não se preocupe com a formatação visual, por enquanto!**

7. BIBLIOGRAFIA

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 3: Hipertexto com HTML

Adicionando Links e Imagens em uma Página Web

Prof. Daniel Caetano

Objetivo: Apresentar os tags fundamentais de links e imagens, explicitando a função de cada elemento e observando suas características especiais.

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

INTRODUÇÃO

Conceitos Chave:
- Problema: Como acrescentar imagens em uma página web?
  Quais tags usar para estruturar um site web?
  * Tags de imagem
  * Tags de link

Até o momento foram apresentados os tags fundamentais da estrutura de uma página web, mas nada se viu sobre a estruturação de um site web. Além disso já vimos diversas maneiras de se acrescentar texto em nossa página, mas ainda não vimos como apresentar imagens.

Assim, nas próximas seções serão apresentadas as tags fundamentais para estas duas necessidades: estruturar um site web através de tags de links e acrescentar imagens à nossa página, através das tags de imagem.

1. IMAGENS NA PÁGINA WEB

Conceitos Chave:
- Grande atrativo das páginas web x Tempo de carregamento
- <IMG SRC="imagem.gif">
  * TITLE: <IMG SRC="..." TITLE="Descrição da Imagem">
  * ALT: <IMG SRC="..." ALT="Texto Alternativo">

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) não necessita um fechamento </IMG>, uma vez que por definição uma imagem já tem início e fim pré-definidos.

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 HTML 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:

```
<HTML LANG="pt-BR">
	<HEAD>
		<TITLE>Teste de Imagem 1</TITLE>
	</HEAD>
	<BODY>
		<IMG SRC="aflogo.gif">
	</BODY>
</HTML>
```

O que será apresentado da seguinte forma:

Em alguns casos, queremos definir um texto explicativo para uma figura, por caso o usuário deixe o ponteiro do mouse sobre a figura por alguns instantes, quando então a explicação aparecerá em um pequeno "balão". 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:

```
<HTML LANG="pt-BR">
	<HEAD>
		<TITLE>Teste de Imagem 2</TITLE>
	</HEAD>
	<BODY>
		<IMG SRC="aflogo.gif" TITLE="Logotipo da Empresa do Professor">
	</BODY>
</HTML>
```

Um outro parâmetro importante 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, por alguns navegadores, da mesma maneira que o texto do parâmetro TITLE, mas sua maior importância é decorrente dos navegadores para deficientes visuais, pois é o texto do ALT que é lido para descrever a figura. Assim, a página abaixo:

```
<HTML LANG="pt-BR">
    <HEAD>
        <TITLE>Teste de Imagem 2</TITLE>
    </HEAD>
    <BODY>
        <IMG SRC="aflogo.gif" TITLE="Logotipo da Empresa do Professor"
        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.

1.1. Acelerando o Carregamento de Páginas com Imagens

Conceitos Chave:
- Dependência do tamanho da imagem para definição do layout
- Especificação do tamanho da imagem no HTML
  * WIDTH: <IMG SRC="..." WIDTH="largura">
  * HEIGHT: <IMG SRC="..." HEIGHT="altura">
- 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, basta indicar a largura da imagem pelo parâmetro WIDTH e a altura da imagem, pelo parâmetro HEIGHT. Esta utilização é apresentada no exemplo a seguir:

```
<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.

Mas o que acontece se usarmos as dimensões erradas? Neste caso, a imagem será redimensionada pelo navegador para aparecer do tamanho especificado. Observe o que ocorre no exemplo a seguir.

```
<HTML LANG="pt-BR">
    <HEAD>
        <TITLE>Teste de Imagem 2</TITLE>
    </HEAD>
    <BODY>
        <P>
        <IMG SRC="aflogo.gif" WIDTH="660" HEIGHT="80"
        TITLE="Logotipo da Empresa do Professor" ALT="Logotipo Amusement Factory">
        Logotipo do site do professor.
        </P>
    </BODY>
</HTML>
```

Será apresentado como:

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 código HTML, 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 HTML. Em geral, sempre usaremos o tamanho exato da imagem na indicação do HTML.

2. 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.

2.1. Definindo Links em HTML

Conceitos Chave:
- Marcação de Link: <A>...</A>
  * HREF: <A HREF="endereco">Texto do Link</A>
  * <A HREF="http://www.uol.com.br/">Vai para o UOL</A>

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>** ... **</A>** (de *Anchor*, ou âncora), que faz uma **referência a um documento HTML** (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:

```
<HTML LANG="pt-BR">
    <HEAD><TITLE>Teste de Link</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".

2.2. Links com Títulos Explicativos

Conceitos Chave:
- TITLE: <A HREF="..." TITLE="Texto Descritivo">...</A>
    * 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>...</A>. Ele é usado da seguinte forma:

```
<HTML LANG="pt-BR">
	<HEAD>
		<TITLE>Teste de Link 3 - Com T&iacute;tulo Explicativo</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.

3. LINKS INTERMEDIÁRIOS (OPCIONAL)

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:

```
<HTML LANG="pt-BR">
    <HEAD><TITLE>Teste de Link 2</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.

3.1. Definindo um Ancoradouro no Meio de uma Página Web

Conceitos Chave:
- NAME: <A NAME="nome_ancoragem"></A>
    * 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 **NAME**, 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, não há a necessidade de "fechar" a tag com o </A>, mas é comum fazê-lo. Assim, para definir um ancoradouro intermediário chamado "Meu_Perfil" no meio de uma página, use a seguinte construção HTML:

```
<HTML LANG="pt-BR">
    <HEAD><TITLE>Teste de Ancoradouro</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 NAME = "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.

3. EXERCÍCIOS

Nas últimas aulas foram apresentados o uso das seguintes tags:

<IMG> e <A>

1. Usemos estas tags para criar a página a seguir, com a imagem http://www.caetano.eng.br/main/images/aflogo_horiz.gif no topo da primeira seção e um link para a página da disciplina http://www.caetano.eng.br/aulas/radial/web/ ao final da segunda seção.

O resultado final deve ser similar ao apresentado abaixo:

Segunda página HTML

Seção 1: Imagem

Seção 2: Link

Informações na página da disciplina Programação Web.

2. Altere esta página para que a imagem do logotipo seja também um link para a página da disciplina.

4. EXERCÍCIOS (OPCIONAL)

1. Crie uma página chamada "Links Gerais", com o nome de arquivo "links1.html", e insira links para as páginas do UOL, Terra, Yahoo e Google.
2. Crie uma página chamada "Meus Links", com o nome de arquivo "links2.html" e, nesta página, coloque links para páginas que você goste.
3. Na página "Links Gerais", acrescente um link para a página "Meus Links".
4. Na página "Meus Links", acrescente um link para a página "Links Gerais".

5. BIBLIOGRAFIA

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

Até a presente aula, vimos várias tags do HTML que servem para descrever a função estrutural de cada trecho do texto dentro da página HTML: 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 HTML.

Além do HTML, 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 HTML
  * Não dá para fazer no HTML?
    + 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 HTML.

Alguns alunos podem se perguntar: "Mas por que não fazer isso dentro do próprio HTML?". Bem, originalmente era assim que as coisas eram feitas. Aliás, muita gente programa ainda daquela forma, embora a definição de apresentação dentro do HTML 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 HTML, 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 HTML
- 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 HTML.

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 HTML, no topo da página
    d) No próprio arquivo HTML, 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 HTML
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 HTML, 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 HTML
    * <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 HTML, 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, TAG2, 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:

```
TAG, 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!)

Por enquanto, não temos uma forma mais adequada de centralizar uma imagem, mas podemos fazer o seguinte: definir um novo tipo de parágrafo <P CLASS="centered"> 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="">
</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;
```

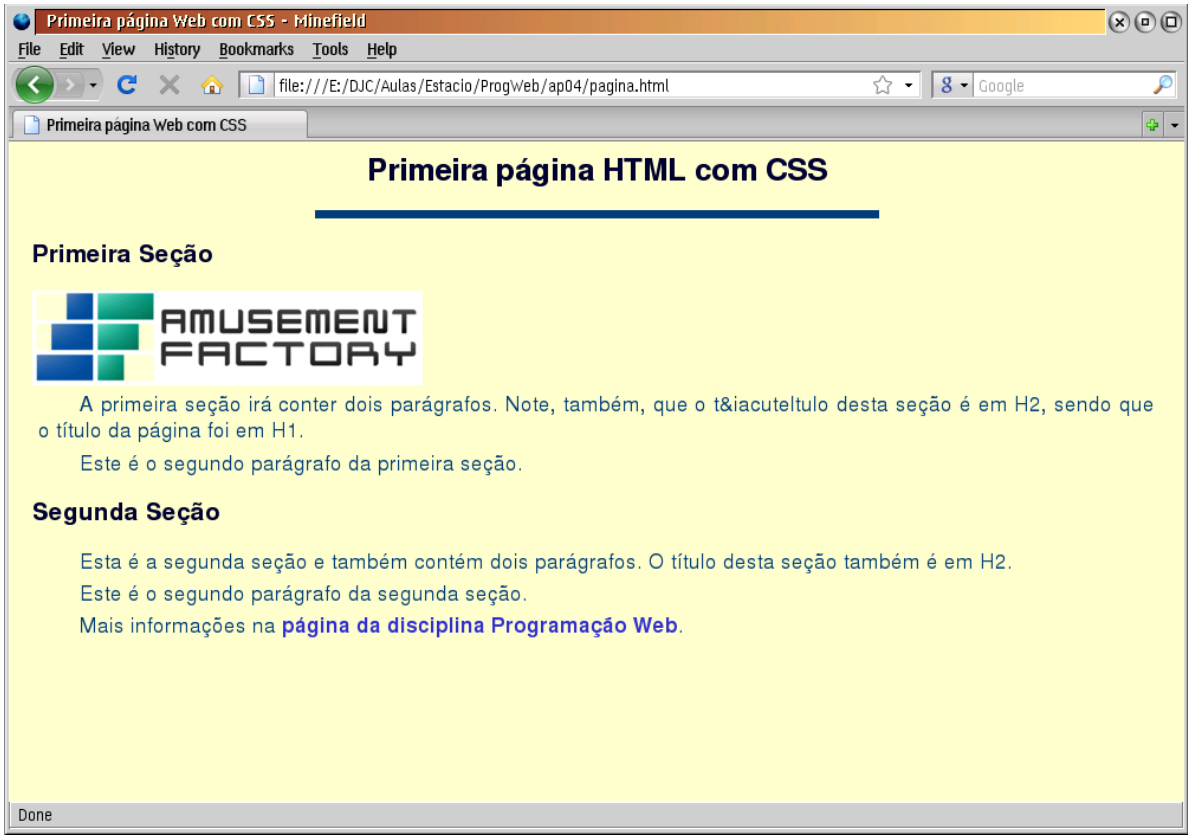
```
   font-family: garamond, times new roman, serif;
   color: #004080;
   }

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. EXERCÍCIO

1. Nesta aula, vimos a mudança de estilo para diversas tags. Usemos estas tags para fazer com que a página disponível em http://www.caetano.eng.br/aulas/radial/web/ap4ex.html fique com essa aparência (em conjunto com o professor):



2. Usando como base o Guia de Referência CSS Volume 1, disponível no site da disciplina pelo link http://www.caetano.eng.br/aulas/radial/web/css_rev_v1.pdf, modifique o arquivo CSS criado para fazer com que a página anterior fique com a aparência indicada abaixo, **sem modificar NADA no arquivo HTML**!

6. 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 5: Noções de Design

Prof. Daniel Caetano

Objetivo: Apresentar conceitos de design web que respeitam critérios de usabilidade.

Bibliografia: NIELSEN, 2000.

INTRODUÇÃO

Conceitos Chave:
- Problema: criar uma página atrativa
- Foco do Projeto Visual e Conteúdo
    * Compreensão
    * Navegabilidade
    * Importante: linguagem visual
- Sucesso x Fracasso
    * Credibilidade é tudo!
        + Cuidado com animações "bonitinhas"

Como visto anteriormente, o usuário é, muitas vezes, desconhecido. Entretanto, uma característica marcante é conhecida: o usuário de Web busca informações. Sendo assim, o projeto das páginas e de seu conteúdo deve ser focado na facilidade de compreensão e da navegabilidade, sendo a linguagem visual utilizada muito importante para uma boa aceitação da mesma.

Em última análise, essas são características que praticamente definem o sucesso ou fracasso de uma determinada página ou site. Por esta razão, continuando as dicas da aula anterior, serão apresentadas dicas relativas ao desenho da página, em termos de diretrizes gerais de desenho) e do conteúdo como um todo.

Lembre-se que credibilidade é tudo e, portanto, se a página a ser criada for séria, nada de firulas como ícones animados para mandar e-mail e coisas do tipo, se quiser que sua página tenha sucesso.

1. DESIGN DAS PÁGINAS

Conceitos Chave:

- Trata da disposição dos elementos e sua finalidade
- Área da Tela e Layout
    * Maior parte para conteúdo
    * Cuidado com excesso de informações
        + Segmentação e Organização
    * Evitar propagandas
        + Correlação com o conteúdo
    * Layout independente de resolução
        + E "bonito" em diferentes navegadores e plataformas
    * Compatibilidade com versões antigas de software
    * Separação da programação e textos do layout
    * Frames? Que frames?
        + Problema: links diretos!
    * Impressão da página!
- Tempos e Tamanhos
    * Temos de resposta mínimos
        + 0,1s / 1s / 10s
    * Usuários de Modem
    * Figuras e textos excessivamente grandes
- Links
    * Links de navegação estrutural: no texto (hipertexto interno, clássico)
    * Links associativos: no texto (hipertexto para páginas externas)
        + Não tente "amarrar" o usuário!
    * Links de referências: ao fim do texto (para quem está mesmo interessado)
    * Use palavras significativas!
    * Use títulos, quando pertinente!
    * Evite mudar completamente o padrão dos links (cores, sublinhado etc.)
    * Use sempre a mesma URL para a mesma página
    * Diferencie links externos

O design da página envolve basicamente como os elementos da página estão dispostos e a finalidade de cada uma das áreas.

ÁREA DA TELA E LAYOUT

- **Deixe a maior parte da tela para conteúdo**. Lembre-se que o usuário não entra em uma página por causa da beleza do logotipo, nem mesmo pelo incrível mecanismo que foi criado para os menus... Muito menos para ver propagandas.

- **Não abarrotar a tela com informações**, em especial se elas forem inúteis. Se há muitas informações a serem apresentadas, segmente-as em conjuntos coerentes e apresente-as

de maneira separada. Abarrotar a tela com informações só fará com que o usuário perca interesse no conteúdo como um todo.

- **Evite gastar área da tela com propaganda**. Embora nem sempre seja possível, o ideal é nem existir propaganda alguma. Se for decidido que haverá propagandas, nunca coloque mais que uma, pois elas disputarão atenção (e espaço) entre si e contra o conteúdo da página, irritando o usuário. Se for usá-las, procure colocar propagandas que tenham alguma correlação com o conteúdo da página.

- **O layout deve ser independente de resolução**, ou seja, deve ser redimensionável, como já dito anteriormente. Caso não seja possível, deve ser fixo em um tamanho adequado para visualização no máximo de dispositivos.

- **O layout deve ficar bom em diferentes navegadores e plataformas**, por isso, realize testes em todas elas! Não há nada mais desagradável do que um usuário visualizar uma página toda distorcida porque a equipe que a projetou só testou em um único navegador. Página distorcida ou não funcional significa usuário perdido... e usuário perdido significa negócios não concretizados!

- **Lembre-se dos softwares antigos**. Muitos usuários ainda usam versões antigas de navegadores e extensões. Procure desenvolver seu site para duas versões mais antigas que a recente. Se fizer para a "última moda" em navegador ou plugin, lembre-se de criar uma versão que funcione bem com versões antigas dos mesmos.

- **Separe programação de layout do conteúdo**. Lembre-se: conteúdo é especificado no HTML, que indica o significado de cada elemento. O layout é definido nos arquivos de CSS (Cascade Style Sheets), ou seja, separados do conteúdo. Isso facilita a manutenção do site, tanto na alteração de conteúdo quanto na criação de novos layouts.

- **Não use frames**. Embora em alguns raros casos o uso de frames seja aceitável (e talvez até mesmo requeridos), os frames criam problemas sérios de navegação, dificuldades para linkar partes específicas do site, além de terem uma visualização muito dificultada em dispositivos de navegação com baixa resolução.

- **Lembre-se da impressão**, ou seja, que os usuários podem querer imprimir sua página. Se sua página tiver fundo escuro com texto claro, crie uma versão alternativa de CSS para impressão, com fundo claro e texto escuro, sem figuras de fundo, de preferência. Se não quiser, adicione links para a matéria em formato postscript (.PS) ou portable data file (.PDF).

TEMPOS E TAMANHOS

- **Os tempos de resposta devem ser mínimos**, já que ninguém gosta de ficar esperando olhando para uma ampulheta.
  - **0,1s é o ideal**, para que o usuário sinta uma navegação em tempo real.
  - **1s é razoável** para que não atrapalhe o fluxo de pensamento do usuário.
  - **10s é o limite** de usabilidade.

- **Pense nos usuários de modem**. Estes tempos não são relativos ao seu próprio micro ou à uma conexão banda larga. Desprezar usuários de modem é desprezar uma grande parcela da população.

- **Cuidado com figuras e textos muito grandes**, pois eles aumentam **muito** o tempo de download. Os tamanhos ideais são:

CONEXÃO	IDEAL	MÁXIMO
- Modem:	**até 2KB**	**34KB.**
- ADSL até 1Mbps:	**até 8KB**	**150KB**
- ADSL rápidas:	**até 100KB**	**2MB.**

USO DE LINKS

- **Use links de navegação estrutural ao longo do texto**, que são aqueles que interligam partes de uma única página. Eles facilitam ao usuário encontrar o que procuram e ajudam a manter os textos enxutos, já que explicações detalhadas de alguns aspectos podem ter suas próprias páginas.

- **Use links associativos ao longo do text**o, que ligam partes de sua página com páginas que expliquem termos ou assuntos citados. Não incorra no erro de querer prender o usuário em sua página. Não é assim que se conquista usuários de Internet.

- **Use links de referência adicional ao fim do texto**, ligando sua página a páginas que falem de assuntos semelhantes e que possam ser de interesse de pessoas que tenham lido sua página do começo ao fim.

- **Use palavras significativas nos links**, não use palavras como "clique aqui" ou algo assim. Isso dificulta o uso por deficientes visuais e dificulta a geração de um menu do tipo "Links" onde o usuário possa navegar sem precisar ler a página.

- **Use títulos nos links**, dentro do possível, explicando de forma resumida para onde os links levam e qual o conteúdo da página indicada.

- **Evite mudar as cores padrão dos links**, já que os usuários geralmente estão acostumadas a elas. Se mudá-las, seja coerente.

- **Use sempre a mesma URL na indicação de links para uma mesma página**, de forma que se ela já houver sido visitada (mesmo que através de outro link), o link fique em cor diferente.

- **Diferencie links externos**, se possível, indicando-os com um estilo diferente ou alguma marca (por exemplo, escrevendo em itálico).

2. PROJETO DO CONTEÚDO

Conceitos Chave:

- O conteúdo é a parte que mais interessa ao usuário
- Texto
    * Em geral: parte mais importante
    * Problemas da leitura na tela...
    * Seja sucinto!
    * Use e abuse de hipertexto
    * Dividir conteúdo em trechos (com critério!)
    * Evite erros ortográficos!! <<= muito importante!
    * Facilite a leitura
        + Use títulos significativos e em vários níveis (2 ou 3)
        + Quebre blocos de texto em partes menores (bullets!)
        + Enfatize palavras importantes
    * Use linguagem adequada (simples!) e cuidado com o humor!
- Títulos (de página)
    * Seja descritivo e sucinto
    * Linguagem simples
    * Evitar atrair usuários "errados"
    * Evite artigos definidos / indefinidos iniciais
    * Use uma primeira palavra bem descritiva
    * Não use o mesmo título em todas as páginas
- Legibilidade
    * Cores de alto contraste (claro/escuro; escuro/claro)
    * Evite figuras de fundo (padrões sutis, baixo contaste)
    * Fontes grandes o suficiente (não fixe o tamanho!)
    * Textos... devem ficar parados!
    * Não escreva "TUDO EM MAIÚSCULA"
- Multimídia
    * Evite vídeos (a não ser que necessário / usuário tenha escolha)
        + Indique tamanho de download
    * Reduza as imagens de forma adequada (crop!)
    * Evite animações
    * Áudio... use com parcimônia (permita desligar / indique tamanho)
    * Navegação 3D?

TEXTO

O texto é, em geral, a **parte mais importante** de um site. Entretanto, a **leitura na tela tem seus problemas** e medidas especiais devem ser adotadas para tornar a experiência do usuário mais agradável.

- Seja sucinto, é a principal dica com relação ao texto. A leitura na tela é cerca de 25% mais lenta que no papel, devido à maior dificuldade de leitura oriunda do brilho e a baixa resolução da tela. Por esta razão, textos longos são cansativos e desagradáveis. Na web os textos devem ter em torno de 50% do tamanho que eles teriam em uma publicação impressa.

- **Use e abuse do hipertexto**. Isso significa que se uma dada palavra do seu texto merece um "artigo" para explicá-la, transforme-a em um link para uma página onde exista a explicação para a mesma. Lembre-se: web não é livro. O uso de hipertexto facilita a redução do tamanho dos textos, já que os termos não serão explicados detalhadamente dentro do texto principal, como ocorreria na mídia impressa.

- **Divida o conteúdo em trechos**, apresentados em páginas diferentes. Observe que isso deve ser feito com critério e não simplesmente cortando um texto no meio, em qualquer parte, apenas para dividir em mais de uma página.

- **Evite erros ortográficos**, em especial em páginas de empresa. Embora em páginas pessoais isso não seja grave, há poucas coisas que tiram mais credibilidade de uma empresa que uma página web cheia de erros ortográficos.

- **Facilite a leitura...** os usuários raramente lêem uma página toda. O uso de elementos especiais ajudam ao usuário focar naquilo que é mais importante. Ou seja: destaque o que é mais importante.

- Estruture o texto em 2 ou 3 níveis de títulos.
- **Use títulos significativos**, ao invés de títulos "bonitinhos".
- **Quebre blocos de texto em trechos menores**. Bullets ajudam nisso.
- **Enfatize palavras importantes**, usando negrito, mudanças de cor, etc.

- **Use linguagem simples**, uma vez que o usuário pode não ser um especialista no assunto sendo tratado. Como em qualquer interface, evite ao máximo o uso de termos computacionais.

- **Cuidado com o uso de humor**, alguns usuários podem não compreender ou se ofender com as brincadeiras e trocadilhos.

TÍTULOS

- **Seja descritivo e sucinto**, usando de 2 a 6 palavras. Lembre-se que os títulos são, muitas vezes, apresentados fora de contexto, em bookmarks ou em sites de busca.

- **Use linguagem simples**, sem trocadilhos que possam dificultar usuários de outras nacionalidades.

- **Evite atrair usuários que não são público alvo**, ou seja, evite títulos que possam ter uma interpretação dúbia e trazer usuários que não se interessariam por sua página. Tais usuários certamente se decepcionarão com o conteúdo encontrado.

- **Pule artigos definidos e indefinidos iniciais** do título, lembrando que muitos sites de diretório e busca organizam os links para os sites pelo título e você não gostaria que o seu site estivesse misturado em meio a um monte de sites começando com "O", "A", "Um" ou "Uma", por exemplo.

- **Use uma primeira palavra bem descritiva**, já que, como dito acima, muitos sites de diretório e busca organizam os links pelo título dos sites. Usando uma primeira palavra indicativa, você facilitará ao usuário encontrar seu site.

- **Não use o mesmo título para todas as páginas** do site, e evite que todas as páginas comecem com título igual. Lembre-se que os sites de busca apresentam o **título** da página como representação do seu conteúdo.

LEGIBILIDADE

- **Use cores de alto contraste**, ou seja, se usar fundo claro, use texto escuro. Se usar fundo escuro, use texto claro.

- **Evite figuras de fundo**, mas se usá-las, use figuras com padrões sutis.

- **Use fontes de tamanho suficiente**, e nunca fixe o tamanho da fonte, permitindo que usuários com deficiências ampliem o tamanho da mesma.

- **Mantenha os textos parados!** Embora isso pareça estranho, há pessoas que adoram usar "marquees", com textos correndo, subindo e descendo, piscando... isso atrapalha **muito** a legibilidade.

- **Não use textos "TUDO EM MAIÚSCULA".** Além de ter uma aparência agressiva, o ser humano lê com muito menos velocidade textos totalmente em maiúsculas, causando um maior cansaço visual e irritação.

MULTIMÍDIA

- **Evite vídeos**, já que eles são demasiado lentos para as redes atuais. Entretanto, quando usá-los, indique o tamanho e tempo estimado de download.

- **Reduza as imagens de forma adequada**, para reduzir seu tamanho sem perda do significado. Normalmente ela deve ser cortada (crop), ressaltando a área de interesse, e depois redimensionada (resize) para reduzi-la a um tamanho adequado.

- **Evite animações**, pois elas costumam tirar a seriedade do site, além de aumentar os tempos de download. Seu uso excessivo também pode causar irritação no usuário.

- **Use áudio com parcimônia**, e sempre permita que o usuário o desligue. Evite música de fundo automática e efeitos sonoros que não possam ser desligados. Audioclipes podem, entretanto, dar uma maior dimensão à apresentação de conteúdo, mostrando um exemplo de um trecho de um CD ou uma ópera. Nestes casos, indique o tamanho e o tempo estimado de download.

- **Evite o uso de navegação 3D**, a menos que isso **realmente** facilite a compreensão. Usar 3D para ficar "bonito" em geral tem o péssimo efeito de tornar a navegação muito lenta. Navegação por ambientes virtuais (uma cidade onde cada prédio é uma página) é também uma péssima idéia, pois é um tipo de navegação muito lento, o que vai na contramão dos interesses do usuário típico da web.

3. ATIVIDADE

1. Procure na Internet dois sites que você considera terem bons designs e dois que você considera terem maus designs. Indique seus endereços e diga por que você os considera desta forma.

2. Escolha um dos que tem, segundo sua opinião, um mau design e, usando papel, lápis e caneta, refaça o design do site, explicando pelo menos duas mudanças que usou para melhorar o design.

4. BIBLIOGRAFIA

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

Unidade 6: Noções de Projeto de Sites Web

Prof. Daniel Caetano

Objetivo: Apresentar os principais tipos de organização e navegação de sites web, além de apresentar dicas introdutórias ao projeto do geral do site.

Bibliografia NIELSEN, 2000; PRESSMAN, 2002; FERREIRA, 2003.

INTRODUÇÃO

Conceitos Chave:

- Problema: Como projetar a navegação do site?
  Onde colocar meus arquivos?
  O que coloco em cada página do website?
- Anteriormente: tags diversas para estruturação de página
  tags de links
- Agora:
  * planejar estrutura do site
    + Navegação
    + Arquivos
  * rascunhar conteúdo
    + Logotipo
    + Menus...

Até este momento foram apresentados várias tags html - incluindo links, que já permitem rascunhar um web site. Entretanto, ainda não foi feito qualquer tipo de explicação sobre a estruturação de várias páginas de um mesmo site, ou seja, em quais páginas colocar links para quais outras.

Apesar de parecer uma tarefa trivial, é fácil produzir uma página em que o usuário se sente perdido, quando não há um planejamento adequado. Por esta razão, antes de partirmos para as páginas web mais complexas, é interessante apresentar algumas considerações sobre a estrutura de navegabilidade do site e também algumas características que devem ser respeitadas no desenvolvimento de um site para obtenção de adequada usabilidade.

Convém lembrar que, para projetarmos a estrutura de um site web, temos de sempre ter em mente seu foco na informação. Além disso, é preciso evitar seguir cegamente alguns dos "hábitos" de desenho de algumas páginas web existentes, pois muitas são impregnadas por maus hábitos de projetistas que iniciaram na "arte" de produzir páginas web muito antes de qualquer estudo sobre usabilidade ter sido realizado.

1. O PROJETO DO SITE

Conceitos Chave:

- Engenharia para a Web x Engenharia de Software
  - * Clones Perfeitos?
- Princípios da Eng. para Web
  - * Confiabilidade
  - * Usabilidade
  - * Adaptabilidade
- Busca-se
  - * Desenvolvimento bem sucedido
  - * Implementação e manutenção simplificados
  - * Alta qualidade para o usuário
- Ponto de Vista de Lowe
  - * Primeiro: linhas mestras do jardim
  - * Depois: cultivo do jardim
  - * Crescimento de maneira controlada e consistente
- Diretivas de Projeto
  - * Imediatismo *(desenvolvimento rápido)*
  - * Segurança *(proteção de partes do site e dados do usuário)*
  - * Estética *(beleza e adequação)*
- Atributos de Qualidade
  - * Usabilidade *(compreensão, ajuda online, estética...)*
  - * Funcionalidade *(buscas, navegação...)*
  - * Confiabilidade *(links corretos, validação de entradas)*
  - * Eficiência *(rapidez)*
  - * Manutibilidade *(adaptabilidade/extensibilidade)*

Em primeiro lugar, é preciso deixar claro que a Engenharia para a Web não é um clone perfeito da Engenharia de Software, embora procure seguir os mesmos princípios. Neste espírito, as características mais importantes (mas não únicas) que norteiam um projeto são:

- Confiabilidade
- Usabilidade
- Adaptabilidade

Busca-se um desenvolvimento bem sucedido, bem como uma implementação e manutenção simplificados, ao mesmo tempo em que se obtém um resultado de alta qualidade sob o ponto de vista do usuário.

A ponto de vista de Lowe (1999) facilita a compreensão da maneira como devemos enxergar o desenvolvimento de um web site: "O desenvolvimento de Websites está muito mais relacionado à criação de uma infraestrutura (as linhas mestras do jardim) e depois ao "cultivo" da informação que cresce e floresce dentro deste jardim. Ao longo do tempo, o jardim (ou seja, o site na Web) vai continuar a evoluir, a se modificar e a crescer. Uma boa arquitetura inicial deve permitir que este crescimento ocorra de forma controlada e consistente."

Existem algumas outras diretivas que impulsionam o processo de projeto de um WebSite:

- Imediatismo, uma necessidade de que o site seja colocado no ar em pouco tempo.
- Segurança, para limitar o acesso a informações para alguns tipos de usuários.
- Estética, fundamental para vender produtos ou idéias, fundamental para o sucesso.

Segundo Pressman (2002), os principais atributos de qualidade de um site são:

- Usabilidade, relativa à compreensão geral do site, ajuda online, interface e estética.
- Funcionalidade, relativo à buscas, navegação e características da aplicação.
- Confiabilidade, relativo a links corretos, recuperação de erros, validação de entradas.
- Eficiência, relativo ao tempo de resposta, tempo de renderização da página.
- Manutibilidade, relativo à facilidade de correção, adaptabilidade e extensibilidade.

1.1. Diferenças no Ciclo de Projeto de Sites

Conceitos Chave:
- Ciclo de Projeto
  * Análise, engenharia, implementação, testes, validação...
- Em sites web
  * Duas Frentes: Back End (core) x Front End (gui)
  * Duas Frentes: Arquitetura, Projeto e Interface x Conteúdo e Produção

Em geral, o ciclo de projeto de um Site é exatamente o mesmo de qualquer aplicação, com planejamento, análise, engenharia, testes, avaliação e assim por diante. Da mesma forma que em projetos de software usual temos uma equipe cuidado das funções básicas do aplicativo e uma outra equipe cuidando da interface com o usuário, no projeto Web também temos duas frentes.

A primeira destas frentes cuida da arquitetura do site, o projeto da navegação e da interface. A segunda, deve se preocupar com o projeto de conteúdo e a produção.

2. ESTRUTURA DO SITE

Conceitos Chave:
- Definição é Primeiro Passo
- Estrutura
  * Baseada no conteúdo e nos usuários
  * Filosofia de navegação proposta
- Filosofias de Navegação (Estruturas Teóricas)
  * Linear; Em malha; Hierárquicas e Em teia

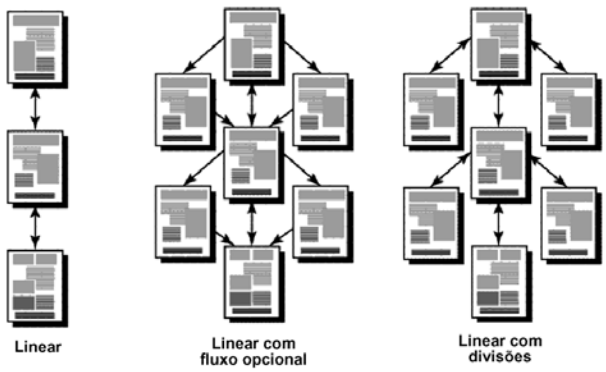
Uma das primeiras coisas a se definir em um projeto de site é a sua estrutura. De forma geral, esta estrutura é baseada no conteúdo a ser apresentado, nas classes de usuários que devem acessar a página e na filosofia de navegação proposta. As filosofias de navegação podem ser: lineares, em malha, hierárquicas, em teia, dentre outras mais caóticas.

Convém ressaltar que estas estruturas de navegação são, quando consideradas isoladamente, meramente teóricas. Na prática estas estruturas aparecem misturadas, sendo que cada trecho de um site adota uma delas.

2.1. Estruturas Lineares

Conceitos Chave:
- Usualmente para artigos longos ou escolha de produtos
- Facilidade de compreensão
- Pouco flexíveis
- Tipos
  * Linear Pura. *Ex.: artigo longo*
  * Com Fluxo Opcional. *Ex.: artigo com visualizações*
  * Com Subdivisões. *Ex.: artigo com explicações de termos*

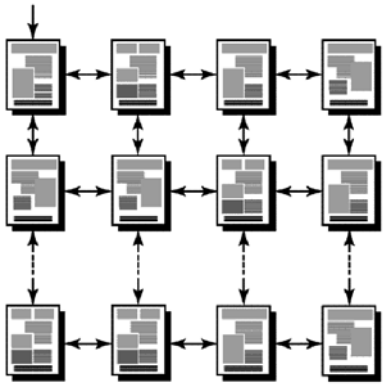
Estruturas lineares são de fácil compreensão, porém pouco flexíveis.



2.2. Estruturas Em Malha

Conceitos Chave:
- Ex.: Para visualização de produtos distintos que possuem categorias semelhantes.
- Compreensão média
- Um pouco mais flexíveis
- Categorias podem ser bidimensionais ou conter ainda mais dimensões
- Navegação normalmente por combo-boxes.

Estruturas em malha são de compreensão um pouco complexa, porém relativamente mais flexíveis.

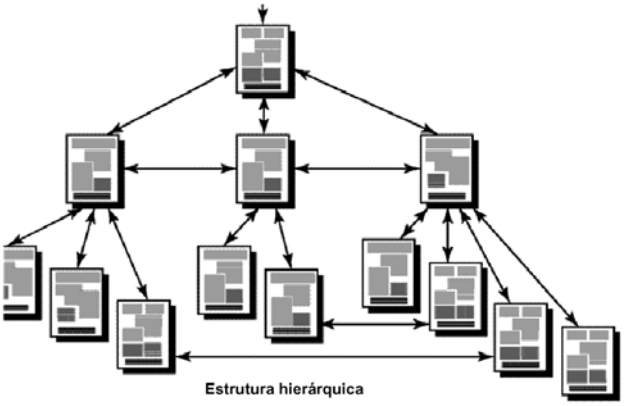


Estrutura em malha com categorias bidimensionais

2.3. Estruturas Hierárquicas

Conceitos Chave:
- Ex.: Para visualização de produtos distintos que possuem categorias distintas.
- Compreensão complexa
- Bastante flexíveis
- Navegação normalmente por menus e submenus distintos em cada página.

Estruturas hierárquicas são de compreensão complexa, porém são flexíveis.

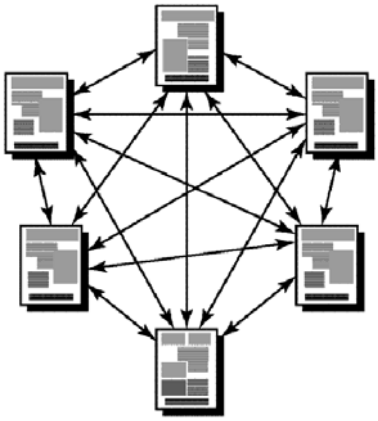


Estrutura hierárquica

2.4. Estruturas em Teia

Conceitos Chave:
- Uso generalizado.
- Compreensão pode ser complexa ou não, depende da lógica de navegação
- Totalmente flexíveis

Estruturas em teia podem ser de compreensão muito complexa, porém são totalmente flexíveis.



Estrutura em Teia (Web)

2.5. Estruturas Mistas

Conceitos Chave:
- Uso generalizado - mais comum
- Compreensão pode ser altamente complexa, se lógica de navegação não for boa
- Totalmente flexíveis

As estruturas mistas possuem partes que respeitam tipos diferentes de estruturas. Em geral temos um núcleo principal num formato em teia e, em cada ponta da teia, pode-se ter uma outra estrutura hierárquica, em malha ou linear.

3. ORGANIZAÇÃO DE DIRETÓRIOS

Conceitos Chave:
- Sem normas rígidas => sugestão?
- Nomes: letras minúsculas, sem caracteres especiais, curtos e significativos
- Diretórios: jscript, image, files, flash, styles.
- Index sempre no raiz
- Seções => Diretórios ou Arquivos?
  * Subdiretórios dentro das seções?
- Nomes Complexos ao invés de diretórios?

De maneira geral, a organização de diretórios de um site web não segue normas rígidas; entretanto, é sempre adequado adotar um padrão para um dado site e, de preferência, para todos os sites de uma empresa, de forma que todos os funcionários precisem aprender a estrutura uma única vez.

A primeira coisa a ressaltar aqui é uma que já foi dita para nomes de arquivos:

a) Nomes de diretórios, sempre que possível, em minúsculas
b) Nomes sem uso de espaços ou caracteres especiais
c) Nomes curtos mas significativos

Em geral, é interessante contar com, no mínimo, os seguintes diretórios

/jscript	- Onde serão colocados os javascripts globais
/image	- Onde serão colocados as imagens globais
/files	- Onde serão colocados arquivos em geral
/flash	- Onde serão colocados os arquivos flash gerais
/styles	- Onde serão colocadas as folhas de estilo

Adicionalmente, podem ser criados diretórios extra, de acordo com a estrutura do site, para que o diretório raiz não fique abarrotado de arquivos HTML. O arquivo *index.html* deve, obrigatoriamente, ficar no diretório raiz. Se houver um menu nesta página, entretanto, pode-se criar um diretório para cada seção indicada no menu, se ela for composta de mais que um arquivo HTML. Por exemplo:

Menu

Quem Somos	=> /quemsomos/quemsomos.html
Produtos	=> /produtos/produtos.html
Contate-nos	=> /contato.html

Já que o contato é só um formulário, não há necessidade de criar um diretório exclusivo. Alguns WebMasters, ao criar diretórios para cada menu, preferem usar um nome "index.html" para a página principal dentro de cada um daqueles diretórios.

Se houver arquivos específicos de uma única seção, alguns WebMasters preferem criar um diretório "image" dentro de cada subdiretório; outros preferem colocar todos no diretório /image que está na raiz. Cada WebMaster deve escolher o método que achar mais conveniente, procurando maximizar a organização geral.

Alguns WebMasters preferem usar nomes de arquivos complexos ao invés de usar subdiretórios. A idéia é, basicamente, anexar o nome dos diretórios no nome de cada arquivo, mantendo-os ordenados como se estivessem em diretórios apenas com o nome:

/produtos/radios/sz30f.html => produtos_radios_sz30f.html

Entretanto, esta prática é desaconselhável por tornar basante complexa a identificação de arquivos. Há uma única situação onde isso pode ser indispensável: na rara situação em que o WebMaster tiver de criar uma página em um servidor em que não possa, em hipótese alguma, criar diretórios.

4. DICAS GERAIS DE PROJETO

Esta seção contém algumas dicas para auxiliar na elaboração do layout geral e rascunho de conteúdo do site. As dicas estão agrupadas por categorias.

4.1. Estrutura do Site

Conceitos Chave:
- Estrutura mais simples possível
- Refletir como usuário enxerga o conteúdo
- Uso de telas de *splash* só quando necessário
- Não obrigue o usuário a entrar pela homepage

- A estrutura escolhida deve ser o menos confusa possível, além de **refletir como o usuário enxerga o conteúdo** do site e não a forma como a empresa é dividida.

- **Não use telas Splash** (aquelas telas de abertura), a menos que necessário. Exemplos destas situações são mudanças do endereço principal, páginas com conteúdos inapropriados para menores, etc.

- **Não force o usuário a entrar pela homepage**. Os links das páginas internas devem estar **sempre** acessíveis e, dentro do possível, não devem ser alterados.

4.2. Desenho Global

Conceitos Chave:
- Largura dinâmica ou fixa com bom tamanho (600 pixels?)
- Cuidado com metáforas
    * Web != TV
    * Carrinho de Compras?
- Região de Navegação
    * Onde usuário está, onde estava, onde pode ir

- **A largura da página deve ser dinâmica**, sempre que possível. Se não puder, ela deve ser corretamente apresentada mesmo em baixas resoluções, ou seja, para boa **visualização em torno de 600 pixels**.

- **Cuidado com o uso de metáforas** no projeto visual geral! Metáforas equivocadas podem ser "bonitinhas" mas dificultarem o uso por parte do usuário. Lembre-se, por exemplo: Web não é TV! Metáforas consagradas (como a do carrinho de compras, por exemplo) devem ser usadas, entretanto.

- **Região de navegação clara**, indicando "onde o usuário está", "onde estava" e "onde pode ir", fazendo indicações com relação à Web e ao Site como um todo.

4.3. Desenho da HomePage

Conceitos Chave:
- Página principal
- Design diferenciado => captar atenção do usuário
- Sem botão "home" => Se tiver, desligue-o!
- Deve ter logotipo maior que em outros locais => onde estou?
- Transmitir a idéia da utilidade do site (não use as "missões" para isso)
- Limite o espaço da área de notícias (a menos que ele seja um site de notícias!)
- Navegação por Menus + Mapa de Site
- Mecanismo de Busca?

A homepage é a **página principal**, onde se entra ao digitar o endereço da página.

- **Deve ter design diferenciado**, visando captar a atenção do usuário.
- **Não deve ter botão "home"**, mas se tiver, deve ficar "desligado".
- **Deve ter um logotipo maior**, respondendo à pergunta "onde estou" que o usuário possa se fazer ao chegar em seu site.
- **Deve passar a idéia do que o site faz**, mas nunca através das enfadonhas "missões" que são apresentadas em alguns sites.
- **Área de notícias restrita**, a menos que seu site seja um site de notícias. A maioria dos usuários entra em uma página buscando uma informação específica, não as últimas novidades que ocorreram em sua empresa.
- **Deve ter um mecanismo de navegação por menus**, se possível indicando também um mapa do site. Muitos usuários preferem a navegação link-a-link ou por mapa de site.
- **Deve ter um mecanismo de busca**, incluindo busca avançada. Nem todo usuário gosta de navegação link-a-link e sempre vai direto no mecanismo de busca.

4.4. Desenho das Páginas Interiores

Conceitos Chave:
- Função: apresentar conteúdo
- Logotipo menos proeminente, linkando para a homepage
- Conteúdo específico e direto

Diferentemente da homepage, as páginas interiores são direcionadas à **apresentação de conteúdo**.

- **O logotipo deve ser menos proeminente**, já que aqui ele é apenas uma informação adicional e não o foco da página interior.

- **O logotipo deve linkar para a homepage**, para que o usuário que entrou "pelo meio" da página, por um mecanismo de busca, possa descobrir mais sobre a empresa que criou aquela página.

- **Deve mostrar conteúdo específico e direto**, não ficar aborrecendo o usuário com mensagens de boas vindas.

4.5. Desenho de SubSites

Conceitos Chave:
- O que são?
  * Mecanismo adicional para categorizar informações
- Podem ter diferenças do site principal, mas não muita!
- Coerência visual com site principal
- Navegação deve ser similar
- Busca com escopo

Subsites são uma **forma interessante de categorizar informações** de maneira que apenas usuários realmente interessados as acessem. Entretanto, alguns cuidados adicionais devem ser ressaltados.

- **Podem ter diferenças do site principal**, mas é bom evitar diferenças demais.

- **Devem ter coerência visual com o site principal**, para manter a identidade do site como um todo.

- **Devem ter navegação similar**, para não frustrar o usuário por ter que aprender a navegar em mais um site diferente.

- **As buscas devem oferecer opções de escopo**, deixando claro quando uma busca é local (no subsite) ou global (no site todo).

4.6. Desenho de URLs

Conceitos Chave:
- O que são?
  * "Endereço na Internet"
- Parte mais divulgada!
- Nomes claros e breves!
- Não misture caixa alta e baixa => use só baixa!
- Endereços estáveis => "capa da edição atual"
- Produtos devem indicar página web => drivers!

A URL é a parte que é mais divulgada de seu site, de forma que as **pessoas possam acessá-lo**. Por esta razão, vale tomar alguns cuidados:

- **Escolha nomes claros**, sem caracteres malucos.

- **Escolha nomes breves**, já que nomes grandes são mais difíceis de guardar e mais propensos a erros de digitação

- **Não use mistura de caixa alta e baixa**. Dentro do possível, use todo o nome em letras minúsculas e não use, em hipótese alguma, acentuação.

- **Use endereços estáveis**. Esta dica é importante para sites com conteúdo que mudam em determinados períodos de tempo. Apesar de ser "bonitinho" ter um link como "edicaoatual.html", isso é péssimo para quem quer linkar um artigo. A pessoa faz um bookmark para um artigo no mês de agosto e ao tentar reler o mesmo artigo no mês de setembro ele sumiu.

- **Produtos devem indicar a URL de sua página web**. Embora seja comum as empresas colocar o site da empresa em seus produtos, é tão ou mais importante indicar a URL da página do produto na embalagem/manual. Poucas coisas irritam mais um usuário do que ter que "achar" a página do produto num site, quando precisa de drivers ou suporte.

5. EXERCÍCIO - PARA ENTREGA (EM GRUPO)

1. Navegue pelas seguintes páginas web e identifique sua estrutura de navegação.
http://www.wikipedia.org/
http://www.cnet.com/
http://www.ibm.com/

2. Elabore um rascunho da estrutura de um site - de preferência aquele em que já está trabalhando para o projeto, indicando as possibilidades de navegação (conforme apresentado nas figuras desta apostila).

3. Numere as páginas na estrutura de navegação e, em folha a parte, relacione todas elas com seu título e descrição, num formato de mapa de site.

4. Crie um rascunho destas páginas em html, contendo a navegação e algum texto simplificado (do tipo - "Aqui será a página disso e daquilo").

5. Crie o arquivo CSS que defina as características visuais básicas da página, lembrando dos conselhos vistos nesta aula e nas anteriores.

6. BIBLIOGRAFIA

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

PRESSMAN, R. Engenharia de Software. Ed. McGraw-Hill, 2002.

FERREIRA, M.A.G.V. *Notas de Aula - Tópicos de Comunicação Homem-Máquina*, EPUSP, 2003.

Unidade 7: Tags Adicionais do HTML

Prof. Daniel Caetano

Objetivo: Apresentar mais importantes tags do HTML.

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

INTRODUÇÃO

Conceitos Chave:
- Problema: Como representar listas e outros elementos básicos?

Nas aulas anteriores foram apresentados os tags mais importantes do HTML. Entretanto, o HTML possui uma série de outros tags para indicar muitos trechos específicos de texto, bem como algumas estruturas especiais.

1. LISTAS EM HTML

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

Um recurso bastante usado do HTML 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.
...

1.1. Listas Não Ordenadas

Conceitos Chave:
- Tag demarcador de Lista Não-Ordenada
    * <UL> ... </UL>
- Tag demarcador de Item de Lista
    * <LI> ... </LI>
- Exemplo

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

Assim, se quisermos indicar uma lista não ordenada, em HTML 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.

1.2. Listas Ordenadas

Conceitos Chave:
- Tag demarcador de Lista Ordenada
  * <OL> ... </OL>
- Tag demarcador de Item de Lista
  * <LI> ... </LI>
- Exemplo

As listas ordenadas são exatamente iguais às listas não-ordenadas, mas ao invés de serem demarcadas pelas tags <UL> ... </UL>, são demarcadas pelas tags **<OL>** ... **</OL>** (OL = Ordered List, ou Lista Ordenada). Os elementos de lista também devem ser delimitados pelas tags **<LI>** ... **</LI>**. Assim, se quisermos indicar uma lista ordenada, em HTML 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.
   1.1 Um sub-item.
2. Mais outro item.
   - Outro sub-item.

1.3 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.

2. 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.

<DEL>...</DEL> - Usado para marcar um trecho do texto como não sendo mais válido (riscado).

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

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

<EM>...</EM> - Usado para marcar um texto de forma que ele seja enfatizado.

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

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

<KDB>...</KDB> - 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.

<NOBR>...</NOBR> - Usado para indicar um trecho de texto que o navegador não deve quebrar no fim de linha.

<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.

<STRONG>...</STRONG> - 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.

<WBR>...</WBR> - Usado para indicar locais possíveis de quebra de palavra. Usado normalmente dentro de um <NOBR>...</NOBR>.

3. TAGS DEPRECIADAS

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

<B>...</B> - Texto em negrito. Não é uma tag depreciada, mas deve ser evitada. Prefira <STRONG>.

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

<CENTER>...**</CENTER>** - Centraliza o texto. Tag depreciada. Use CSS.

<DIR>...**</DIR>** - Marca uma lista de diretório. Tag depreciada. Use <UL>.

<FONT>...**</FONT>** - Muda a fonte usada em um texto. Tag depreciada. Use CSS.

<I>...</I> Texto em itálico. Não é depreciada, mas deve ser evitada. Prefira <EM>.

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

<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 <STRONG> ou <EM>, de acordo com a situação.

4. ATIVIDADE (INDIVIDUAL)

1. Crie uma página HTML básica, com um menu contendo links para: Site do UOL, Site do Terra, Site do Google, Site do Yahoo!

2. Transforme estes links em uma lista não-ordenada.

3. Adicione informações para contato com o autor e use a tag <ADDRESS> para marcar tais informações.

4. Crie um arquivo de estilo visual para a página, tornando-a mais atraente.

5. BIBLIOGRAFIA

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 8: Tags Adicionais do HTML

Prof. Daniel Caetano

Objetivo: Apresentar as tags finais do HTML.

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

INTRODUÇÃO

Conceitos Chave:
- Problema: Como representar tabelas?

Nas aulas anteriores foram apresentados quase todos os tags importantes do HTML. Ficaram ausentes alguns relativos a tabelas, mapas de imagens, frames e formulários. Esta aula apresenta estes tags, excluindo os formulários, que serão estudados em momento oportuno.

1. TABELAS

Um recurso muito útil e importante no HTML - 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 HTML 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

<table>
<tr><th>Título Coluna 1</th><th colspan="2">Título Coluna 2</th></tr>
<tr><td rowspan="2">Coluna 1, Linha 1</td><td>Título Sub Coluna 1</td><td>Título Sub Coluna 2</td></tr>
<tr><td>Sub Coluna 1, Linha 1</td><td>Sub Coluna 2, Linha 1</td></tr>
<tr><td>Coluna 1, Linha 2</td><td colspan="2">Coluna 2, Linha 2</td></tr>
</table>

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**.

2. MAPAS DE IMAGENS (OPCIONAL)

Um mapa de imagem é um mapa que associa links a determinadas regiões de uma imagem. A indicação do mapa é feita com o modificador **USEMAP** na tag de imagem. Por exemplo, esta linha indica para que o mapa chamado "mapamenu" seja usado com a imagem "menu.gif":

```
<IMG SRC ="menu.gif" WIDTH="145" HEIGHT="126" USEMAP="#mapamenu" />
```

Mas para que isto funcione, é necessário criar um mapa. Um mapa é definido pelas tags **<MAP>**...**</MAP>**, como apresentado abaixo:

```
<IMG SRC ="menu.gif" WIDTH="145" HEIGHT="126" USEMAP="#mapamenu" />
<MAP ID="mapamenu" NAME="mapamenu">
	...
```

```
</MAP>
```

Um mapa é composto por um conjunto de áreas associadas a links específicos, definidas pela tag **<AREA>**, de forma que se o usuário clicar em uma determinada área da imagem, um link será acionado. Um exemplo encontra-se no código a seguir.

```
<IMG SRC ="menu.gif" WIDTH="145" HEIGHT="126" USEMAP="#mapamenu" />
<MAP ID="mapamenu" NAME="mapamenu">
    <AREA SHAPE="rect" COORDS="0,0,80,120" HREF="page1.html" ALT="p1" />
    <AREA SHAPE="circ" COORDS="90,50,10" HREF="page2.html" ALT="p2" />
    <AREA SHAPE="poly" COORDS="5,3,9,9,1,2" HREF="page3.html" ALT="p3"
/>
</MAP>
```

Observe que a tag <AREA> tem modificadores muito importantes: SHAPE indica o tipo de área sendo definida. O modificador COORDS depende do valor de SHAPE:

SHAPE	COORDS
RECT	Os quatro vértices do retângulo (esquerda, superior, direita, inferior)
CIRC	Centro e raio do círculo (x, y, raio)
POLY	Pontos que definem a área (x1, y1, x2, y2, x3, y3 ... xn, yn)

Há ainda o modificador NOHREF="true", que serve para "excluir" uma área do mapa.

3. MOLDURAS (FRAMES - OPCIONAL)

Em geral, as páginas web ocupam toda a área do navegador, e isso é bastante desejável. Entretanto, algumas vezes o programador deseja ter a janela do navegador dividida em várias partes, com um documento html diferente em cada uma delas. Para isso são usados os "Frames" (molduras).

Sempre que se desejar particionar uma janela, deve-se criar um documento HTML específico para isso, com uma região **<FRAMESET>**...**</FRAMESET>**. Na tag FRAMESET é possível usar os modificadores ROWS e COLS, para indicar qual o tamanho das partes que se deseja dividir a janela. Por exemplo:

```
<FRAMESET COLS="100,500,*">
    ...
</FRAMESET>
```

Esse código vai dividir a janela em 3 colunas: a primeira com 100 pixels, a segunda com 500 pixels e a terceira com o que sobrar. Dentro da região FRAMESET é necessário definir o que será apresentado em cada uma destas regiões, usando a tag **FRAME**. Como há três regiões definidas no FRAMESET do exemplo, será necessário indicar três frames, como no exemplo a seguir:

```
<FRAMESET COLS="100,500,*">
	<FRAME SRC="menu.html" />
	<FRAME SRC="principal.html" />
	<FRAME SRC="noticias.html" />
</FRAMESET>
```

É possível aninhar FRAMESETs, ou seja, indicar um frameset dentro de outro. Por exemplo: se for interessante separar uma área no topo da região central para colocar um logotipo, é possível fazer da seguinte forma:

```
<FRAMESET COLS="100,500,*">
	<FRAME SRC="menu.html" />
	<FRAMESET ROWS="60,*">
		<FRAME SRC="logotipo.html" />
		<FRAME SRC="principal.html" />
	</FRAMESET>
	<FRAME SRC="noticias.html" />
</FRAMESET>
```

Cada um dos frames se comportará como uma janela diferente do navegador. Isso quer dizer que, se um usuário clicar em um link do frame onde está o menu, a página será carregada naquele frame do menu, e não no frame da página principal.

Para poder clicar em um link de um frame e carregar uma página em um outro frame, é preciso antes dar um **nome** para cada frame, usando o modificador NAME:

```
<FRAMESET COLS="100,*">
	<FRAME SRC="menu.html" NAME="menu" />
	<FRAME SRC="principal.html" NAME="principal" />
</FRAMESET>
```

Para indicar em qual frame uma página deve ser carregada, a tag <A> deve receber um novo modificador: o TARGET. Assim, um link do menu para indicar uma página a ser carregada no frame chamado "principal" deve ser algo como:

```
	<A HREF="quemsomos.html" TARGET="principal">Quem Somos</A>
```

Outros valores possíveis para TARGET são:

Valor	Significado
_top	Elimina a divisão de frames antes de frames e carrega o novo link
_self	Carrega no frame atual (default)
_blank	Abre uma nova janela com o link
frameset	Abre no frame com o nome *frameset*. Se não existir, cria nova janela.

Apesar de parecerem práticos, os FRAMES devem ser evitados nas páginas web, pois são ligeiramente desconfortáveis de operar (a não ser em casos especiais), além do fato que em navegadores rodando em telas com resolução limitada (videogames em TV, celulares, palmtops) o resultado será bastante ruim.

Se ainda assim você desejar usar frames, sempre acrescente uma seção <NOFRAMES>...</NOFRAMES>, onde você pode colocar uma versão da página sem frames ou simplesmente indicar uma mensagem para que o usuário atualize o navegador.

4. BIBLIOGRAFIA

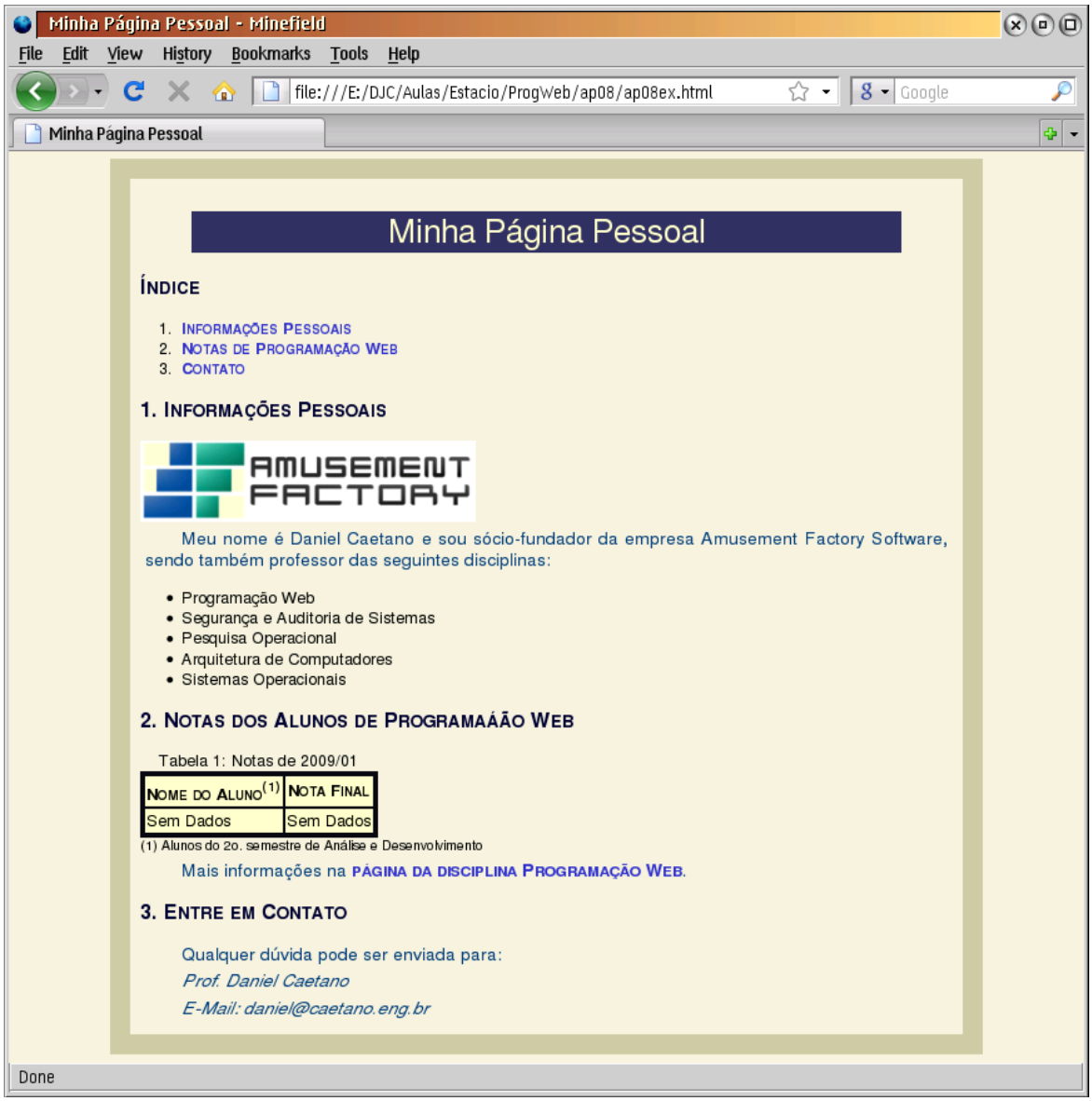
RAMALHO, J.A. *HTML 4 Prático e Rápido*. Editora Berkeley, 1999.

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

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

5. TUTORIAL

Neste momento, foram vistas praticamente todas as tags HTML (e cluindo as tags de formulários). Usemos estas tags para criar uma página bastante incrementada. O resultado final deve ser similar ao apresentado abaixo:



O link deve apontar para o endereço: http://www.caetano.eng.br/aulas/radial/web/

Unidade 9: 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 HTML
  * 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.
:lang	Especifica estilo para uma língua específica (*)

(*) Estas propriedades não funcionam automaticamente em nenhuma versão do IE.

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:)(*)

(*) Estas propriedades não funcionam automaticamente em nenhuma versão do IE.

2. TRANSPARÊNCIA

Conceitos Chave:

- Transparência/Opacidade de imagens
  - * FireFox, 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, x de 0.0 a 1.0
-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

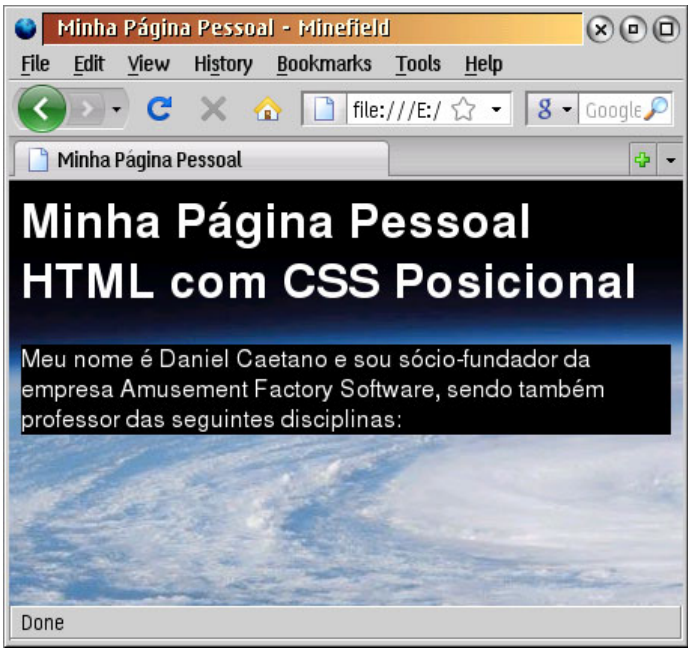
encontra sobre a figura do planeta Terra, devido ao baixo contraste entre as cores da imagem e a cor usada no texto.

exemplo.html

```
<HTML>
<HEAD>
	<LINK HREF="exemplo.css" REL="stylesheet" TYPE="text/css">
	<TITLE>Minha P&aacute;gina 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.

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:

```
<HTML LANG="pt_BR"><HEAD><TITLE>Teste</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>Conte&uacute;do</H2>
	<P>Esta &eacute; uma p&aacute;gina 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:

```
<HTML LANG="pt_BR"><HEAD><TITLE>Teste</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>Conte&uacute;do</H2>
		<P>Esta &eacute; uma p&aacute;gina 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:

```
<HTML LANG="pt_BR"><HEAD><TITLE>Teste</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>Conte&uacute;do</H2>
		<P>Esta &eacute; uma p&aacute;gina 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:

<DIV ID="titulo">

Minha Página Pessoal

</DIV>
<DIV ID="indice">

ÍNDICE

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

</DIV>
<DIV ID="secao1">

1. INFORMAÇÕES PESSOAIS

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

</DIV>
<DIV ID="secao2">

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.

</DIV>
<DIV ID="secao3">

3. ENTRE EM CONTATO

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

</DIV>

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!

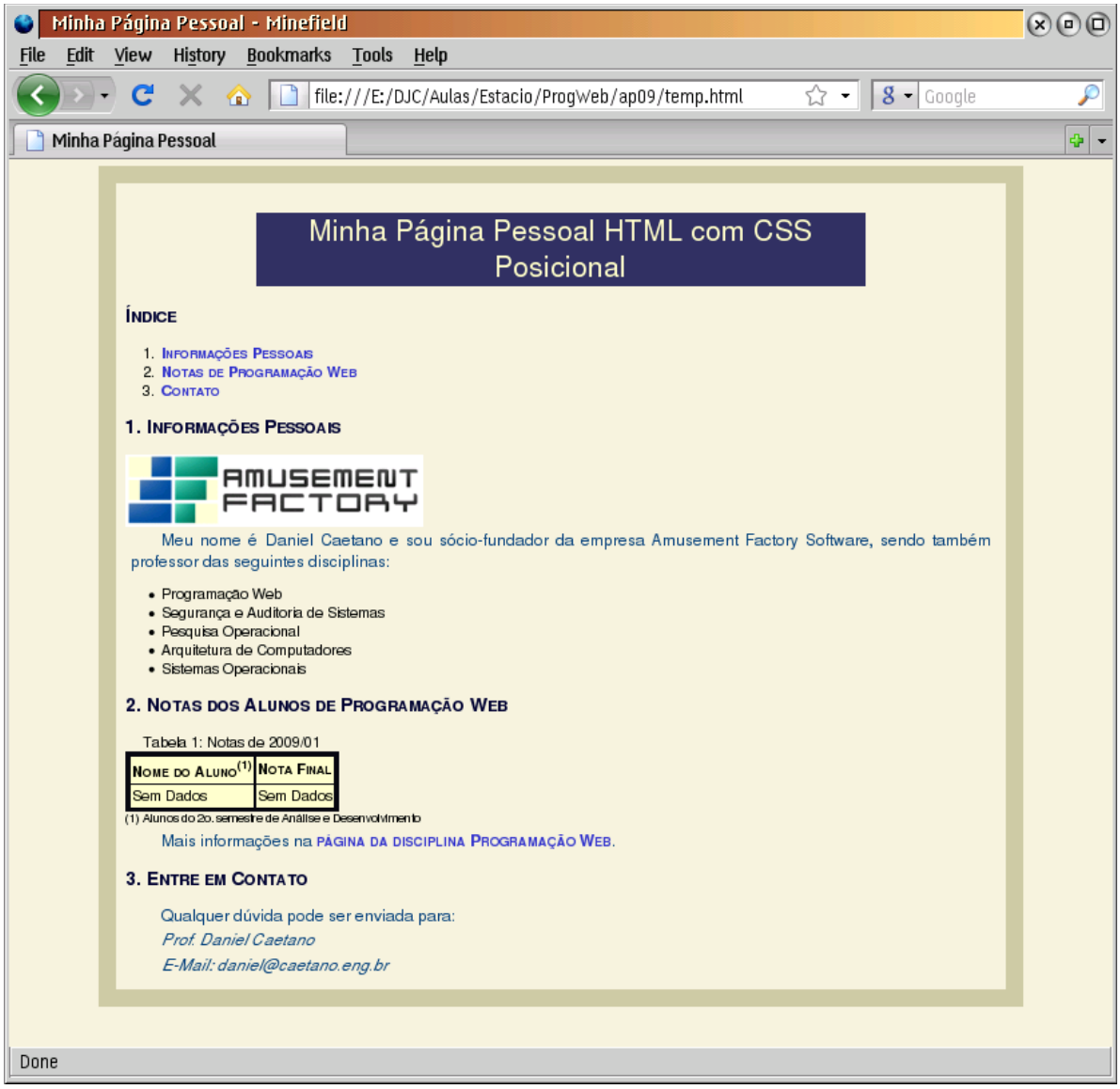
Nas próximas páginas serão apresentados alguns exemplos de modificação de propriedades e posicionamento através de CSS.

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%;
  }
```

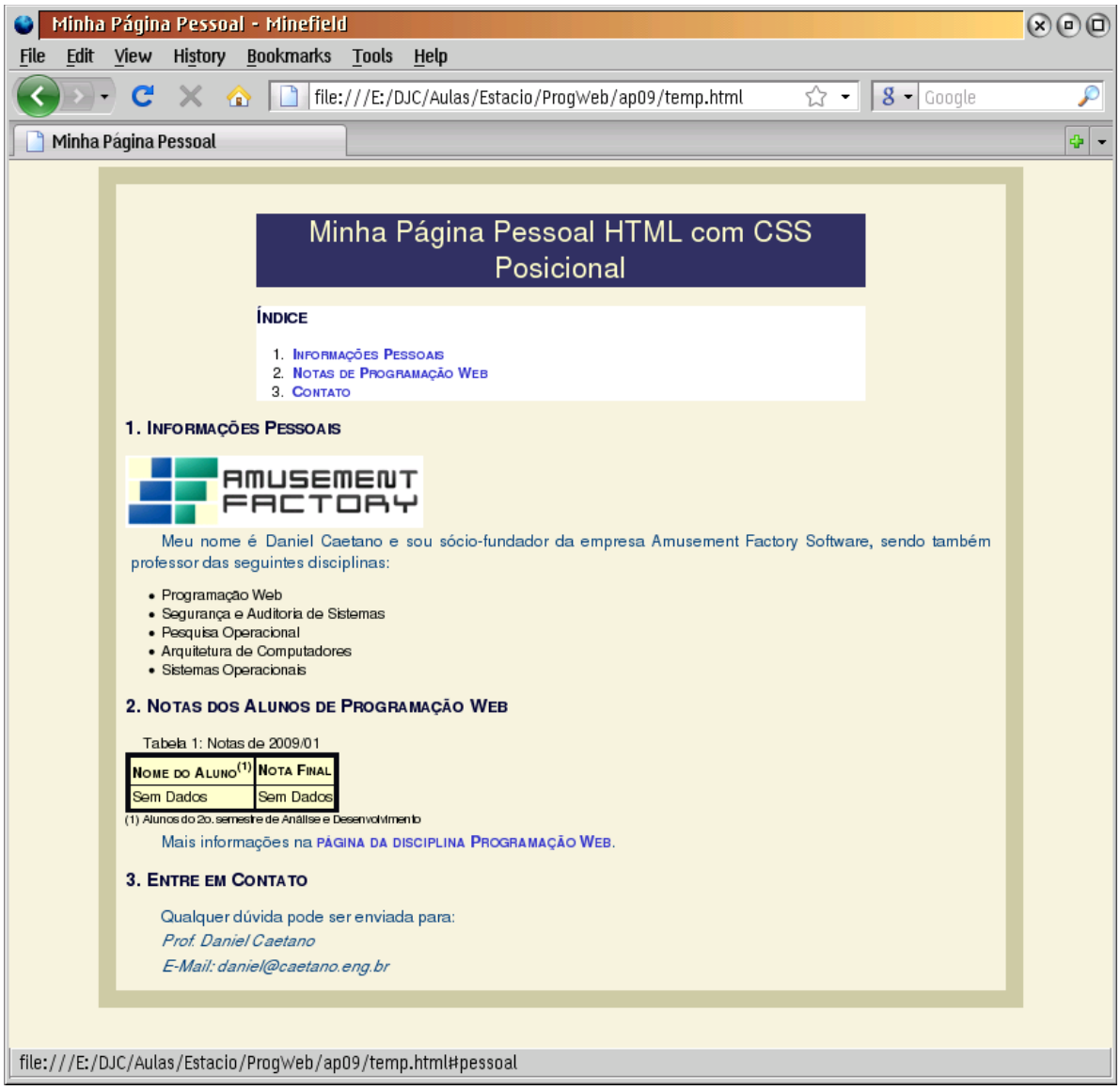
O resultado será o apresentado a seguir.



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);
  }
```

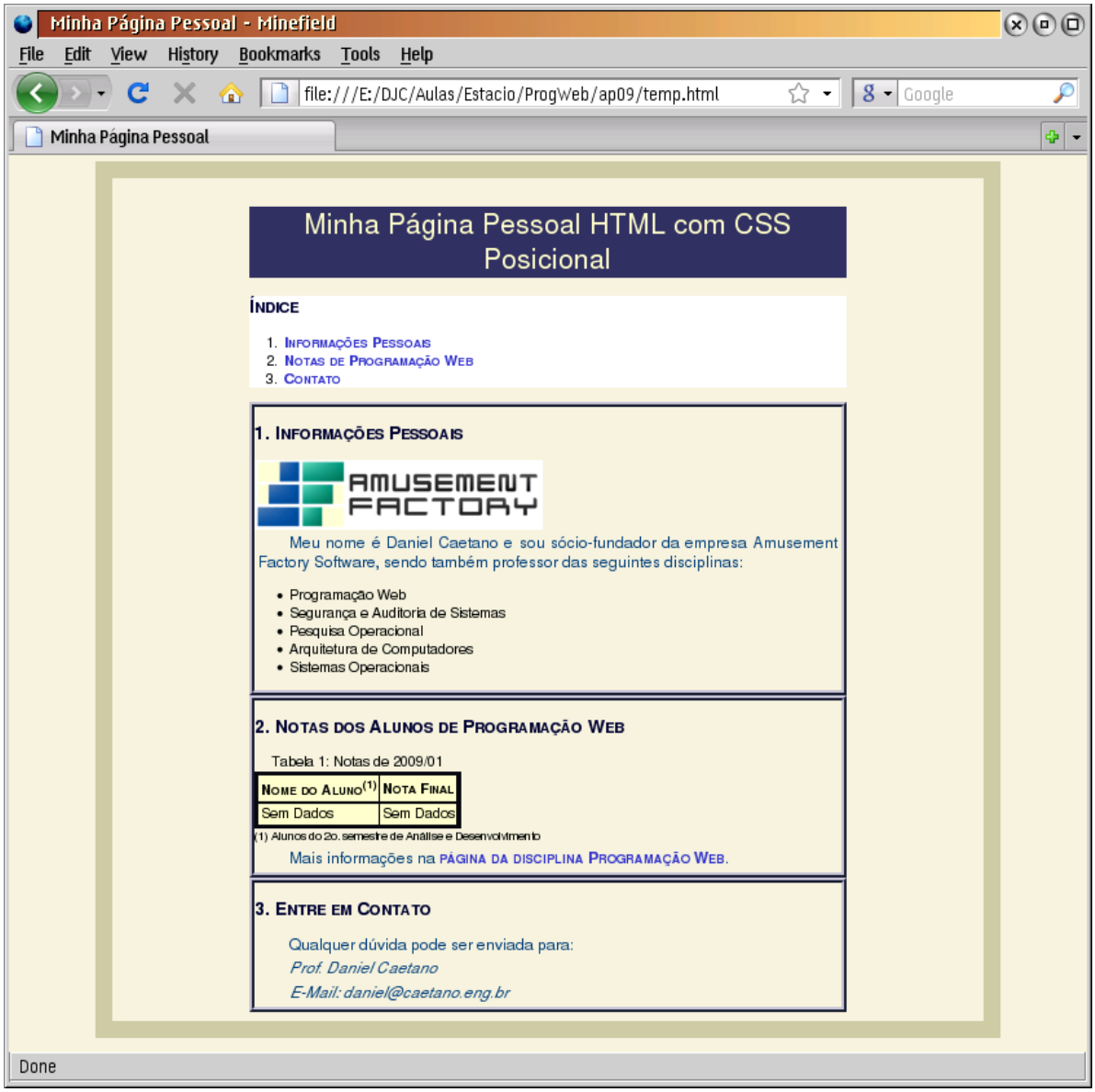
O resultado é apresentado a seguir.



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;
  }
```

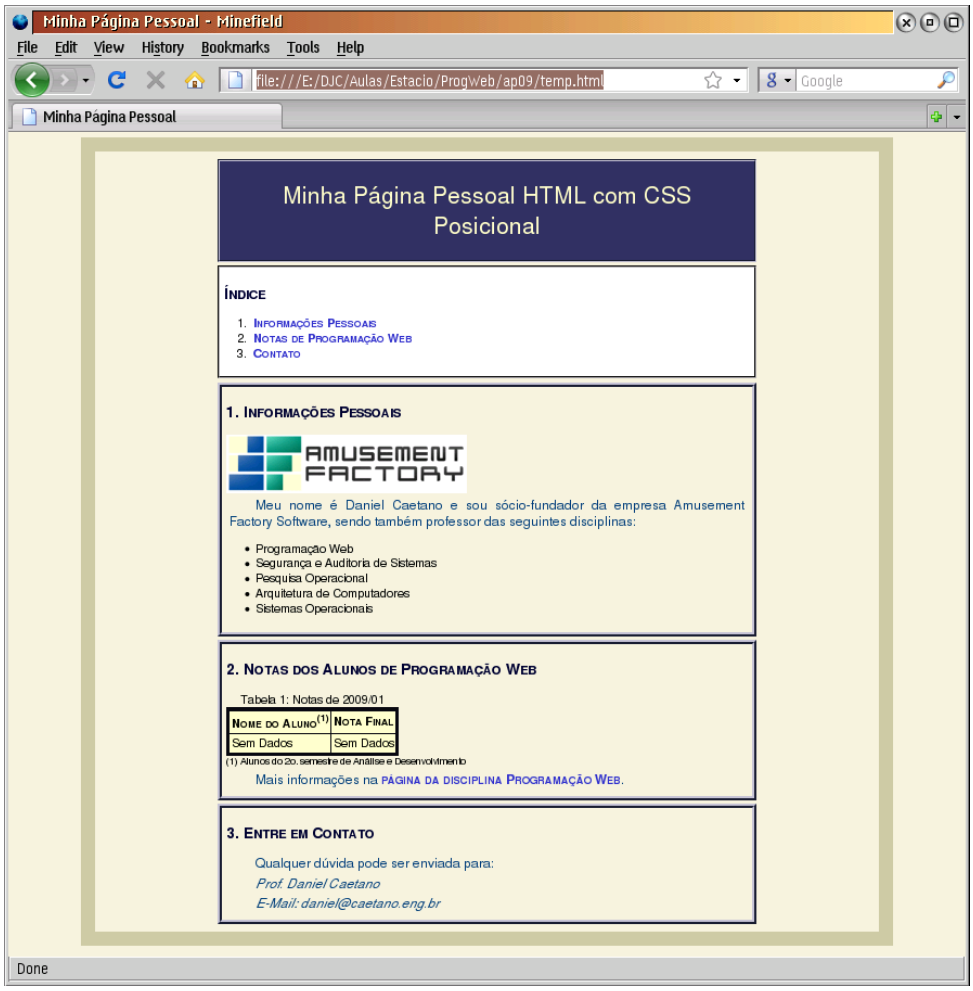
Isso corrige a diferença para o FireFox, mas tem dois efeitos colaterais: se reduzirmos demais a janela, o texto ficará cortado. Da outra forma o texto não é cortado, mas sairá para fora das regiões delimitadas. Ou seja: de qualquer forma o resultado de uma janela muito pequena não será bonito. Veremos 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:



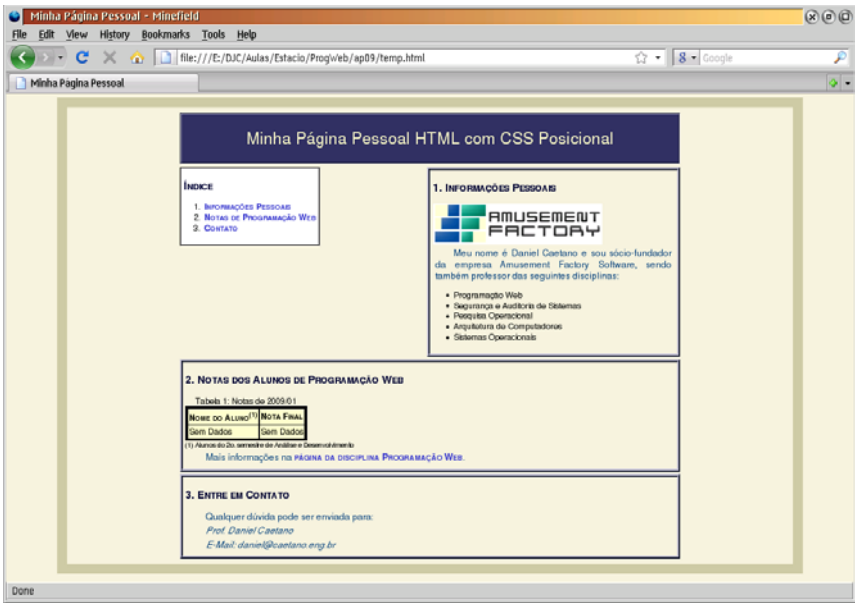
Certo, usamos alguns efeitos interessantes. Mas o layout não mudou muito com relação ao que tínhamos antes. Isso ocorre porque apenas criamos as regiões com os DIVs e mudamos algumas poucas propriedades de cores e posição delas, mas não alteramos significativamente o *fluxo de informações* da página.

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;
   }
```

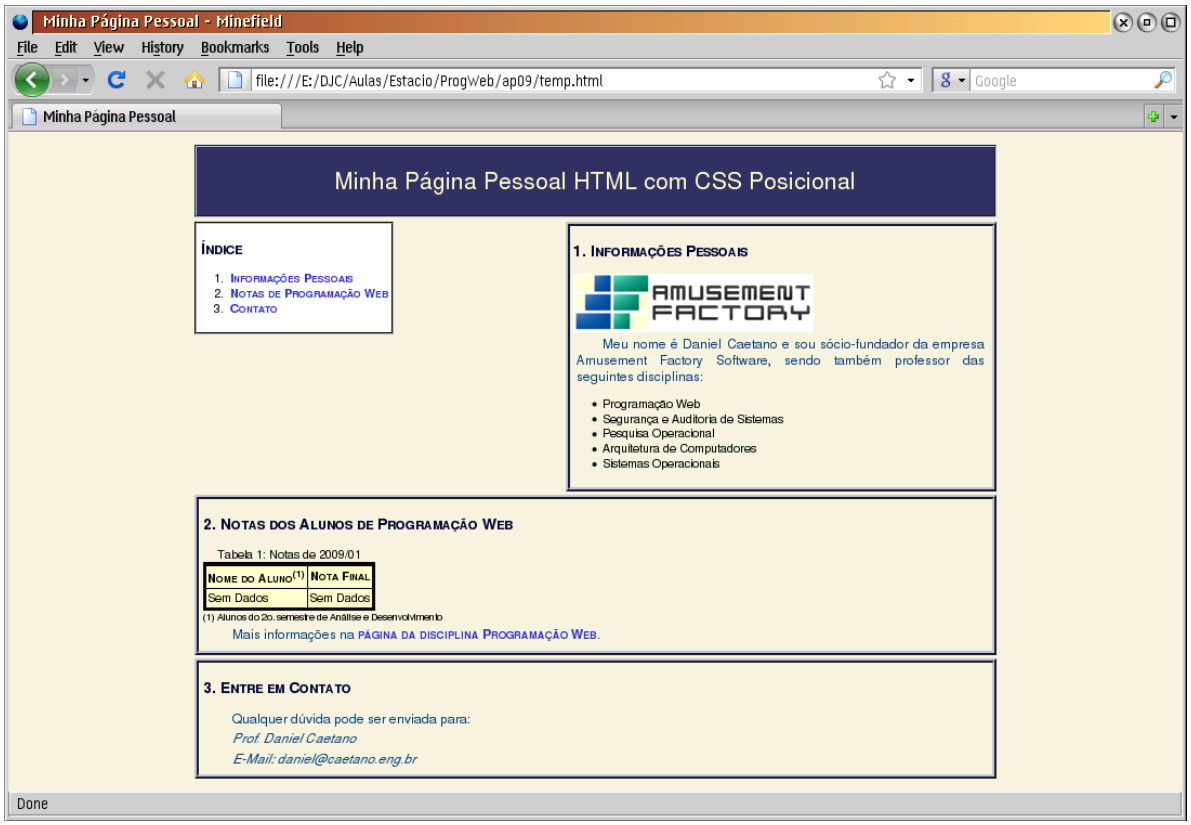
Faça a modificação e veja o que ocorreu:



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:

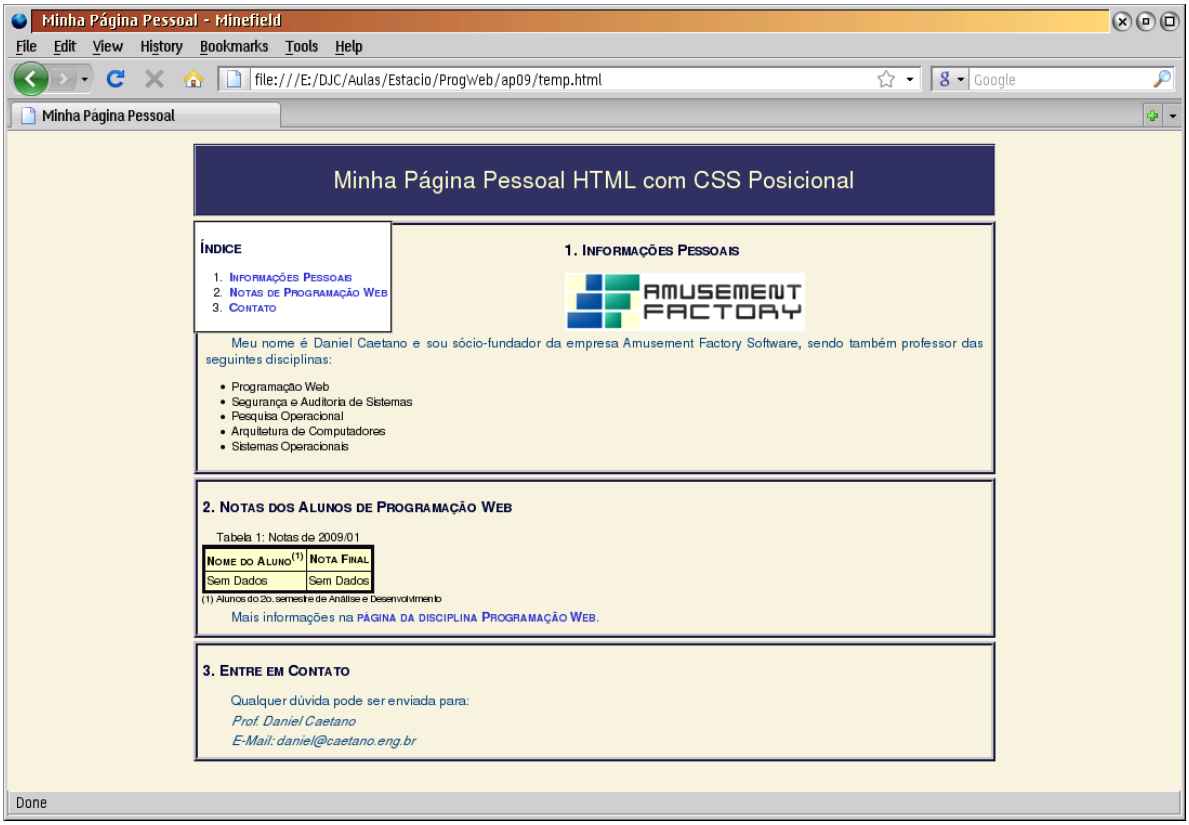


Antes de continuarmos a "ajeitar" nossa página, lembram-se do segundo efeito colateral do "overflow: hidden"? Pois bem: retiremos o "overflow: hidden" das seções 1 a 3:

```
#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).

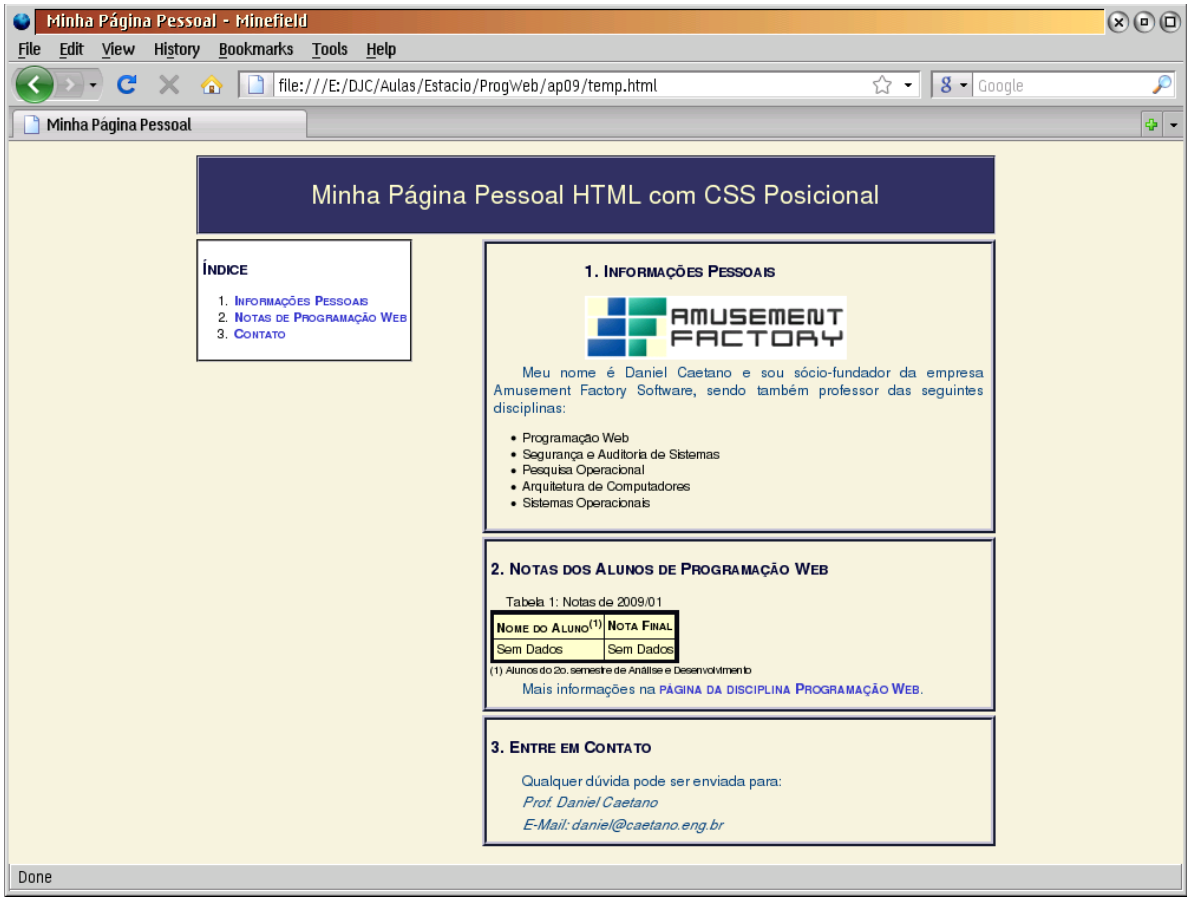
No caso das seções 2 e 3, como não há elementos à esquerda do bloco, a margem continua sendo contada a partir da borda da janela. É muito importante compreender estes "efeitos colaterais" de algumas tags para conseguir produzir exatamente o efeito que desejamos em nossas páginas.

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:

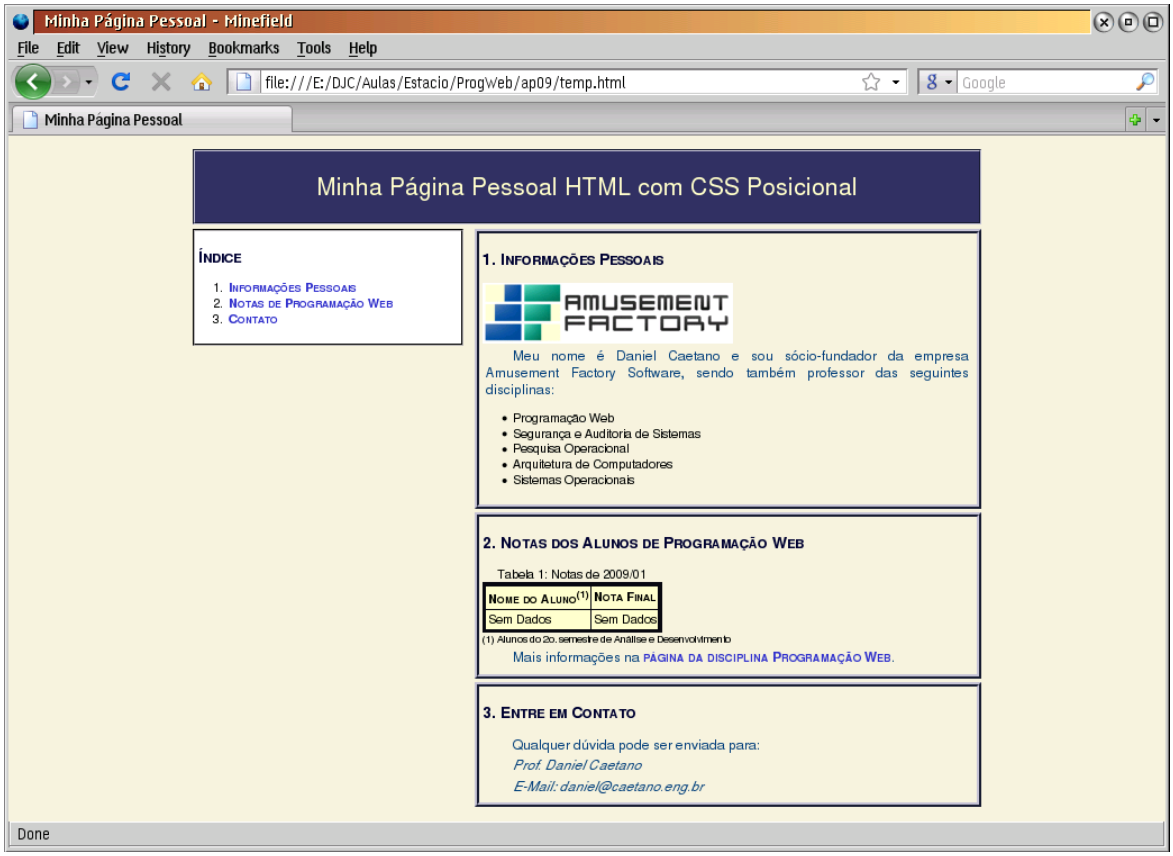


Melhorou bem! Observe que o texto da seção 1 parece meio "empurrado". Isso ocorre porque a seção índice tem uma margem esquerda definida. Vamos mdificar a seção índice de

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 10: FrontPage como Ferramenta de Protótipo

Prof. Daniel Caetano

Objetivo: Apresentar a ferramenta MS FrontPage e apresentar seus principais recursos.

INTRODUÇÃO

Com o objetivo de facilitar a vida do desenvolvedor, dezenas de ferramentas para desenvolvimento web foram criadas, mas poucas delas se destacaram tanto quanto o FrontPage, e isso não é apenas porque ele vem com o Office, mas sim pelo grande número de recursos que ele oferece.

Para o leigo ou o desenvolvedor iniciante, o FrontPage parece bastante promissor e atraente. No entanto, para o desenvolvedor profissional, pleno ou sênior, ele apresenta mais amarras do que seria interessante. Exigindo que tudo seja configurado por meio de janelas, torna o trabalho mais lento do que o necessário; além disso, apesar de a versão 2003 oferecer o recurso de configurar estilos, o processo é burocrático e pouco prático.

O sistema de "modelos" (ou templates) do FrontPage gera páginas gigantes em termos de tamanho e, via de regra, o código gerado usa uma porção de tags depreciados, e a estrutura padrão de arquivos para o site, criada pelo FrontPage, é uma completa bagunça. Para completar, o FrontPage foi descontinuado, sendo a 2003 sua última versão.

Depois de tudo isso, o aluno deve se perguntar a razão pela qual uma ferramenta "tão ruim" é abordada no curso.

Primeiramente é importante dizer que não há ferramentas "muito melhores". O DreamWeaver, da Adobe, é mais flexível e produz melhores resultados, mas não resolve todos os problemas e também custa bem mais caro. Adicionalmente, o FrontPage é uma excelente ferramenta para criar alguns protótipos realistas de visual de páginas, isto é, protótipos que tenham um layout possível.

Note que o FrontPage **não permite tudo que é possível** em HTML+CSS (ou, pelo menos, não sem editar direto o código), mas **tudo que ele permite é possível**. Isso é muito importante para não mostrarmos para o cliente nenhum tipo de layout impossível.

Por estas razões - e pela facilidade de uso - teremos nesta aula uma breve introdução à operação básica do MS FrontPage 2003.

1. VISÃO GERAL

O FrontPage tem um funcionamento bastante similar ao de todos os programas da família Office, podendo ser usado como uma espécie de "Word Muito Travado", isto é, um Word em que não se consegue fazer as coisas exatamente como você desejaria.

As opções da barra de menu são as clássicas, com exceção de Dados e Quadros. O menu de Dados serve para inserir informações específicas para servidores rodando o serviço SharePoint e o menu Quadros serve para especificar frames. Nenhuma destas duas opções serão cobertas por este tutorial.

O FrontPage permite a visualização de diversas páginas simultaneamente, sendo a seleção feita por abas que ficam logo abaixo dos botões de atalho, acima da área cliente da janela.

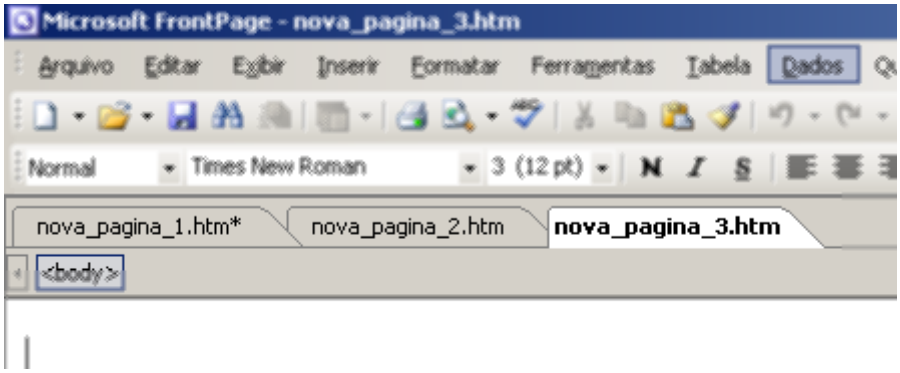


Figura 1: Abas indicam que vários documentos estão sendo editados

Outra característica importante do FrontPage são os tipos de visualização. É possível escolher entre eles no canto inferior esquerdo.

Figura 2: Os quatro tipos de visualização do FrontPage

Design mostra a visualização como supostamente ela seria vista pelo usuário. Funciona bem se você não estiver usando nenhum recurso complexo e não estiver preocupado em gerar código "ruim", isto é, cheio de tags depreciadas. Observe, enquanto digita, que neste modo são apresentadas algumas tags na linha logo abaixo das abas. Estas tags ajudam você a ver a "configuração" do que está digitando no momento (quais tags estão abertas).

Divisão mostra um modo misto: você pode editar como no modo design na parte de baixo, mas o código aparece em cima. Você pode igualmente editar o código em cima e o resultado aparecerá em baixo. Funciona bem, também.

Código mostra apenas o código fonte HTML da página, com identificação de cores na sintaxe. Algumas pessoas gostam de usar o FrontPage neste modo.

Visualização tenta mostrar o resultado mais próximo possível do que seria visualizado no navegador. Infelizmente o interpretador de página dele é o do Internet Explorer 5, que está bastante desatualizado.

Como uma alternativa ao modo visualização, o FrontPage tem no menu Arquivo a opção "Visualizar no Navegador", que abre o navegador escolhido com a página atual.

2. TUTORIAL

Antes de mais nada, quem não está com o programa aberto, deve abri-lo. Depois de aberto, em geral o FrontPage apresenta uma página em branco, no modo Design. Neste ponto, você pode usar o FrontPage como se fosse um editor estilo Word.

Você pode inserir tabelas, figuras, da mesma maneira que no Word, embora os recursos disponíveis sejam um tanto limitados, mas existe um elemento especial: o link, que será visto mais adiante.

Vamos começar definindo o título da página, palavras-chave, pano de fundo e outras características deste tipo. Para isso, deve ser usado o menu **Arquivo > Propriedades**.

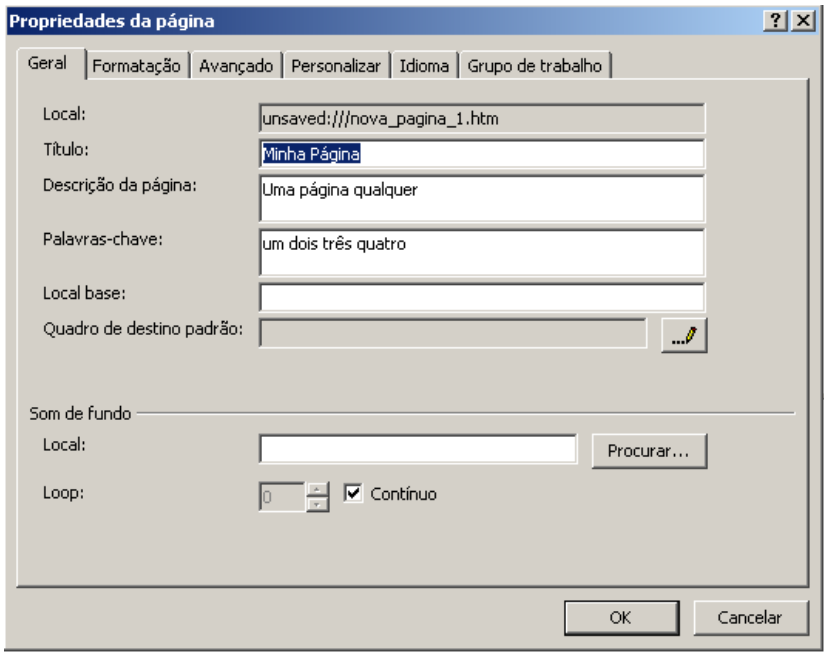


Figura 3: Propriedades serve para mudar as características básicas da página

Na aba "Geral" é possível definir o título da página (aquele que aparece na barra título do navegador), colocar uma descrição da página e até algumas palavras-chave (que serão

inseridas usando a tag "meta"). É possível ainda indicar "Local Base" que é o endereço base padrão da página, o quadro (frame) no qual esta página aparecerá e até mesmo um som de fundo para esta página.

Configure como indicado na figura e selecione a aba Formatação.

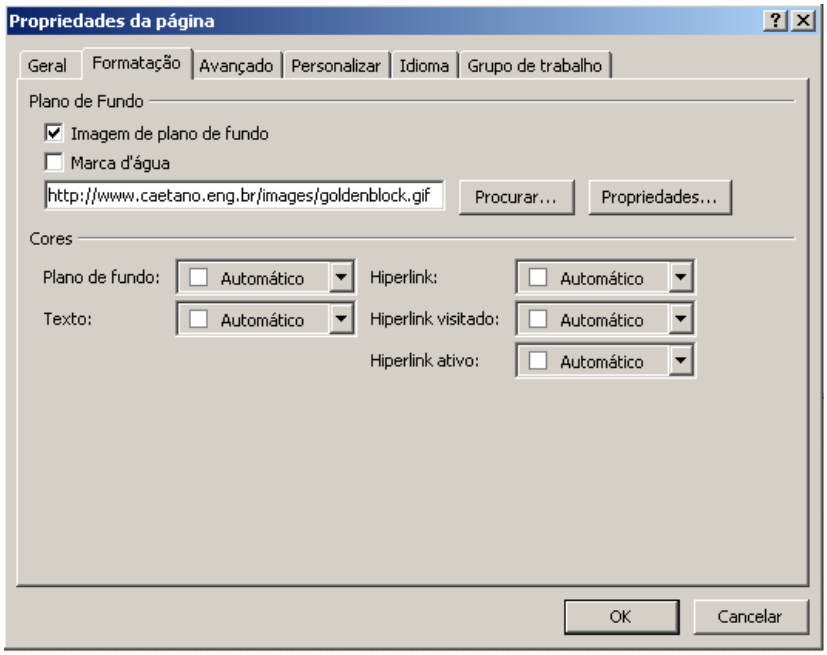


Figura 4: Aba de Formatação da janela Propriedades

Na aba "Formatação" é possível definir o plano de fundo (imagem), cores de fundo, texto, hiperlink, link ativo, visitado... Note, porém, que o uso destes recursos irá inserir tags e propriedades de tags depreciadas: como já vimos, estes parâmetros deveriam ser definidos na folha de estilo. Como estamos usando o FrontPage apenas para protótipos, não há problema em definir estas características por aqui, se não pretendermos usar o código do FrontPage como base para o código da página, o que, de todo modo, não é recomendável.

Para o nosso tutorial, configure a página conforme indicado na figura, e depois selecione a aba "Avançado".

Na aba "Avançado" é possível configurar as margens do documento e também alterar os estilos do corpo, além de definir a aparência do link quando o cursor do mouse estiver sobre ele. Nenhuma destas características serão alteradas neste turorial. A aba "Personalizar" permite definir alguns detalhes específicos no cabeçalho, e também não serão tratados neste tutorial. Selecione, então, a aba "Idioma".

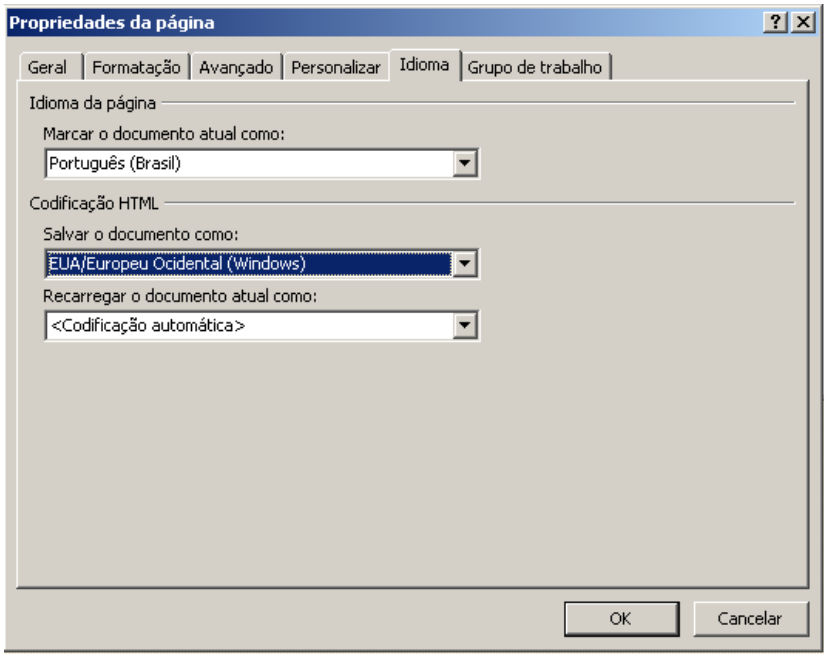


Figura 5: Aba de Idioma da janela Propriedades

A aba "Idioma" permite configurar a língua padrão do documento, bem como identificar a codificação de página do mesmo. Lembre-se da importância destas definições, em especial porque o FrontPage não se preocupa em indicar a acentuação no formato HTML. A aba "Grupo de Trabalho" também não será vista neste tutorial e, sendo assim, após configurar a aba "Idioma" como indicado, o aluno pode clicar no botão "Ok".

O fundo da página ficará preenchida com a imagem indicada previamente. Se for analisado o código (selecionando modo "Divisão" ou modo "Código"), podem ser vistas algumas tags que não vimos no curso, mas que são perfeitamente válidas.

A tag **<meta http-equiv="Content-Language" content="pt-br">** define a linguagem do documento, de maneira similar a <HTML LANG="pt-BR">. Vale lembrar que o modo usado pelo FrontPage, neste caso específico, é mais compatível com navegadores antigos.

A tag **<meta name="keywords" content="um dois três quatro">** acrescenta palavras-chave à página, fornecendo elementos para um sistema de busca como o Google encontrar o assunto da sua página de uma maneira mais consistente.

A tag **<meta name="description" content="Uma página qualquer">**, por sua vez, acrescenta uma descrição da sua página, em poucas palavras. É útil também para que sites de busca possam apresentar a descrição de seu site.

Por outro lado, é possível ver que a tag BODY apareceu com um modificador:

<body background="http://www.caetano.eng.br/images/goldenblock.gif">

Esse modificador *background* é depreciado, ou seja, não deve ser usado em sites profissionais. Ele é um recurso antigo, que era usado antes da existência do CSS. Como, mais uma vez, estamos apenas fazendo um layout para apresentar ao cliente, não nos preocuparemos com isso.

Se tivesse sido definido um estilo na aba "Avançado", ele teria aparecido também dentro da tag <BODY>, o que é "dentro do padrão", mas deve ser evitado, já que o ideal é que estilos estejam definidos no arquivo CSS.

Digite agora o texto:

Teste de Título com H1

Note que, no código, este título foi inserido como um simples parágrafo <P>...</P>. Para transformá-lo em um título H1, selecione o texto digitado e clique na caixa de estilos, selecionando "Título 1" em seguida.

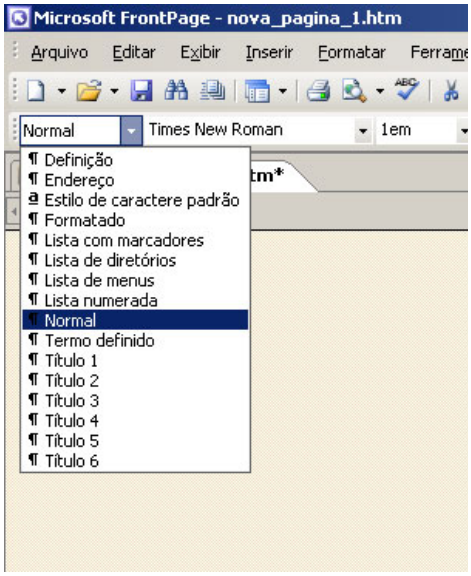


Figura 6: Caixa de seleção de estilos

Se você observar o código verá que o FrontPage inseriu mais algum lixo no código do nosso título, mas vamos ignorar isso na elaboração do layout.

Vamos agora inserir uma imagem. Clique no ícone "Inserir Imagem do Arquivo" (ou, pelo menu, **Inserir > Imagem > Do Arquivo**) e selecione o arquivo ***Inverno.jpg***. A imagem deve aparecer na página. Se você observar o código, verá que uma coisa o programa fez "bonito": inseriu um parágrafo para colocar a imagem dentro. por outro lado, a tag da imagem foi:

```
<img border="0" src="file:///C:/Documents%20and%20Settings/All%20Users/Documentos/Minhas
%20imagens/Amostras%20de%20imagens/Inverno.jpg" width="800" height="600">
```

Observe que o FrontPage usou tags adequadas para width e height, mas usou a tag depreciada *border*. A borda deve ser definida na folha de estilo, e não no documento HTML. Adicionalmente, o FrontPage inseriu o nome completo da imagem, algo que era necessário neste caso (pois a imagem estava em um local bastante distinto da página web), mas nenhum aviso foi feito de que este link **não** irá funcionar se colocado na Web, e que o FrontPage copiará este arquivo para o mesmo diretório onde está o arquivo HTML, ignorando os critérios de organização de arquivos explicador em aula. Mas, mais uma vez, como o objetivo é apenas gerar um layout, não há problemas.

Agora, clique com o botão direito na imagem e selecione "Propriedades da Imagem". Isso abrirá uma janela.

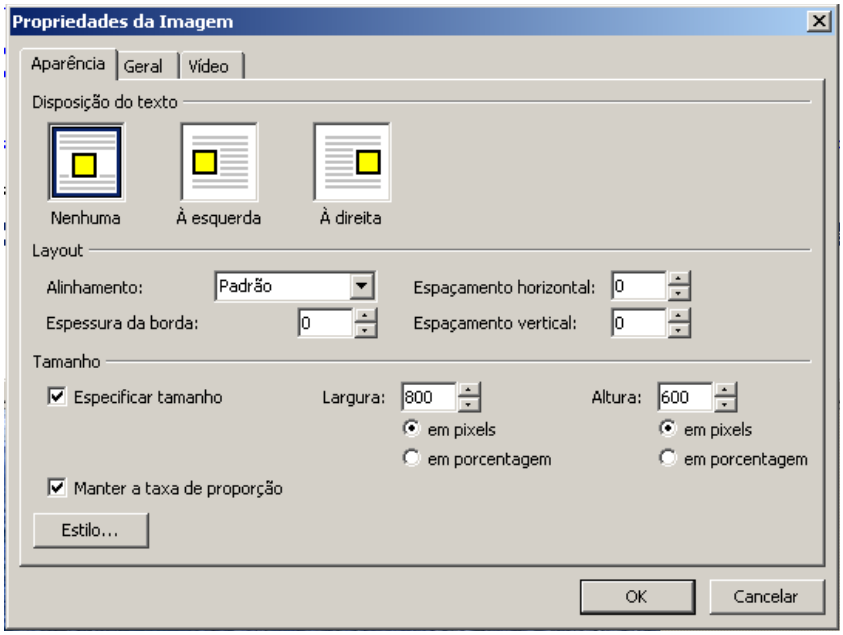


Figura 7: Configuração da aparência da imagem

A aba "Aparência" permite configurar alguns elementos de interesse. O primeiro é a "Disposição do Texto", que indica se a figura estará flutuando à esquerda, direita ou se ela estará interrompendo o texto. A parte de layout indica como se dá o alinhamento do texto e os espaçamentos ao redor da figurar. Tudo isso será definido na tag HTML, com parâmetros depreciados, e não no CSS como seria adequado. Mas serve para o layout.

Finalmente, é possível especificar o tamanho da figura, em pixels ou em porcentagem, opção essa que usa os parâmetros WIDTH e HEIGHT da tag IMG, que são corretos e podem ser usados normalmente. Clique agora na aba "Geral".

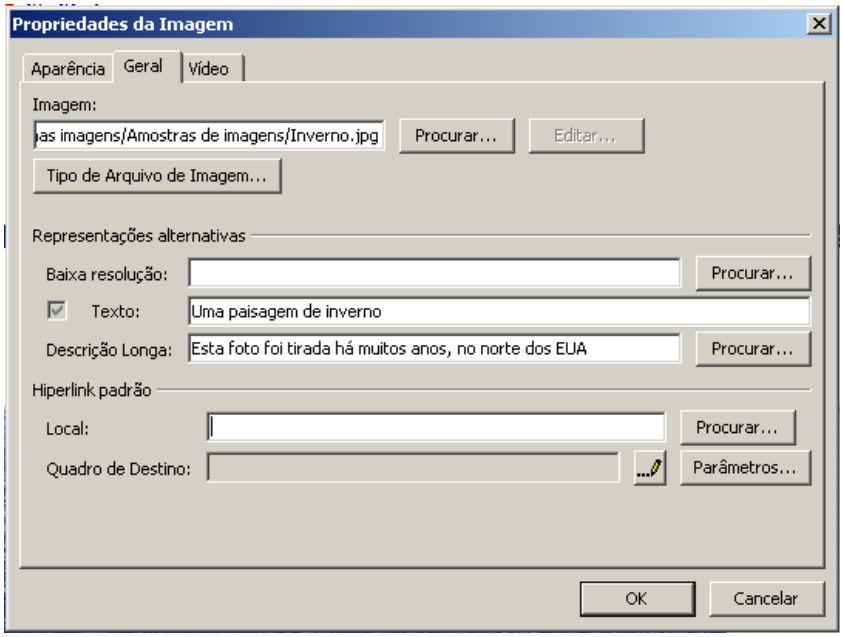


Figura 8: Configurações gerais da imagem

Nesta aba é possível definir uma imagem alternativa de baixa resolução, que será especificada de uma maneira fora do padrão, a mensagem "Texto:" será especificada como "ALT", que é o texto que o Internet Explorer exibe como "balão de ajuda" na ausência da definição de TITLE, a descrição longa também é especificada de uma maneira fora do padrão.

Finalmente, a opção "Local" permite indicar um link (para transformar a imagem automaticamente em um link) e "Quadro Destino" permite indicar um frame para o carregamento da página apontada pelo link. A aba "Vídeo" não será tratada neste tutorial. Assim, clique em Ok e veja os efeitos das mudanças feitas.

A útlima parte deste tutorial será a criação de uma tabela, bastante útil para criar layouts em um ambiente de edição limitado. Para dar início, crie um novo documento (**Arquivo > Novo** e depois clique em "Documento em Branco" no lado direito da página) para que possamos criar um layout baseado em tabela.

Antes de mais nada, é preciso imaginar o layout que se deseja para a página. Vamos adotar o layout abaixo:

<table>
<tr><td colspan="2">Logotipo</td></tr>
<tr><td>M
E
N
U</td><td>Área de

Conteúdo</td></tr>
</table>

Esse desenho é muito importante quando se vai desenhar um layout por tabelas, pois precisamos verificar duas coisas:

a) O maior número de colunas necessário
b) O maior número de linhas necessário

No caso, esse layout pode ser composto em uma tabela com 2 linhas e duas colunas, sendo que as duas células da primeira linha devem ser mescladas. O primeiro passo para criar este layout será criar a tabela. Isso pode ser feito usando a opção do menu: **Tabela > Inserir > Tabela**. Neste momento, aparecerá uma janela, que deve ser configurada como especificado abaixo:

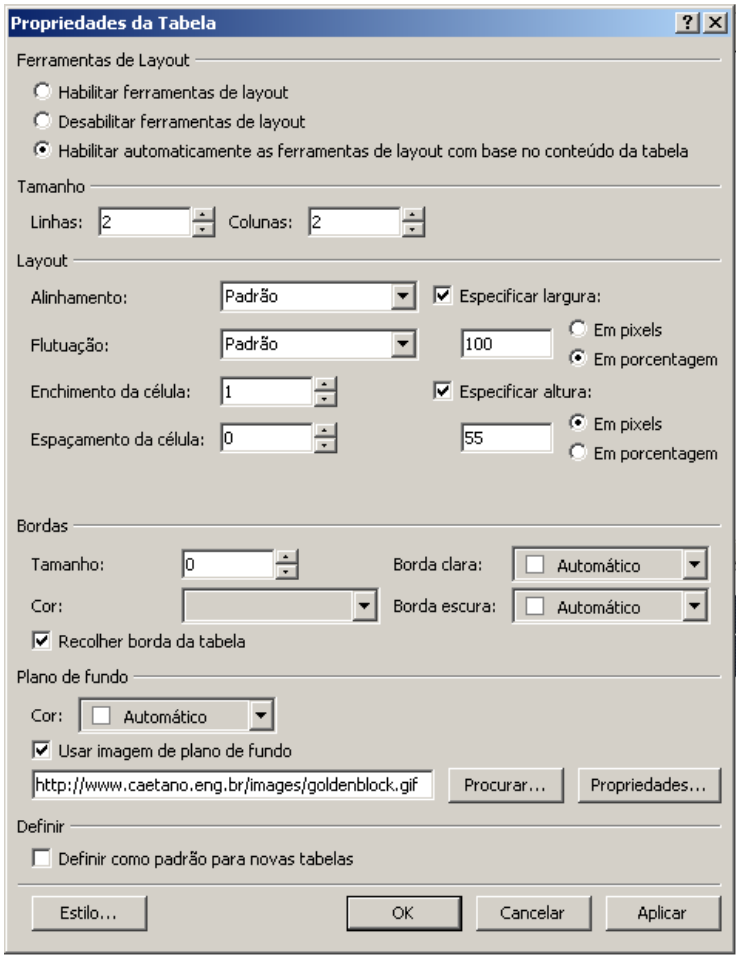


Figura 9: Janela de criação de tabelas

A primeira parte permite que configuremos uma tabela automática ou dimensionada manualmente. Normalmente não precisamos alterar isso. A parte "Tamanho" permite definir o número de linhas e colunas que a tabela terá.

Na parte "Layout" define-se o alinhamento das células, se a largura da tabela será especificada (se não for, fica automática). No caso a largura foi especificada em 100% da tela. A flutuação define se a tabela vai estar à esquerda, à direita (em ambos os casos com o texto

"escorrendo" pelo outro lado), ou se a tabela vai estar dividindo o texto, que parará no topo da tabela e continuará na parte inferior da tabela. O "Enchimento da Célula" é o número de pixels que se deseja entre a borda da tabela e o conteúdo. Use no mínimo 1, para que o conteúdo não fique tão grudado na beirada. O "Espaçamento da Célula" é o espaço entre as células. Como se trata de uma tabela de layout, o ideal é definir isso como 0, para que uma célula fique grudada em outra. É possível, ainda, especificar a altura da tabela, mas em geral esta é deixada como automática (de acordo com o conteúdo).

Na parte "Bordas", por se tratar de uma tabela de layout, não desejamos bordas, então basta marcar "Recolher Borda da Tabela" e definir seu tamanho para 0. Se desejássemos uma borda, aqui seria o local de configurá-la.

A parte "Plano de Fundo" serve para definir o fundo da tabela. Se desejar, marque "Usar imagem de fundo" e coloque, por exemplo, o link "**http://www.caetano.eng.br/images/goldenblock.gif**" para que o fundo da tabela fique com o amarelo que usamos anteriormente. Após isso, clique no botão "Ok" e a tabela aparecerá, sem formatação alguma, isto é, com quatro células iguais:

Figura 10: Tabela não formatada

Marque as duas células da esquerda, clique com o botão direito na região marcada, e selecione a opção "Propriedades da Célula". A seguinte janela deve aparecer:

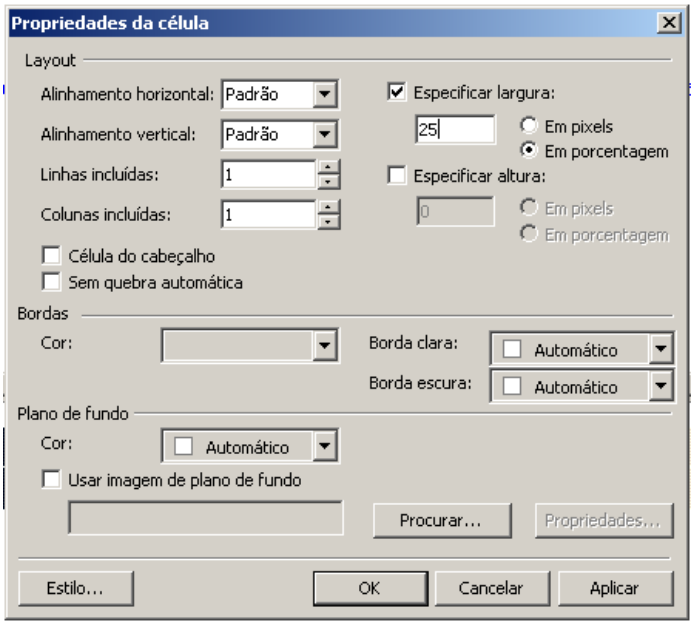


Figura 11: Janela de configuração de células

Marque "Especificar Largura" e defina a mesma para 25%. Clique em "Aplicar" e "Ok" e observe as mudanças tendo efeito.

Marque agora as duas células superiores. Clique com o botão direito na região marcada, e selecione a opção "Mesclar Células". Pronto. O primeiro passo está concluído: a janela já tem a configuração que desejamos.

Agora vamos inserir o logotipo na região superior. Para isso, selecione a célula superior e use os menus para adicionar uma imagem (**Inserir > Imagem > Do Arquivo**). Insira qualquer imagem e, depois, clique com o botão direito nela, selecionando "Propriedades da Imagem". Clique na aba "Geral" e mude o nome da imagem para "**http://www.caetano.eng.br/main/images/aflogo_horiz_transp.gif**", indicando o Texto ALT como **AF Logo**.

Selecione a célula onde está a imagem e clique no botão para centralizar texto. Selecione agora a célula do menu e insira um hiperlink para a página http://www.caetano.eng.br/, conforme indicado na janela abaixo:

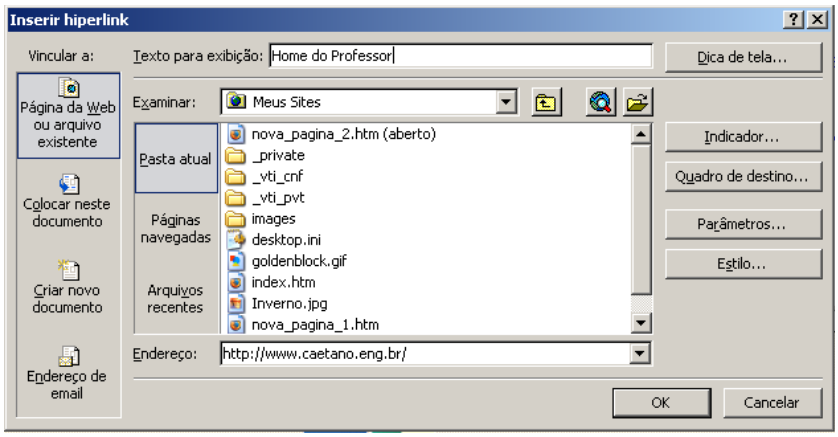


Figura 12: Adição de Hiperlink

Adicione um hiperlink direto para a página da disciplina (http://www.caetano.eng.br/aulas/radial/web/) e outro para o UOL (http://www.uol.com.br). Selecione todo o conteúdo desta célula e clique no botão para centralizar o conteúdo.

Vá agora para a célula onde ficará o conteúdo da página e digite um título como "**Home Page Teste com FrontPage**". Digite isso em texto normal, depois selecione este texto e, no menu de estilos, defina como "**Título 1**" e depois centralize o texto. Adicione um texto qualquer abaixo. Pode formatá-lo à vontade. O resultado deve ter o aspecto apresentado na figura 13.

Figura 13: Aspecto da página construída

Teste a página no navegador real, usando a opção "**Arquivo > Visualizar no Navegador**". Depois que tudo estiver testado, está na hora de publicá-lo. Isso pode ser feito pelo meno "**Arquivo > Publicar Site**".

Como o servidor web está em nossa máquina, publicaremos no sistema de arquivos, no local de compartilhamento web do nosso servidor, que deve ser algo como **C:\WAMP\WWW\PROTO\.** Para isso, configuraremos a janela "Publicar Site" da seguinte forma:

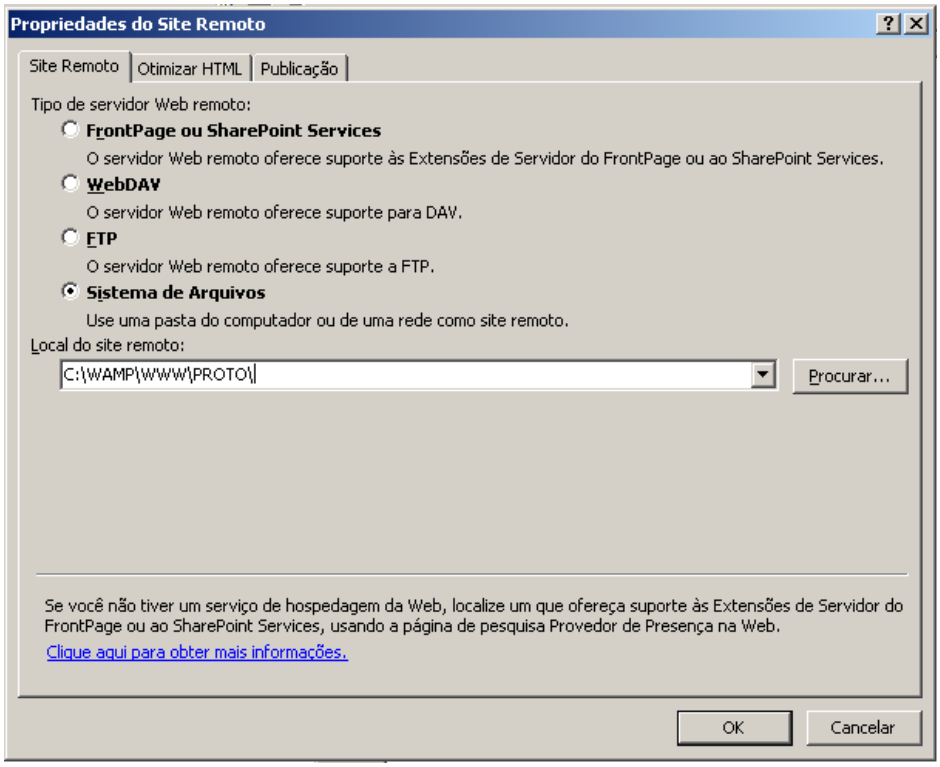


Figura 14: Janela de publicação do site

Depois de publicá-lo, teste-o com os navegadores, navegando pelo http://localhost/PROTO/nome_do_arquivo.html. Lembre-se, no entanto, que isso se trata apenas de um **protótipo**. O código gerado pelo FrontPage não pode ser, de forma alguma, considerado "de produção".

Você pode conferir as páginas publicadas no site:
http://www.caetano.eng.br/proto/nova_pagina_1.htm
http://www.caetano.eng.br/proto/nova_pagina_2.htm

3. ATIVIDADE

1. Faça no FrontPage um dos três protótipos originais criados por você para o Projeto Integrado Web.

Unidade 11: Formulários HTML

Prof. Daniel Caetano

Objetivo: Apresentar as tags de formulário e sua manipulação.

INTRODUÇÃO

Conceitos Chave:

- Linguagens de Programação
  - * Parâmetros de trabalho?
    - + Fornecidos pelo usuário!
- Como fornecer estes dados?
  - * Pelo endereço (Método GET)
  - * Por formulários (Método POST)
- Três Passos:
  - * Criar Formulário
  - * Validar Formulário
  - * Processar Formulário

Após esta aula, nos dedicaremos ao estudo de algumas linguagens de programação que podem ser associadas à Web. Entretanto, na Web, as linguagens de programação acabam ficando relativamente limitadas se o usuário não puder fornecer informações, ou seja, se o usuário não puder fornecer dados.

Para que o usuário possa fornecer dados, há duas maneiras: uma seria digitando parâmetros na linha de endereços (algo que veremos no futuro), em um sistema análogo à uma "linha de comandos" (Método GET). A outra maneira é enviar os dados por meio de um arquivo (Método POST). Esta segunda forma é mais elegante e é a mais usada.

Para que o usuário digite estes dados, é comum o uso de formulários. Os formulários são bastante flexíveis, mas exigem uma certa seqüência de passos:

a) Criar o formulário
b) Validar o formulário no lado do cliente
c) Processar os dados recebidos pelo formulário

Nesta aula será visto o primeiro passo e, nas próximas, serão apresentados os outros passos.

1. O QUE SÃO FORMULÁRIOS

Conceitos Chave:
- Tag <FORM>...</FORM>
- Parâmetros
  * ACTION = "pagina_destino"
  * METHOD = "post"
  * NAME = "form1"
  * ID = "form1"
- Formulário tem nome!
- Campos também 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 "nomes" aos formulários, usando o parâmetro NAME. Por exemplo:

```
<FORM ACTION="pagina.php" METHOD="post" NAME="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.

2. 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 a tag <INPUT>, uma tag que não exige fechamento (</INPUT>), 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. A utilidade disso surge apenas quando associarmos os formulários ao uso de javascript.

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

2.1. 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">
```

2.2. 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).

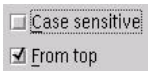
Finalmente, há o parâmetro READONLY, que indica que o texto de um campo não pode ser modificado.

2.3. 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]>
```

Onde o "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:



2.4. 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:

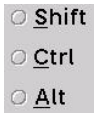
```
<INPUT TYPE="radio" NAME="sexo" ID="fem" VALUE="fem" [CHECKED] >
<INPUT TYPE="radio" NAME="sexo" ID="masc" VALUE="masc">
```

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:

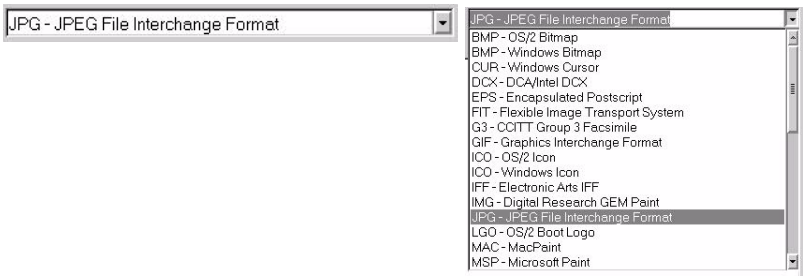
```
<INPUT TYPE="radio" NAME="key" ID="shf" VALUE="0" ACCESSKEY="S" />Shift<BR />
<INPUT TYPE="radio" NAME="key" ID="ctr" VALUE="1" ACCESSKEY="C" />Ctrl<BR />
<INPUT TYPE="radio" NAME="key" ID="alt" VALUE="2" ACCESSKEY="A" />Alt
```

Isso seria apresentado de maneira similar à indicada abaixo:



2.5. 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. Podemos ter uma caixa de seleção múltipla com o parâmetro 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 dentro da tag <OPTION> correspondente.

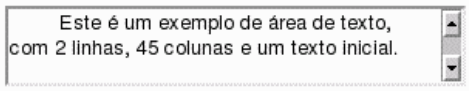
2.6. Área de Texto

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.

2.7. 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 HTML, 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:

2.8. 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">
```

2.9. 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>
```

3. OUTRAS TAGS PARA FORMULÁRIOS (OPCIONAL)

Conceitos Chave:
- <LABEL FOR="id">
- <BUTON 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.

3.1. 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.

3.2. 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 HTML dentro deste botão. Exemplo:

```
<BUTTON TYPE="button">
    <TABLE>
        <TR><TD>
        <IMG SRC="http://www.caetano.eng.br/main/images/aflogo_horiz.gif">
        </TD><TD>
        <P>Um texto qualquer</P>
        </TD>
    </TABLE>
</BUTTON>
```

O que será apresentado da seguinte forma:

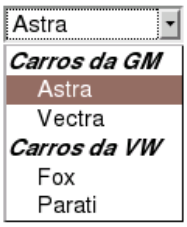
Observe a flexibilidade que esta tag proporciona, por permitir que praticamente se crie elementos HTML dentro do botão e que, obviamente, podem ser configurados com o uso de CSS.

3.3. 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:



3.4. 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, 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">
```

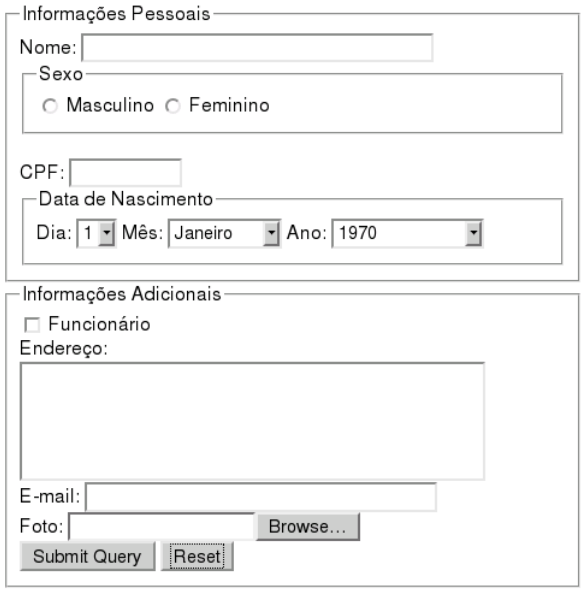
4. TUTORIAL

Crie um formulário de cadastro de usuário, contendo as seguintes informações:

Grupo: Informações Pessoais
Nome: texto, até 40 caracteres
Sexo: caixa de opção (radiobox) Masculino x Feminino
CPF: texto, até 11 caracteres
Dia de Nascimento: lista de seleção (combobox) de 0 a 31
Mês de Nascimento: lista de seleção (combobox) de Janeiro a Dezembro
Ano de nascimento: lista de seleção (combobox) de 1980 a 2000

Grupo: Informações Adicionais
Funcionário: caixa de seleção (checkbox)
Endereço: área de texto, até 1024 caracteres
E-mail: texto, até 40 caracteres
Foto: enviar arquivo

O botão deve ter um botão "enviar", um botão "apagar" e, quando os dados forem enviados, devem ser enviados para o arquivo "processa.php". A aparência do formulário deve ser a indicada abaixo:



5. BIBLIOGRAFIA

W3 schools - CSS Tutorial. Disponível em < http://www.w3schools.com/ >. Visitado em 10/03/2009.
MUTO, C.A. PHP & MySQL: Guia Introdutório. Rio de Janeiro: Brasport, 2006.
RATSCHILLER, T; GERKEN, T; Desenvolvendo aplicações Web com PHP 4.0. Ed. Ciência Moderna, 2000.

Unidade 12: 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 sobre a construção de páginas web, as páginas construídas até o momento são puramente estáticas, isto é, sem elementos que possam ser alterados automaticamente.

Foi visto anteriormente, por exemplo, como era possível construir formulários e como enviá-los ao servidor. Entretanto, ainda não foi visto como validar este formulário antes de enviá-lo, e nem como processar este formulário depois que ele tenha sido enviado.

Para realizar estas duas tarefas, serão usadas duas linguagens bastante diferentes: JavaScript e PHP. Embora sejam diferentes, estas linguagens compartilham uma característica importante: elas são interpretadas.

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 HTML), 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	Sun	Interpretada	Java/C	Cliente
PHP	opensource	Interpretada	C/Java	Servidor
ASP	Microsoft	Interpretada	BASIC	Servidor
JSP	Sun	Interpretada	Java/C	Servidor
Java Servlets	Sun	"Compilada"	Java	Servidor
ASP .Net	Microsoft	Compilada	BASIC .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:

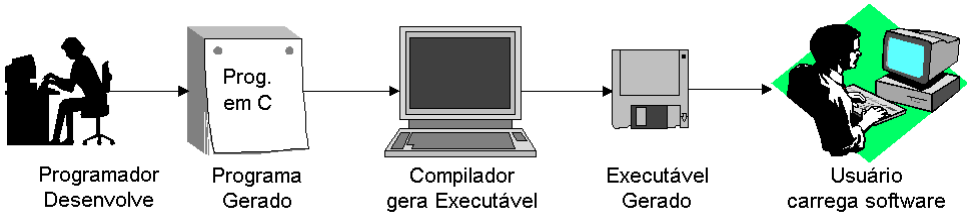


Figura 1: Processo de Criação à Execução com uso de Linguagens Compiladas

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:

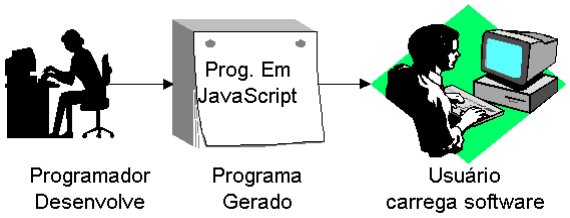


Figura 1: Processo de Criação à Execução com uso de Linguagens Compiladas

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.

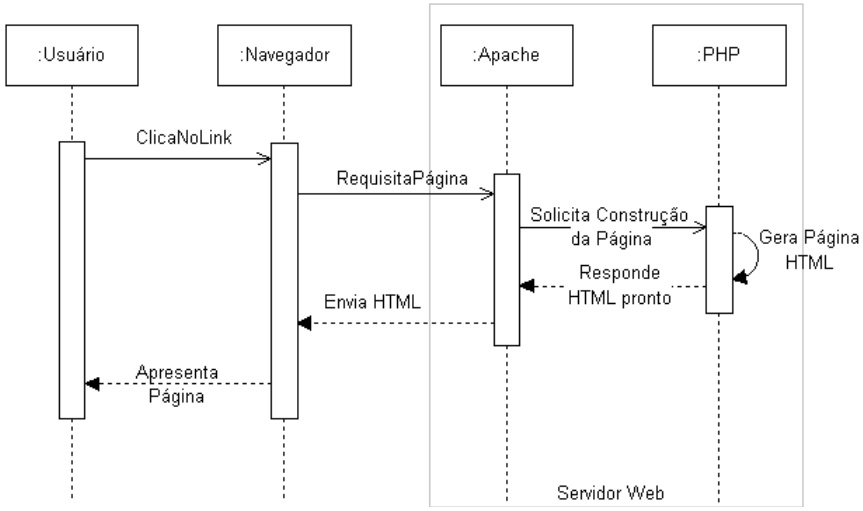


Figura 3: Linguagens processadas "no lado do servidor"

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.

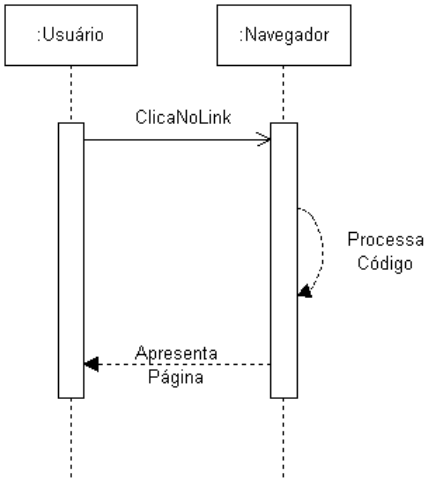


Figura 4: Linguagens processadas "no lado do cliente"

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

```
<HTML LANG="pt-BR">
<HEAD>
	<TITLE>Teste de JavaScript</TITLE>
</HEAD>
<BODY>
	<INPUT TYPE="button" VALUE="Cor"  />
</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

> Nota: Quase tudo dentro de um documento HTML pode ser acessado usando a estrutura: **document.*nome_do_elemento*.proprietade**. *Nome_do_elemento* pode ser definido em praticamente todas as tags usando o parâmetro NAME="nome_do_elemento". No caso da tag body, em especial, ela pode ser acessada diretamente, sem a necessidade de nomes, porque ela é sempre única no documento.

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 HTML, para que ele possa ser usado. Isso pode ser feito conforme indicado a seguir, com a tag SCRIPT:

teste.html

```
<HTML LANG="pt-BR">
<HEAD>
	<TITLE>Teste de JavaScript</TITLE>
	<SCRIPT TYPE="text/javascript" SRC="efeitos.js"></SCRIPT>
</HEAD>
<BODY>
	<INPUT TYPE="button" VALUE="Cor"  />
</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

```
<HTML LANG="pt-BR">
<HEAD>
	<TITLE>Teste de JavaScript</TITLE>
	<SCRIPT TYPE="text/javascript" SRC="efeitos.js"></SCRIPT>
</HEAD>
<BODY>
	<INPUT TYPE="button" VALUE="Cor"  ID="bmudacor" />
</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:

```
<HTML LANG="pt-BR"><HEAD>
    <TITLE>Teste de JavaScript</TITLE>
    <P ID="ajuda">Aqui aparece o help!</P>
    <SCRIPT TYPE="text/javascript" SRC="efeitos.js"></SCRIPT>
</HEAD>
<BODY>
    <INPUT TYPE="button" ID="bmudacor" VALUE="Cor">
</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.

Uma das formas de acessar estes eventos é 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;
```

Este "caminho" de nomes nem sempre é fácil identificar. Note que se o <FORM> não tivesse sido definido, o caminho **não** poderia ser document.dado.onchange... Como resolver este dilema?

Uma outra forma de acessar os elementos, para evitar a dor de cabeça descrita anteriormente, é pedindo para o documento (HTML) encontrar um elemento qualquer, usando seu ID como chave de busca (parâmetro ID na tag HTML). 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:

Acessando diretamente o elemento:

```
document.continente.nome.style.estilo = valor;
```

Exemplo:

.html

```
<FORM NAME="form1">
        <INPUT TYPE="text" NAME="dado">
</FORM>
```

.js

```
document.form1.dado.style.backgroundColor = "black";
```

Acessando indiretamente o elemento:

```
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 13: Validação de Formulários com Javascript

Prof. Daniel Caetano

Objetivo: Capacitar o aluno para o uso de javascript na validação de daods client-side.

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

INTRODUÇÃO

Conceitos Chave:
- JavaScript...
  * Como usar isso para algo útil?
  * Validação de Formulários Client-Side!
- Vantagens de validação 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: validar dados digitados pelo usuário ainda no navegador, isto é, 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.

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. PREPARANDO O 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 CPF 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 o CPF pode ter até 11 caracteres (campo de tamanho 12). A "ação" do formulário deve carregar o documento "**cadastro.php**". O formulário terá a seguinte "cara":

cadastro.html

```
<HTML LANG="pt-BR">
<HEAD>
  <TITLE>Valida&ccedil;&atilde;o de Formul&aacute;rio</TITLE>
</HEAD>
<BODY>
  <FORM METHOD="POST" ACTION="cadastro.php" ID="formCadastro">
    <LABEL FOR="nome">Nome:</LABEL>
    <INPUT TYPE="text" SIZE="30" MAXLENGTH="255" ID="nome"><BR>
    <LABEL FOR="idade">Idade:</LABEL>
    <INPUT TYPE="text" SIZE="4" MAXLENGTH="3" ID="idade"><BR>
    <LABEL FOR="cpf">CPF:</LABEL>
    <INPUT TYPE="text" SIZE="12" MAXLENGTH="11" ID="cpf"><BR>
    <INPUT TYPE="submit" VALUE="Ok" ID="ok">
  </FORM>
</BODY>
</HTML>
```

Quando manipulamos dados de formulários, apesar de todos os métodos descritos anteriormente funcionarem, é comum darmos um nome para o formulário e 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

```
<HTML LANG="pt-BR">
<HEAD>
  <TITLE>Valida&ccedil;&atilde;o de Formul&aacute;rio</TITLE>
</HEAD>
<BODY>
 <FORM METHOD="POST" ACTION="cadastro.php" ID="formCadastro" NAME="formCadastro">
    <LABEL FOR="nome">Nome:</LABEL>
    <INPUT TYPE="text" SIZE="30" MAXLENGTH="255" ID="nome" NAME="nome"><BR>
    <LABEL FOR="idade">Idade:</LABEL>
    <INPUT TYPE="text" SIZE="4" MAXLENGTH="3" ID="idade" NAME="idade"><BR>
    <LABEL FOR="cpf">CPF:</LABEL>
    <INPUT TYPE="text" SIZE="12" MAXLENGTH="11" ID="cpf" NAME="cpf"><BR>
    <INPUT TYPE="submit" VALUE="Ok" ID="ok">
  </FORM>
</BODY>
</HTML>
```

Isso nos permitirá acessar os dados diretamente. Por exemplo: para acessar o texto do nome, bastará indicar: *document.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

```
<HTML LANG="pt-BR">
<HEAD>
  <TITLE>Valida&ccedil;&atilde;o de Formul&aacute;rio</TITLE>
  <SCRIPT TYPE="text/javascript" SRC="cadastro.js"></SCRIPT>
</HEAD>
<BODY>
  <FORM METHOD="POST" ACTION="cadastro.php" ID="formCadastro" NAME="formCadastro">
    <LABEL FOR="nome">Nome:</LABEL>
    <INPUT TYPE="text" SIZE="30" MAXLENGTH="255" ID="nome" NAME="nome"><BR>
    <LABEL FOR="idade">Idade:</LABEL>
    <INPUT TYPE="text" SIZE="4" MAXLENGTH="3" ID="idade" NAME="idade"><BR>
    <LABEL FOR="cpf">CPF:</LABEL>
    <INPUT TYPE="text" SIZE="12" MAXLENGTH="11" ID="cpf" NAME="cpf"><BR>
    <INPUT TYPE="submit" VALUE="Ok" ID="ok">
  </FORM>
</BODY>
</HTML>
```

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 */
function configura() {
  }
```

Precisamos associar esta função ao evento "window.onload", para que ela seja chamada assim que o carregamento da página termine. Isso é feito como indicado abaixo:

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) O CPF precisa ser um número
g) O CPF precisa ser válido

3. VALIDANDO O FORMULÁRIO

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

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

Convém indicar o que esta trecho faz, com um comentário:

```
/* Verifica se nome está preenchido */
if (document.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 (document.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() {
        /* Verifica se nome está preenchido */
        if (document.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() {

      /* Verifica se nome está preenchido */
      if (document.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() {

      /* Verifica se nome está preenchido */
      if (document.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. 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 (document.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() {
        /* Verifica se nome está preenchido */
        if (document.formCadastro.nome.value.length == 0) {
          informarErro("Você precisa digitar um nome!");
          return false;
          }
        /* Verifica se nome tem, no mínimo, 6 caracteres  */
        if (document.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().

3.3. 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( document.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() {
        /* Verifica se nome está preenchido */
        if (document.formCadastro.nome.value.length == 0) {
          informarErro("Você precisa digitar um nome!");
          return false;
          }
        /* Verifica se nome tem, no mínimo, 6 caracteres  */
        if (document.formCadastro.nome.value.length < 6) {
          informarErro("Nome muito curto!");
          return false;
          }
        /* Verifica se idade é um número  */
        if ( isNaN( document.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);
  }
```

3.4. 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 (document.formCadastro.idade.value < 18) {
  informarErro("A idade mínima é 18 anos!");
  return false;
  }
```

3.5. 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 (document.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!).

3.6. Se CPF é um número

O teste aqui é análogo ao da idade. Caso não seja um número, o erro deve ser "CPF precisa ser um número!":

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

3.7. Validação do CPF

Finalmente chega-se à verificação mais complicada. Por ser uma comparação mais complexa, iremos criar uma função específica "checaCPF()", que deve retornar "true" se o CPF for válido ou "false" se for inválido.

Considerando isso, o teste pode ser resumido, indicando o erro "CPF inválido!", caso o mesmo esteja incorreto:

```
/* Verifica se CPF é válido  */
if ( checaCPF( document.formCadastro.cpf.value ) == false ) {
  informarErro("CPF inválido!");
  return false;
  }
```

Isso pode ser acrescentado na função validar logo após a verificação de que o CPF é um número (o teste não faz muito sentido se o dado não for um número!).

Entretanto, ainda falta o código para chegar o CPF. Por se tratar de um algoritmo um tanto complexo, o código será fornecido:

```
/* Função auxiliar usada para validar um CPF */
/* Original em: http://www.fundao.wiki.br/articles.asp?cod=23 */
/* Comentários adicionados pelo Prof. Daniel Caetano */
function checaCPF(CPF) {
  /* Alguns CPFs são facilmente invalidados... */
  if (CPF.length != 11 || CPF == "00000000000" || CPF == "11111111111" ||
    CPF == "22222222222" ||      CPF == "33333333333" || CPF == "44444444444" ||
    CPF == "55555555555" || CPF == "66666666666" || CPF == "77777777777" ||
    CPF == "88888888888" || CPF == "99999999999")
      return false;
  /* Incia-se cálculo do primeiro dígito, zerando a soma */
  soma = 0;
  /* Calcula-se a soma d1*10 + d2*9 + ... + d9*2 */
  for (i=0; i < 9; i ++)
    soma += parseInt(CPF.charAt(i)) * (10 - i);
  /* Calcula-se "11 - resto da divisão por 11" */
  resto = 11 - (soma % 11);
  /* Ajuste */
  if (resto == 10 || resto == 11)
    resto = 0;
  /* Verifica se primeiro dígito bate */
  if (resto != parseInt(CPF.charAt(9)))
    return false;

  /* Incia-se cálculo do segundo dígito, zerando a soma */
  soma = 0;
  /* Calcula-se a soma d1*11 + d2*10 + ... + d10*2 */
  for (i = 0; i < 10; i ++)
    soma += parseInt(CPF.charAt(i)) * (11 - i);
```

```
  /* Calcula-se "11 - resto da divisão por 11" */
  resto = 11 - (soma % 11);
  /* Ajuste */
  if (resto == 10 || resto == 11)
    resto = 0;
  /* Verifica se segundo dígito bate */
  if (resto != parseInt(CPF.charAt(10)))
    return false;
  return true;
  }
```

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() {
        /* Verifica se nome está preenchido */
        if (document.formCadastro.nome.value.length == 0) {
          informarErro("Você precisa digitar um nome!");
          return false;
          }
        /* Verifica se nome tem, no mínimo, 6 caracteres  */
        if (document.formCadastro.nome.value.length < 6) {
          informarErro("Nome muito curto!");
          return false;
          }
        /* Verifica se idade é um número  */
        if ( isNaN( document.formCadastro.idade.value ) ) {
          informarErro("A idade precisa ser um número!");
          return false;
          }
        /* Verifica se idade é menor que 18 anos  */
        if (document.formCadastro.idade.value < 18) {
          informarErro("A idade mínima é 18 anos!");
          return false;
          }
        /* Verifica se idade é maior que 150 anos  */
        if (document.formCadastro.idade.value > 150) {
          informarErro("Ninguém vive tanto tempo!");
          return false;
          }
        /* Verifica se CPF é um número  */
        if ( isNaN( document.formCadastro.cpf.value ) ) {
          informarErro("CPF precisa ser um número!");
          return false;
```

```
            }
          /* Verifica se CPF é válido  */
          if ( checaCPF( document.formCadastro.cpf.value ) == false ) {
            informarErro("CPF inválido!");
            return false;
            }

  return true;
  }

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

/* Função auxiliar usada para validar um CPF */
/* Original em: http://www.fundao.wiki.br/articles.asp?cod=23 */
/* Comentários adicionados pelo Prof. Daniel Caetano */
function checaCPF(CPF) {
  /* Alguns CPFs são facilmente invalidados... */
  if (CPF.length != 11 || CPF == "00000000000" || CPF == "11111111111" ||
     CPF == "22222222222" ||       CPF == "33333333333" || CPF == "44444444444" ||
     CPF == "55555555555" || CPF == "66666666666" || CPF == "77777777777" ||
     CPF == "88888888888" || CPF == "99999999999")
       return false;
  /* Incia-se cálculo do primeiro dígito, zerando a soma */
  soma = 0;
  /* Calcula-se a soma d1*10 + d2*9 + ... + d9*2 */
  for (i=0; i < 9; i ++)
    soma += parseInt(CPF.charAt(i)) * (10 - i);
  /* Calcula-se "11 - resto da divisão por 11" */
  resto = 11 - (soma % 11);
  /* Ajuste */
  if (resto == 10 || resto == 11)
    resto = 0;
  /* Verifica se primeiro dígito bate */
  if (resto != parseInt(CPF.charAt(9)))
    return false;

  /* Incia-se cálculo do segundo dígito, zerando a soma */
  soma = 0;
  /* Calcula-se a soma d1*11 + d2*10 + ... + d10*2 */
  for (i = 0; i < 10; i ++)
    soma += parseInt(CPF.charAt(i)) * (11 - i);
  /* Calcula-se "11 - resto da divisão por 11" */
  resto = 11 - (soma % 11);
  /* Ajuste */
  if (resto == 10 || resto == 11)
    resto = 0;
  /* Verifica se segundo dígito bate */
  if (resto != parseInt(CPF.charAt(10)))
    return false;
  return true;
  }
```

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

```
<HTML LANG="pt-BR">
<HEAD>
  <TITLE>Valida&ccedil;&atilde;o de Formul&aacute;rio</TITLE>
  <SCRIPT TYPE="text/javascript" SRC="cadastro.js"></SCRIPT>
</HEAD>
<BODY>
  <P>Preencha o formul&aacute;rio abaixo:</P>
  <P ID="erros"></P>
  <FORM METHOD="POST" ACTION="cadastro.php" ID="formCadastro" NAME="formCadastro">
    <LABEL FOR="nome">Nome:</LABEL>
    <INPUT TYPE="text" SIZE="30" MAXLENGTH="255" ID="nome" NAME="nome"><BR>
    <LABEL FOR="idade">Idade:</LABEL>
    <INPUT TYPE="text" SIZE="4" MAXLENGTH="3" ID="idade" NAME="idade"><BR>
    <LABEL FOR="cpf">CPF:</LABEL>
    <INPUT TYPE="text" SIZE="12" MAXLENGTH="11" ID="cpf" NAME="cpf"><BR>
    <INPUT TYPE="submit" VALUE="Ok" ID="ok">
  </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! O código final será:

cadastro.js

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

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

```
        /* Verifica se CPF é válido  */
        if ( checaCPF( document.formCadastro.cpf.value ) == false ) {
          informarErro("CPF inválido!");
          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;
  }

/* Função auxiliar usada para validar um CPF */
/* Original em: http://www.fundao.wiki.br/articles.asp?cod=23 */
/* Comentários adicionados pelo Prof. Daniel Caetano */
function checaCPF(CPF) {
  /* Alguns CPFs são facilmente invalidados... */
  if (CPF.length != 11 || CPF == "00000000000" || CPF == "11111111111" ||
    CPF == "22222222222" ||      CPF == "33333333333" || CPF == "44444444444" ||
    CPF == "55555555555" || CPF == "66666666666" || CPF == "77777777777" ||
    CPF == "88888888888" || CPF == "99999999999")
      return false;
  /* Incia-se cálculo do primeiro dígito, zerando a soma */
  soma = 0;
  /* Calcula-se a soma d1*10 + d2*9 + ... + d9*2 */
  for (i=0; i < 9; i ++)
    soma += parseInt(CPF.charAt(i)) * (10 - i);
  /* Calcula-se "11 - resto da divisão por 11" */
  resto = 11 - (soma % 11);
  /* Ajuste */
  if (resto == 10 || resto == 11)
    resto = 0;
  /* Verifica se primeiro dígito bate */
  if (resto != parseInt(CPF.charAt(9)))
    return false;

  /* Incia-se cálculo do segundo dígito, zerando a soma */
  soma = 0;
  /* Calcula-se a soma d1*11 + d2*10 + ... + d10*2 */
  for (i = 0; i < 10; i ++)
    soma += parseInt(CPF.charAt(i)) * (11 - i);
  /* Calcula-se "11 - resto da divisão por 11" */
  resto = 11 - (soma % 11);
  /* Ajuste */
  if (resto == 10 || resto == 11)
    resto = 0;
  /* Verifica se segundo dígito bate */
  if (resto != parseInt(CPF.charAt(10)))
    return false;
  return true;
  }
```

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 14: Linguagens Interpretadas: PHP

Prof. Daniel Caetano

Objetivo: Apresentar uma introdução ao PHP, sua integração com a página Web, a operação com suas variáveis e o uso de suas funções mais básicas.

Bibliografia: MUTO,2006; RATSCHILLER,2000.

INTRODUÇÃO

Conceitos Chave:
- Como processar os dados enviados pelo usuário?
  * Uso de uma linguagem server-side
  * Linguagem PHP (interpretada)

Apesar de tudo que já foi apresentado sobre a criação de páginas e formulários, isso não foi suficiente para implementarmos um sistema completo, isto é, que processe informações enviadas pelo usuário e as armazene, por exemplo, em um banco de dados.

Para fazer isso, é necessário operar com uma linguagem "server-side", isto é, cujo trabalho ocorra no servidor web e não no navegador. Para isso usaremos a linguagem PHP, uma das mais poderosas para este tipo de função, sendo ela também uma linguagem interpretada e que é executada no lado do servidor.

1. LINGUAGEM INTERPRETADA SERVER SIDE?

Conceitos Chave:
- Linguagem Interpretada
  * Precisa de um software interpretador (computador não a entende sozinho)
  * Interpretador PHP
    + Executável Isolado x DLL
- Vantagens:
  * Praticidade
  * Compatibilidade
  * Rapidez de desenvolvimento
- "Vantagem": código junto com o HTML

Recordando, a linguagem interpretada é aquela que não precisamos traduzir previamente o conteúdo para a linguagem do computador porque haverá, no computador, um outro programa - o interpretador - que fará a *tradução simultânea*.

No caso do PHP, o "interpretador" é um módulo instalado no servidor web, que é justamente responsável por realizar esta tradução. Este módulo é, sem criatividade alguma, chamado de "módulo interpretador de PHP". Ele existe no formato de biblioteca (MODPHP.DLL) para alguns servidores Web, mas em outros é simplesmente um executável (PHP.EXE)

O PHP tem todas as vantagens de ser uma linguagem server-side já vistas anteriormente, como: praticidade de debug, rapidez de desenvolvimento (por não precisar de compilação), alta compatibilidade com servidores web e diferentes plataformas (existe interpretador PHP para todos os computadores e sistemas operacionais modernos - e até para alguns mais antigos), é compatível com todos os tipos os clientes web (o navegador nem sabe da existência do PHP), dentre outras.

Adicionalmente, o PHP tem uma grande vantagem para seu aprendizado inicial: para seu uso em páginas web, por ter sido criado para isso e por trabalhar integrado ao servidor web, ele pode ser totalmente integrado ao código HTML.

Do ponto de vista de clareza de código que vimos até agora, isso não é exatamente algo bom. Entretanto, o aprendizado se dá mais facilmente deste jeito e, para os alunos que se interessarem, fica sugerido o estudo da biblioteca "***Smarty***" que permite separação praticamente completa entre o PHP e o HTML, completando o último nível de separação de informações: html, css, javascript e php, todos totalmente separados.

2. FUNCIONAMENTO SERVER SIDE

Conceitos Chave:

- Seqüência:
    * Usuário clica em link
    * Navegador solicita arquivo.php
        + http://www.caetano.eng.br/main/index.php
    * Servidor solicita que interpretador PHP processe arquivo.php
        + Exemplo de processamento PHP
    * Interpretador devolve HTML pronto ao servidor
    * Servidor repassa HTML ao navegador
    * Navegador mostra HTML para usuário
- Navegador recebe **só HTML**.
- Servidor envia sempre um documento inteiro
    * Alternativa ... uso em conjunto com JavaScript: AJAX

Recordemos agora o funcionamento da linguagem interpretada server-side.

Lembremos que a idéia era não haver páginas HTML no lado do servidor, há somente alguns programas capazes de **gerar** as páginas HTML necessárias. Qualquer página que seja solicitada pelo navegador ao servidor, será gerada por um destes programas, apenas no momento de enviá-la e, após o envio, ela será destruída.

O processo havia sido representado conforme a figura 1.

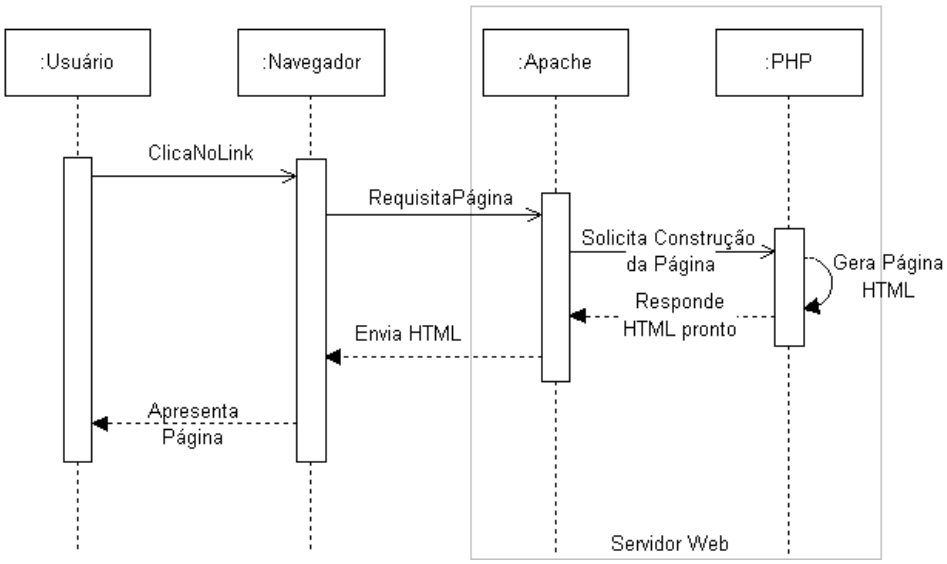


Figura 1: Processamento de uma requisição de página ao PHP

Por exemplo: suponhamos que o usuário solicite a seguinte página em seu navegador:

http://www.caetano.eng.br/

Que, na verdade, seria a seguinte página:

http://www.caetano.eng.br/main/index.php

Ora, INDEX.PHP é um programa em PHP e seu código, simplificadamente, poderia ser algo assim:

index.php

```
print ("<H1>T&iacute;tulo da p&aacute;gina</H1>\n");
print ("<P>Texto da página!</P>\n");
```

Considerando que ***print("...");*** é uma função que imprime tudo que estiver entre aspas, após o processamento pela linguagem PHP, o que será impresso e, portanto, chegará ao navegador do cliente é o que está indicado a seguir:

index.html *(gerado dinamicamente)*

```
<H1>T&iacute;tulo da p&aacute;gina</H1>
<P>Texto da página!</P>
```

Ou seja, o código em PHP (os "print"s no trecho original) não chega ao navegador. Apesar de o usuário indicar o nome de um programa PHP, o que chega até o navegador é o **resultado** do processamento (e não o código da linguagem). Assim, o navegador não precisa sequer saber qual linguagem foi utilizada.

Note, porém, 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. Para os interessados, é possível evitar isso, usando JavaScript e PHP em conjunto, no que se convencionou chamar de tecnologia "Ajax".

3. FUNCIONAMENTO DO PHP DENTRO DE UM ARQUIVO

Conceitos Chave:

- Seqüência:
  - * Entra: HTML+PHP
  - * Interpretador PHP processa
  - * Sai: HTML Puro
  - * Servidor envia HTML puro para servidor
- Como é arquivo PHP?
  - * Arquivo HTML
  - * Tag <?PHP ... ?>
- Interpretador processa só o que estiver delimitado pelas tags PHP.
- Exemplo.

Quando um usuário digita um endereço normal, e este aponta um arquivo .HTML, o Servidor Web simplesmente localiza aquele arquivo e, em seguida, envia seus dados ao computador do usuário, para que o navegador apresente as informações.

Por outro lado, quando o usuário digita um endereço que aponta para um arquivo com a extensão .PHP, o Servidor Web automaticamente 'pensa': "Ei, eu não entendo este tal de PHP... eu só entendo HTML! Vou chamar o intérprete!".

Neste momento, ele executa o interpretador PHP e passa para ele o arquivo .PHP. O interpretador PHP, por sua vez, irá ler este arquivo, fazer o seu trabalho e, ao terminar, diz para o Servidor Web: "Então, lembra aquela tradução que você me pediu? Está aqui... em formato .HTML". A partir de então o Servidor Web faz o que ele sabe fazer: envia aqueles dados HTML para o computador do cliente, que irá mostrar os dados.

O processo está indicado graficamente na figura 2. Este figura não apresenta os passos do Servidor Web para simplificar a compreensão.

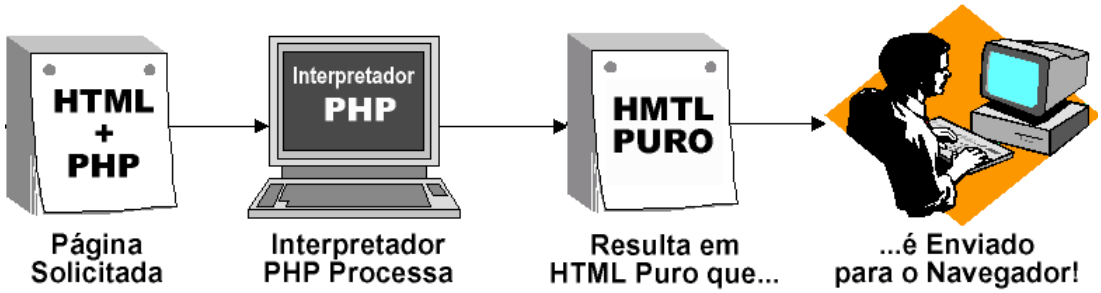


Figura 2: Fluxo de conversão de PHP para HTML puro

Mas como é um arquivo .PHP por dentro? É algo tão complexo assim? Na verdade, não. Um arquivo .PHP costuma ser um arquivo HTML normal, mas contendo alguns trechos de código embutido. A aparência geral de uma página simples com código PHP é a seguinte:

arquivo.php

```
<HTML LANG="pt-BR">
	<HEAD>
		<TITLE>Teste de PHP</TITLE>
	</HEAD>
	<BODY>
		<?PHP
			[... Código PHP ...]
		?>
	</BODY>
</HTML>
```

O Interpretador PHP não irá fazer **nada** com as linhas HTML normais. Entretanto, quando ele encontrar a tag:

<?PHP
....
?>

Ele irá processar tudo que encontrar aí dentro.

A seguir temos uma tabela que indica os passos do processamento do arquivo PHP pelo interpretador. A primeira coluna apresenta o arquivo de entrada e a segunda o arquivo de saída (que será repassado ao Servidor Web).

A linha em negrito mostra a linha sendo "interpretada" no instante.

Código Original	Saída em HTML
<HTML LANG="pt-BR"> <HEAD><TITLE>Teste PHP</TITLE></HEAD> <BODY> <?PHP print ("<P>Um teste.</P>\n"); ?> </BODY> </HTML>	**<HTML LANG="pt-BR">**
<HTML LANG="pt-BR"> **<HEAD><TITLE>Teste PHP</TITLE></HEAD>** <BODY> <?PHP print ("<P>Um teste.</P>\n"); ?> </BODY> </HTML>	<HTML LANG="pt-BR"> **<HEAD><TITLE>Teste PHP</TITLE></HEAD>**
<HTML LANG="pt-BR"> <HEAD><TITLE>Teste PHP</TITLE></HEAD> **<BODY>** <?PHP print ("<P>Um teste.</P>\n"); ?> </BODY> </HTML>	<HTML LANG="pt-BR"> <HEAD><TITLE>Teste PHP</TITLE></HEAD> **<BODY>**
<HTML LANG="pt-BR"> <HEAD><TITLE>Teste PHP</TITLE></HEAD> <BODY> **<?PHP print ("<P>Um teste.</P>\n"); ?>** </BODY> </HTML>	<HTML LANG="pt-BR"> <HEAD><TITLE>Teste PHP</TITLE></HEAD> <BODY> <P>Um teste.</P>
<HTML LANG="pt-BR"> <HEAD><TITLE>Teste PHP</TITLE></HEAD> <BODY> <?PHP print ("<P>Um teste.</P>\n"); ?> **</BODY>** </HTML>	<HTML LANG="pt-BR"> <HEAD><TITLE>Teste PHP</TITLE></HEAD> <BODY> <P>Um teste.</P> **</BODY>**
<HTML LANG="pt-BR"> <HEAD><TITLE>Teste PHP</TITLE></HEAD> <BODY> <?PHP print ("<P>Um teste.</P>\n"); ?> </BODY> **</HTML>**	<HTML LANG="pt-BR"> <HEAD><TITLE>Teste PHP</TITLE></HEAD> <BODY> <P>Um teste.</P> </BODY> **</HTML>**

A primeira, segunda e terceira linhas são simplesmente copiadas pelo interpretador PHP, já que se trata de código HTML.

A quarta linha o interpretador processa, pois encontra o "<?PHP ... ?>" nela. Ao invés de reproduzi-la na saída, ele coloca o resultado do processamento em seu lugar.

A quinta e sexta linhas ele, novamente, apenas copia, pois tratam-se se linhas de código HTML puro.

4. PRIMEIRO TESTE COM PHP

A primeira função que veremos será para avaliar se o PHP está instalado corretamente no servidor e se estamos trabalhando corretamente. A função usada para isso será a função **phpinfo()**, que tem por objetivo mostrar a configuração completa do Servidor Web e de nossa instalação do Interpretador PHP.

Para realizar o teste, crie o arquivo abaixo em seu desktop:

index.php

```
<HTML>
	<HEAD>
		<TITLE>Teste de PHP</TITLE>
	</HEAD>
	<BODY>
		<?PHP
			phpinfo();
		?>
	</BODY>
</HTML>
```

Gravando este arquivo, ao abrir no navegador veremos que, infelizmente, há algo errado, pois a página ficará completamente em branco. O que está errado?

O que está **errado** é que não colocamos o arquivo no servidor!

Para que o PHP seja processado, o arquivo tem que ser solicitado pelo servidor. Assim, para que possamos ver esse arquivo (index.php), grave-o na seguinte pasta:

C:\WAMP\WWW

Lembrando que "localhost" apontará para sua própria máquina, sempre, em qualquer sistema operacional moderno, abra seu navegador e digite:

http://localhost/

Isso fará com que uma página com muitas informações sobre o PHP e o servidor apareçam no navegador.

5. PROGRAMANDO EM PHP

Agora que já sabemos como fazer uma página PHP ser processada, é interessante introduzir alguns conceitos e funções.

A primeira função de que falaremos, ***print***, já apresentada anteriormente. Sua função é apresentar uma informação na tela. Uma página simples, completa, usando a função **print** está representada a seguir.

teste1.php

```
<HTML LANG="pt-BR">
        <HEAD>
                <TITLE>Teste de PHP</TITLE>
        </HEAD>
        <BODY>
                <?PHP
                        print ("<H1>T&iacute;tulo da P&aacute;gina</H1>\n");
                        print ("<P>Texto da P&aacute;gina</P>\n");
                ?>
        </BODY>
</HTML>
```

Salve este arquivo no diretório do servidor (C:\WAMP\WWW) e teste em seu navegador, usando o endereço: http://localhost/teste1.php.

Uma página absolutamente simples será apresentada:

Título da Página

Texto da Página

Como o texto a ser impresso precisa estar dentro de aspas (" ... "), não é possível imprimir o caractere de aspas diretamente. Para isso deve-se usar a indicação:

\"

Que indica para o comando print que o **caractere "** deve ser impresso. Assim:

print (" Isso é um caractere de aspas: \" ");

Será impresso da seguinte forma:

Isso é um caractere de aspas: "

Neste momento você deve estar se perguntando: mas para que usar este tal de PHP dentro do HTML se o que ele faz é exatamente igual ao que o HTML faz?

De fato, o uso do PHP no formato em que apresentamos até agora não faz muito sentido. Entretanto, modificando um pouco o código, teremos um efeito mais interessante:

teste1.php

```
<HTML LANG="pt-BR">
	<HEAD>
		<TITLE>Teste de PHP</TITLE>
	</HEAD>
	<BODY>
		<?PHP
			$pagina = 1;
			print ("<H1>T&iacute;tulo da P&aacute;gina $pagina</H1>\n");
			print ("<P>Texto da P&aacute;gina $pagina</P>\n");
		?>
	</BODY>
</HTML>
```

Grave este arquivo novamente e recarregue a página no servidor. Da forma como a página foi escrita, o interpretador PHP responderá o seguinte texto, que será enviado ao navegador (confira com o "Exibir Código Fonte"!):

```
<HTML>
	<HEAD>
		<TITLE>Teste de PHP</TITLE>
	</HEAD>
	<BODY>
<H1>P&aacute;gina 1</H1>
<P>Texto da P&aacute;gina 1</P>
	</BODY>
</HTML>
```

O resultado visual deve ser similar ao apresentado a seguir:

Título da Página 1

Texto da Página 1

O que acabamos de fazer foi criar uma variável chamada **$pagina** e colocamos o número da página dentro desta variável. Em PHP, **uma variável sempre começa com o caractere $**.

Experimente mudar o valor da variável para 2, e recarregue a página. Veja o que acontece!

teste1.php

```
<HTML LANG="pt-BR">
	<HEAD>
		<TITLE>Teste de PHP</TITLE>
	</HEAD>
	<BODY>
		<?PHP
			$pagina = 2;
			print ("<H1>T&iacute;tulo da P&aacute;gina $pagina</H1>\n");
			print ("<P>Texto da P&aacute;gina $pagina</P>\n");
		?>
	</BODY>
</HTML>
```

Uma observação importante é que, diferentemente do JavaScript, para imprimir uma variável basta colocar seu nome *dentro do texto* a ser impresso:

```
print ("<H1>T&iacute;tulo da P&aacute;gina $pagina</H1>\n");
```

O PHP reconhece o caractere $ como sendo o início de uma variável, e ao invés de imprimir o nome desta variável, ele coloca o valor dela no lugar.

Se quiser imprimir um "$" em um texto, deve digitá-lo duas vezes dentro do print, da seguinte forma:

```
print ("Isto é um $$.");
```

Uma forma mais interessante de usar variáveis é para a tomada de decisão, usando a diretiva **IF**. Por exemplo, suponhamos que desejemos que um texto só apareça quando o valor de $pagina for 1. Isso pode ser feito da seguinte forma:

teste1.php

```
<HTML LANG="pt-BR">
	<HEAD>
		<TITLE>Teste de PHP</TITLE>
	</HEAD>
	<BODY>
		<?PHP
			$pagina = 1;
			print ("<H1>T&iacute;tulo da P&aacute;gina $pagina</H1>\n");
			if ($pagina == 1) {
			    print ("<P>Texto da P&aacute;gina $pagina</P>\n");
			    }
		?>
	</BODY>
</HTML>
```

Grave o arquivo e teste no navegador: http://localhost/teste1.php . Depois, mude o valor de $pagina para um outro valor qualquer, e execute novamente. O que mudou?

A diretiva **if** é usada para construir uma estrutura de controle de fluxo de execução, sendo bastante comum nas linuguagens de programação... e o PHP certamente não é uma exceção.

A estrutura construída com **if** serve para decidir quando realizar determinadas ações: se a verificação que está dentro do parênteses for verdadeira, o código no bloco (delimitado por { }) após o **if** será executado. Ou seja, no código anteriormente testado:

```
if ($pagina == 1) {
    print ("<P>Texto da P&aacute;gina $pagina</P>\n");
    }
```

Significa: "Se o valor da variável **$pagina for igual a 1**, imprima o texto ***<P>Texto da Página 1</P>***".

Agora, suponhamos que um texto diferente deva ser apresentado quando o valor de **$pagina** for diferente de 1.

Isso, neste caso, pode ser feito com a diretiva ELSE, que significa "caso contrário..." e deve sempre acompanhar uma comparação anterior, feita com IF. Veja o caso abaixo:

teste1.php

```
<HTML LANG="pt-BR">
    <HEAD>
        <TITLE>Teste de PHP</TITLE>
    </HEAD>
    <BODY>
        <?PHP
            $pagina = 1;
            print ("<H1>T&iacute;tulo da P&aacute;gina $pagina</H1>\n");
            if ($pagina == 1) {
                print ("<P>Texto da P&aacute;gina $pagina</P>\n");
                }
            else {
                print ("<P>Texto Diferente.</P>\n");
                }
        ?>
    </BODY>
</HTML>
```

Grave o arquivo e recarregue-o no navegador, vendo o que acontece quando o valor da variável $pagina é igual a 1. Agora edite o arquivo e mude o valor da variável para outro qualquer. Recarregue a página. O que mudou?

A palavra ELSE, que sempre vem depois de um IF, indica qual é o bloco que deve ser executado caso a afirmação testada pelo IF não seja verdadeira.

Mas... e se quiséssemos criar uma página com 2 textos e um terceiro caso a página não exista? Simples, basta "aninhar" os IFs, da seguinte forma:

teste1.php

```
<HTML LANG="pt-BR">
	<HEAD>
		<TITLE>Teste de PHP</TITLE>
	</HEAD>
	<BODY>
		<?PHP
			$pagina = 1;
			print ("<H1>T&iacute;tulo da P&aacute;gina $pagina</H1>\n");
			if ($pagina == 1) {
			   print ("<P>Uma primeira p&aacute;gina.</P>\n");
			   }
			else {
			   if ($pagina == 2) {
			      print ("<P>Uma segunda p&aacute;gina.</P>\n");
			      }
			   else {
			      print ("<P>P&aacute;gina inexistente..</P>\n");
			      }
			   }
		?>
	</BODY>
</HTML>
```

Grave e teste para diferentes valores da variável $pagina, incluindo 0, 1, 2, 3, -1...

Sempre que um ELSE é seguido diretamente (e exclusivamente) por um IF, é comum abolir as chaves do ELSE, simplificando o código:

teste1.php

```
<HTML LANG="pt-BR">
	<HEAD>
		<TITLE>Teste de PHP</TITLE>
	</HEAD>
	<BODY>
		<?PHP
			$pagina = 1;
			print ("<H1>T&iacute;tulo da P&aacute;gina $pagina</H1>\n");
			if ($pagina == 1) {
			   print ("<P>Uma primeira p&aacute;gina.</P>\n");
			   }
			else if ($pagina == 2) {
			   print ("<P>Uma segunda p&aacute;gina.</P>\n");
```

```
                    }
                else {
                    print ("<P>P&aacute;gina inexistente..</P>\n");
                    }
            ?>
        </BODY>
</HTML>
```

A adição de alguns comentários podem ajudar na compreensão do código:

teste1.php

```
<HTML LANG="pt-BR">
    <HEAD>
        <TITLE>Teste de PHP</TITLE>
    </HEAD>
    <BODY>
        <?PHP
            $pagina = 1;
            print ("<H1>T&iacute;tulo da P&aacute;gina $pagina</H1>\n");
            /* Se estivermos na página 1... */
            if ($pagina == 1) {
                print ("<P>Uma primeira p&aacute;gina.</P>\n");
                }
            /* Se estivermos na página 2... */
            else if ($pagina == 2) {
                print ("<P>Uma segunda p&aacute;gina.</P>\n");
                }
            /* Se em nenhuma delas, a página não existe! */
            else {
                print ("<P>P&aacute;gina inexistente..</P>\n");
                }
        ?>
    </BODY>
</HTML>
```

5.1. PHP é uma Linguagem Fracamente Tipada

Um aspecto importante da linguagem PHP é que não é necessário informar tipos de variáveis, ou seja, se uma variável serve para guardar números, se serve para guardar textos...

De uma maneira geral, o PHP "adivinha" sozinho o tipo da variável. Por esta razão, é necessário algum cuidado com a execução de operações "estranhas", como por exemplo:

```
$var1 = "2";
$var2 = "2";
$var3 = $var1+$var2;
```

Qual será o resultado? O resultado é 4, mas observe que não era óbvio. O resultado poderia ser "22". Isso ocorre porque, quando é realizada uma operação aritmética, o PHP sempre tenta converter o valor para um número... e isso pode inclusive conduzir a resultados estranhos. Por exemplo:

```
$var1 = "teste";
$var2 = "2";
$var3 = $var1+$var2;
```

Resulta em $var3 valendo "2" (já que a conversão de teste para um número não é possível, o PHP considera o valor zero).

No PHP, o sinal "+" é sempre usado para SOMAR os valores numéricos de cada componente. Se desejarmos a **concatenação** de duas variáveis texto (chamadas *strings*), isto é, grudar uma na outra, deve-se usar o operador **.** (ponto). Ele é usado da seguinte forma:

```
$var1 = "teste";
$var2 = "2";
$var3 = $var1 . $var2;
```

O resultado será "teste2". No caso de ambos serem números:

```
$var1 = "2";
$var2 = "2";
$var3 = $var1 . $var2;
```

Teremos como resultado "22".

5.2. Estrutura FOR, WHILE e DO...WHILE

Assim como na maioria das outras linguagens de programação, para conseguir execuções repetidas de alguns trechos de código usamos laços FOR. A sintaxe do laço FOR é:

```
for (definição_inicial; verificação_de_continuidade; regra_de_atualização)
        {
        ....
        }
```

Por exemplo, para escrever um texto "Linha N" 7 vezes na tela, onde N é o número da linha, usamos o seguinte código:

teste2.php

```
<HTML LANG="pt-BR">
        <HEAD>
                <TITLE>Teste de PHP</TITLE>
        </HEAD>
        <BODY>
                <?PHP
                        for ( $i=1;  $i<=7;  $i=$i+1) print ("<P>Linha $i</P>\n");
                ?>
        </BODY>
</HTML>
```

Que resulta em:

Linha 1
Linha 2
Linha 3
Linha 4
Linha 5
Linha 6
Linha 7

Isso é bastante útil, quando desejamos criar listas de formulários com muitos elementos. Por exemplo, para gerar uma combobox com todos os anos de 1970 a 2010, basta usar o seguinte código:

teste3.php

```
<HTML LANG="pt-BR">
<HEAD>
        <TITLE>Teste de PHP</TITLE>
</HEAD>
<BODY>
        <FORM>
                <SELECT>
                        <?PHP
                                for ( $i=1970; $i<=2010; $i=$i+1) {
                                   print ("<OPTION VALUE=\"$i\">$i</OPTION>\n");
                                   }
                        ?>
                </SELECT>
        </FORM>
</BODY>
</HTML>
```

Como em outras linguagens, é possível construir os loops com laços WHILE, como indicado abaixo. A única diferença do FOR para o WHILE é que as inicializações precisam ser feitas ANTES do WHILE (já que não há um campo para isso) e a atualização do contador precisa ser feita ao final do laço:

```
for ( $i=1970; $i<=2010; $i=$i+1) {
    print ("<OPTION VALUE=\"$i\">$i</OPTION>\n");
    }
```

```
$i=1970;
while ($i<=2010) {
    print ("<OPTION VALUE=\"$i\">$i</OPTION>\n");
    $i=$i+1;
    }
```

Mas por que dois jeitos de fazer a mesma coisa? Em alguns casos é mais conveniente fazer de uma forma ou de outra. Uma coisa que ambos, FOR e WHILE, possuem em comum é que a verificação é feita sempre ANTES de executar o bloco. Ou seja: a cada passo, antes de executar o bloco, a verificação é feita. Se for verdadeira, o bloco é processado. Se não for, o bloco é pulado.

Mas e se quiséssemos fazer um bloco que fosse sempre processado pelo menos uma vez, independente da condição?

Para isso criaram o laço DO ... WHILE. Este laço executa obrigatoriamente um bloco, verificando **ao final** do bloco se deve repeti-lo mais vezes. A conversão de um laço WHILE para um laço DO-WHILE normalmente só muda a posição da verificação.

```
$i=1970;
while ($i<=2010) {
    print ("<OPTION VALUE=\"$i\">$i</OPTION>\n");
    $i=$i+1;
    }
```

```
$i=1970;
do {
    print ("<OPTION VALUE=\"$i\">$i</OPTION>\n");
    $i=$i+1;
    } while ($i<=2010)
```

Convém lembrar que estes dois laços têm funções distintas. Por exemplo, considere os códigos abaixo:

```
$i=2050;
while ($i<=2010) {
    print ("<OPTION VALUE=\"$i\">$i</OPTION>\n");
    $i=$i+1;
    }
```

```
$i=2050;
do {
    print ("<OPTION VALUE=\"$i\">$i</OPTION>\n");
    $i=$i+1;
    } while ($i<=2010)
```

No primeiro caso, o do WHILE, como a comparação é feita ANTES de executar o bloco, nenhuma linha de opções será impressa, já que 2050 **não é** menor ou igual a 2010. O bloco será totalmente pulado.

No segundo caso, o do DO...WHILE, como a comparação é feita DEPOIS de executar o bloco, uma primeira opção com o ano 2050 será impressa, e só... por que a comparação será feita e 2051 **não é** menor ou igual a 2010... evitando que o bloco seja executado mais vezes.

6. BIBLIOGRAFIA

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 15: PHP - Funções e Parâmetros Básicos

Prof. Daniel Caetano

Objetivo: Apresentar a forma GET de passar parâmetros e algumas funções básicas do PHP.

Bibliografia: PHP, 2009; MUTO,2006; RATSCHILLER,2000.

INTRODUÇÃO

Conceitos Chave:
- Como modificar conteúdo sem ter que modificar código?
  * Parâmetros!
- Como deixar a nossa página PHP mais "limpa" e bonita?
  * Funções!

Foi apresentada anteriormente a linguagem PHP e algumas de suas características. Entretanto, com o que foi visto, nossa página era ainda muito limitada: para modificar seu conteúdo ainda precisávamos modificar o código!

Nesta aula, veremos que isso nem sempre é necessário, se soubermos usar **parâmetros** corretamente. Há várias formas de passar parâmetros para uma página; nesta aula nos veremos a primeira delas.

Adicionalmente, veremos algumas funções básicas do PHP, que possibilitem, de alguma forma, incrementar um pouco o visual de nossas páginas.

1. O QUE SÃO PARÂMETROS?

Como foi visto anteriormente, podíamos modificar o comportamento de uma página inteira simplesmente modificando um valor de variável (no caso, era a variável **$pagina**).

O problema é que, para modificar o valor da variável, era necessário modificar o código da própria página, algo muito pouco prático. Seria interessante se pudéssemos modificar o valor de uma variável **sem** precisar modificar o código da página.

E é exatamente para isso que servem os parâmetros: modificar o valor de variáveis sem ter de modificar o código!

Resumidamente, a passagem de parâmetros para uma página é uma forma de modificar o conteúdo de determinadas variáveis antes que a página seja executada (de forma que ela se comporte como desejamos), sem ter que alterar o código da página em si.

1.1. Como Passar Parâmetros?

A primeira dúvida que pode surgir é: "ok, eu posso modificar algumas variáveis sem ter de mexer no código da página... mas como é que eu faço isso?".

Há diversas maneiras de realizar esta tarefa; neste curso veremos duas delas mas, na aula de hoje, veremos apenas uma: através da URL, ou seja, do endereço da página.

Passar parâmetros para uma página pela URL é similar a passar parâmetros para um programa por linha de comando. Por exemplo, no Prompt do Windows ou DOS, para copiar um arquivo faríamos:

COPY ORIGEM.DOC DESTINO.DOC

"**COPY**" é o nome do programa, "**ORIGEM.DOC**" é o primeiro parâmetro e "**DESTINO.DOC**" é o nome do segundo parâmetro.

No início do curso, vimos que, na web, em geral temos uma linha de endereços com a seguinte "cara":

http://www.servidor.com/diretorio/pagina.html

Bem, se considerarmos a linha de endereços como uma espécie de "prompt" e essa URL indicar um programa PHP...

http://www.servidor.com/diretorio/mostra_pagina.php

... ela pode ser considerada como se fosse um comando e, para passar parâmetros para esse programa, usaremos o caractere "**?**" para separar o que é nome do comando do que é parâmetro.

Por exemplo, se quisermos definir que um parâmetro chamado "**pagina**" seja igual a **1**, basta indicar da seguinte forma:

http://www.servidor.com/diretorio/mostra_pagina.php?pagina=1

Note que, diferentemente do Prompt do Windows/DOS, na Web nós devemos não apenas indicar *valores* dos parâmetros, mas também o *nome* dos mesmos.

NOTA: Não se esqueça! Para indicar um parâmetro de um programa que roda na web, é necessário indicar o nome e o valor deste parâmetro!

Mas... agora uma outra dúvida pode surgir: e se quisermos passar ***mais de um*** parâmetro? Por exemplo, e se quisermos indicar, além do parâmetro **pagina=1**, quisermos indicar também o parâmetro **usuario=1345**?

Existe uma solução simples para isso: iremos indicar todos os parâmetros separados pelo símbolo "**&**", como apresentado abaixo:

http://www.servidor.com/diretorio/mostra_pagina.php?pagina=1&usuario=1345

Onde, pode-se observar, são definidos os valores de **pagina** como **1** e de **usuario** como **1345**.

1.2. Como Receber Parâmetros em Minha Página?

Ainda que as indicações que tenhamos feito acima *enviem de fato* os parâmetros para o servidor, é importante saber que eles não são enviados automaticamente para nosso programa.

NOTA: os parâmetros são passados para o servidor, e não diretamente para nosso programa PHP!

Assim, quando formos construir um programa PHP que usa parâmetros, a primeira coisa que devemos fazer é solicitar os valores dos parâmetros ao servidor!

Por exemplo, considere a página em PHP abaixo:

aula15.php

```
<HTML LANG="pt-BR">
        <HEAD><TITLE>Teste de Par&acirc;metrosHP</TITLE></HEAD>
        <BODY>
                <?PHP   print ("<H1>T&iacute;tulo da P&aacute;gina $pagina</H1>\n");
                        /* Se estivermos na página 1... */
                        if ($pagina == 1) print ("<P>Uma primeira p&aacute;gina.</P>\n");
                        /* Se estivermos na página 2... */
                        else if ($pagina == 2) print ("<P>Uma segunda p&aacute;gina.</P>\n");
                        /* Se em nenhuma delas, a página não existe! */
                        else print ("<P>P&aacute;gina inexistente..</P>\n");
                ?>
        </BODY>
</HTML>
```

Em princípio, ao carregar:

http://localhost/aula15.php

A mensagem "Página inexistente." será exibida. Na verdade, ainda que passemos um valor para o parâmetro **pagina**, a mensagem continuará aparecendo. Teste!

http://localhost/aula15.php?pagina=1

Esta página só é capaz de mostrar a mensagem "Página inexistente." porque ainda não pedimos que o servidor coloque na variável **$pagina** o valor do parâmetro **pagina**!

O jeito de fazer isso é usando a seguinte instrução:

$variavel **= $_GET['*parametro*'];**

No nosso caso, seria:

$pagina **= $_GET['*pagina*'];**

E isso precisaria ser acrescentado em nossa página antes de qualquer uso da variável **$pagina**, pois antes de fazer essa solicitação ao servidor, seu valor será "vazio". Assim, nada melhor do que fazer isso no topo da página!

aula15.php

```
<?PHP
$pagina = $_GET['pagina'];
?>
<HTML LANG="pt-BR">
	<HEAD><TITLE>Teste de Par&acirc;metrosHP</TITLE></HEAD>
	<BODY>
		<?PHP	print ("<H1>T&iacute;tulo da P&aacute;gina $pagina</H1>\n");
			/* Se estivermos na página 1... */
			if ($pagina == 1) print ("<P>Uma primeira p&aacute;gina.</P>\n");
			/* Se estivermos na página 2... */
			else if ($pagina == 2) print ("<P>Uma segunda p&aacute;gina.</P>\n");
			/* Se em nenhuma delas, a página não existe! */
			else print ("<P>P&aacute;gina inexistente..</P>\n");
		?>
	</BODY>
</HTML>
```

Podemos acrescentar um comentário, para que não nos esqueçamos do que esta linha faz, como indicado no código a seguir:

aula15.php

```
<?PHP
/* Solicita o valor do parâmetro "pagina" e coloca na variável $pagina */
$pagina = $_GET['pagina'];
?>
<HTML LANG="pt-BR">
	<HEAD><TITLE>Teste de Par&acirc;metrosHP</TITLE></HEAD>
	<BODY>
		<?PHP	print ("<H1>T&iacute;tulo da P&aacute;gina $pagina</H1>\n");
			/* Se estivermos na página 1... */
			if ($pagina == 1) print ("<P>Uma primeira p&aacute;gina.</P>\n");
			/* Se estivermos na página 2... */
			else if ($pagina == 2) print ("<P>Uma segunda p&aacute;gina.</P>\n");
			/* Se em nenhuma delas, a página não existe! */
			else print ("<P>P&aacute;gina inexistente..</P>\n");
		?>
	</BODY>
</HTML>
```

Observe que o nome da variável **não** precisa ser o mesmo do parâmetros. É perfeitamente possível usar códigos como os que seguem:

```
$var1 = $_GET['pagina'];
$var2 = $_GET['usuario'];
```

Observe que o nome das variáveis não é o mesmo do parâmetro e, ainda assim, o valor do parâmetro será transferido para a variável.

É usual, entretanto, usar variáveis com os mesmos nomes dos parâmetros:

```
$pagina = $_GET['pagina'];
$usuario = $_GET['usuario'];
```

A partir do momento que colhemos os valores dos parâmetros usando **$_GET**, basta usar a variável em que colocamos os valores (no exemplo anterior, **$var1** e **$var2** ou **$pagina** e **$usuario**) como se ela tivesse sido definida dentro do próprio documento PHP.

Modifique o código como indicado a seguir, e execute-o usando a URL:

http://localhost/aula15.php?pagina=1&usuario=alberto

Depois disso, execute-o novamente com esta outra URL:

http://localhost/aula15.php?pagina=2&usuario=carlos

aula15.php

```
<?PHP
/* Solicita o valor do parâmetro "pagina" e coloca na variável $pagina */
$pagina = $_GET['pagina'];
/* Solicita o valor do parâmetro "usuario" e coloca na variável $var1 */
$var1 = $_GET['usuario'];
?>
<HTML LANG="pt-BR">
        <HEAD><TITLE>Teste de Par&acirc;metrosHP</TITLE></HEAD>
        <BODY>
                <?PHP   print ("<H1>T&iacute;tulo da P&aacute;gina $pagina</H1>\n");
                        /* Se estivermos na página 1... */
                        if ($pagina == 1) print ("<P>Uma primeira p&aacute;gina.</P>\n");
                        /* Se estivermos na página 2... */
                        else if ($pagina == 2) print ("<P>Uma segunda p&aacute;gina.</P>\n");
                        /* Se em nenhuma delas, a página não existe! */
                        else print ("<P>P&aacute;gina inexistente..</P>\n");
                        print ("<P>Boa noite, $var1!</P>\n");
                ?>
        </BODY>
</HTML>
```

NOTA: Se você carregar a página, neste caso, sem definir os valores para os parâmetros "pagina" e "usuario", verificará que o PHP indicará um "Warning" (aviso), dizendo que os valor dos parâmetros não foram definidos. Para eliminar este aviso, basta indicar um @ na frente do nome da variável.

Por exemplo:

```
@$var1 = $_GET['usuario'];
```

2. FUNÇÕES BÁSICAS

Há algumas funções e instruções no PHP que são muito usadas, pela sua praticidade. Aqui serão apresentadas algumas delas, dentro de funções prontas e úteis.

2.1. A Função Date

A função **date** retorna uma data completa, de acordo com o formato indicado:

string date (string $formato [, int $carimbo_de_hora])

Por exemplo:

aula15b.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP
			$hoje = date('d-m-Y');
			print ("<P>Hoje &eacute; $hoje</P>\n";
		?>
	</BODY>
</HTML>
```

Também é possível mostrar as horas:

aula15b.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP
			$hoje = date('d-m-Y');
			print ("<P>Hoje &eacute; $hoje</P>\n";
			$hora = date('H:i:s');
			print ("<P>A hora &eacute; $hora</P>\n";
		?>
	</BODY>
</HTML>
```

É possível mostrar as duas coisas ao mesmo tempo:

aula15b.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP
			$hoje = date('d-m-Y H:i:s');
			print ("<P>Agora &eacute; $hoje</P>\n";
		?>
	</BODY>
</HTML>
```

Para saber todos os valores possíveis de formato, consulte o manual do PHP: http://br.php.net/manual/pt_BR/ . Nele, procure por (search for) **date** na lista de funções (function list), por exemplo.

2.2. A Instrução For

A instrução **for**, já apresentada anteriormente, é bastante usada, sobretudo para a criação de menus, por exemplo. Guardaremos nosso menu na variável **$menu**.

Como um menu é um elemento complexo, teremos que "estruturar" a nossa variável no formato de uma "tabela" com duas colunas, uma para o **texto da opção** e outro com o **link da opção**. Cada linha representa uma opção e, neste exemplo, faremos um menu de 3 opções:

	texto	link
0	Home	http://www.caetano.eng.br/
1	Ajuda	http://www.caetano.eng.br/ajuda/
2	Contato	mailto:daniel@caetano.eng.br

Para indicar em que posição queremos colocar o valor, usamos a seguinte construção do PHP:

$variável[linha][coluna] = valor;

No nosso caso, as linhas representam as informações de uma opção do menu, cada uma delas em uma coluna. A primeira opção é indicada pelas colunas da primeira linha...

...ocorre que em PHP a primeira linha de uma tabela é sempre a linha ZERO. Assim, a indicação dos dados da primeira linha será:

$menu[0][coluna] = valor;

Vamos agora armazenar o **texto da opção** na coluna **texto**...

```
$menu[0]['texto'] = "Home";
```

...e o **link da opção** na coluna **1**.

```
$menu[0]['link'] = "http://www.caetano.eng.br/";
```

Isso completa a primeira linha da tabela. É possível fazer o mesmo para a linha **1** e linha **2**:

```
$menu[1]['texto'] = "Ajuda";
$menu[1]['link'] = "http://www.caetano.eng.br/ajuda/";
$menu[2]['texto'] = "Contato";
$menu[2]['link'] = "mailto:daniel@caetano.eng.br";
```

Assim, o nosso arquivo aula15c.php pode ser iniciado:

aula15c.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP
			/* Define as opções do menu em uma "tabela" */
			$menu[0]['texto'] = "Home";
			$menu[0]['link'] = "http://www.caetano.eng.br/";
			$menu[1]['texto'] = "Ajuda";
			$menu[1]['link'] = "http://www.caetano.eng.br/ajuda/";
			$menu[2]['texto'] = "Contato";
			$menu[2]['link'] = "mailto:daniel@caetano.eng.br";
		?>
	</BODY>
</HTML>
```

Esse código armazena os dados do menu na "variável-tabela" **$menu**, mas não os apresenta na tela! Para apresentá-lo, usaremos a instrução **for**, que serve para repetir uma tarefa um certo número de vezes.

Se temos três linhas, numeradas de 0 a 2, podemos usar um **for** que vai de 0 a 2... para isso, precisamos de uma variável temporária para o **contador**. O nome usual deste tipo de variável é **$i** ou **$j**. No caso, usaremos **$i**, e o for terá o seguinte aspecto:

```
$num_linhas = 3;
for ($i = 0; $i < $num_linhas; $i = $i+1) {
  /* Aqui deve imprimir a linha do Menu */
  }
```

O código, por enquanto, fica assim:

aula15c.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP	/* Define as opções do menu em uma "tabela" */
			$menu[0]['texto'] = "Home";
			$menu[0]['link'] = "http://www.caetano.eng.br/";
			$menu[1]['texto'] = "Ajuda";
			$menu[1]['link'] = "http://www.caetano.eng.br/ajuda/";
			$menu[2]['texto'] = "Contato";
			$menu[2]['link'] = "mailto:daniel@caetano.eng.br";
			/* Imprime as opções do menu */
			$num_linhas = 3;
			for ($i = 0; $i < $num_linhas; $i = $i+1) {
			  /* Aqui deve imprimir a linha do Menu */
			  }
		?>
```

```
	</BODY>
</HTML>
```

Cada linha do menu deve ser impressa como um item de lista, que deve ter mais ou menos o seguinte aspecto:

<LI><A HREF="link_da_opção">texto_da_opção</A></LI>

Isso também pode ser descrito, simplificadamente, como:

<LI><A HREF="$link">$texto</A></LI>

Isso pode ser impresso pelo PHP com a seguinte instrução, indicada a seguir:

print ("<LI><A HREF=\"$link\">$texto</A></LI>\n");

NOTA: o código **\"** indica para que seja impresso o caractere "aspas"

Transportando isso para o código anterior...

aula15c.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP	/* Define as opções do menu em uma "tabela" */
			$menu[0]['texto'] = "Home";
			$menu[0]['link'] = "http://www.caetano.eng.br/";
			$menu[1]['texto'] = "Ajuda";
			$menu[1]['link'] = "http://www.caetano.eng.br/ajuda/";
			$menu[2]['texto'] = "Contato";
			$menu[2]['link'] = "mailto:daniel@caetano.eng.br";

			/* Imprime as opções do menu */
			$num_linhas = 3;
			for ($i = 0; $i < $num_linhas; $i = $i+1) {
			  /* Aqui imprime a linha do Menu */
			  print ("<LI><A HREF=\"$link\">$texto</A></LI>\n");
			  }
		?>
	</BODY>
</HTML>
```

Isso está quase bom, não fosse o fato de que *ainda não definimos o valor a ser impresso* nas variáveis $link e $texto, mas como já sabemos que o **link_da_opção** da linha **$i**

já está na variável $menu[**$i**][**'link'**] e o **texto_da_opção** da linha **$i** já está na variável $menu[**$i**][**'texto'**], basta indicar:

```
$texto = $menu[$i]['texto'];
$link = $menu[$i]['link'];
```

Dentro do loop que o menu será impresso.

> NOTA: O PHP não entende quando colocamos uma "variável-tabela" diretamente dentro de uma função **print**. Por essa razão usamos as variáveis temporárias **$link** e **$texto**.

Colocando isso no código, teremos:

aula15c.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP	/* Define as opções do menu em uma "tabela" */
			$menu[0]['texto'] = "Home";
			$menu[0]['link'] = "http://www.caetano.eng.br/";
			$menu[1]['texto'] = "Ajuda";
			$menu[1]['link'] = "http://www.caetano.eng.br/ajuda/";
			$menu[2]['texto'] = "Contato";
			$menu[2]['link'] = "mailto:daniel@caetano.eng.br";
			/* Imprime as opções do menu */
			$num_linhas = 3;
			for ($i = 0; $i < $num_linhas; $i = $i+1) {
			  /* Aqui imprime a linha do Menu */
			  $texto = $menu[$i]['texto'];
			  $link = $menu[$i]['link'];
			  print ("<LI><A HREF=\"$link\">$texto</A></LI>\n");
			  }
		?>
	</BODY>
</HTML>
```

Falta agora indicar <UL>...</UL> para que nosso menu esteja finalizado. Isso pode ser feito fora do <?PHP ... ?>:

aula15c.php

```
<HTML>
	<HEAD></HEAD>
	<BODY><UL>
		<?PHP	/* Define as opções do menu em uma "tabela" */
			$menu[0]['texto'] = "Home";
			$menu[0]['link'] = "http://www.caetano.eng.br/";
			$menu[1]['texto'] = "Ajuda";
			$menu[1]['link'] = "http://www.caetano.eng.br/ajuda/";
```

```
                        $menu[2]['texto'] = "Contato";
                        $menu[2]['link'] = "mailto:daniel@caetano.eng.br";
                        /* Imprime as opções do menu */
                        $num_linhas = 3;
                        for ($i = 0; $i < $num_linhas; $i = $i+1) {
                          /* Aqui imprime a linha do Menu */
                          $texto = $menu[$i]['texto'];
                          $link = $menu[$i]['link'];
                          print ("<LI><A HREF=\"$link\">$texto</A></LI>\n");
                          }
                ?></UL>
        </BODY>
</HTML>
```

Para finalizar, seria interessante que não tivéssemos de informar o número de linhas do menu, certo? Se ao invés de fazer:

$num_linhas = 3;

Pedirmos para o PHP **contar** o número de entradas da variável **$menu**, alcançaremos nosso objetivo. Isso pode ser feito da seguinte forma:

$num_linhas = count($menu);

Fazendo isso, o código final fica:

aula15c.php

```
<HTML>
        <HEAD></HEAD>
        <BODY>
                <UL>
                <?PHP   /* Define as opções do menu em uma "tabela" */
                        $menu[0]['texto'] = "Home";
                        $menu[0]['link'] = "http://www.caetano.eng.br/";
                        $menu[1]['texto'] = "Ajuda";
                        $menu[1]['link'] = "http://www.caetano.eng.br/ajuda/";
                        $menu[2]['texto'] = "Contato";
                        $menu[2]['link'] = "mailto:daniel@caetano.eng.br";
                        /* Imprime as opções do menu */
                        $num_linhas = count($menu);
                        for ($i = 0; $i < $num_linhas; $i = $i+1) {
                          /* Aqui imprime a linha do Menu */
                          $texto = $menu[$i]['texto'];
                          $link = $menu[$i]['link'];
                          print ("<LI><A HREF=\"$link\">$texto</A></LI>\n");
                          }
                ?>
                </UL>
        </BODY>
</HTML>
```

2.3. A diretiva Include

Quando elaboramos páginas web e somos obrigados a inserir um código que se repete por todas as páginas (como um cabeçalho ou um menu, por exemplo), surge sempre um desejo de que fosse possível definir o menu em um outro arquivo e, nos arquivos em que ele deve aparecer, simplemente podermos indicar "inclua aqui o menu".

Pois bem, é exatamente para esta finalidade que existe a diretiva "**include**", cujo objetivo é "copiar e colar" o conteúdo de um outro arquivo dentro do arquivo atual. Sendo assim, esta diretiva é bastante usada e é muito popular. Sua sintaxe é:

include "nome_de_arquivo"

Por exemplo, examinemos o código da seção anterior:

aula15c.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<UL>
		<?PHP	/* Define as opções do menu em uma "tabela" */
			$menu[0]['texto'] = "Home";
			$menu[0]['link'] = "http://www.caetano.eng.br/";
			$menu[1]['texto'] = "Ajuda";
			$menu[1]['link'] = "http://www.caetano.eng.br/ajuda/";
			$menu[2]['texto'] = "Contato";
			$menu[2]['link'] = "mailto:daniel@caetano.eng.br";
			/* Imprime as opções do menu */
			$num_linhas = count($menu);
			for ($i = 0; $i < $num_linhas; $i = $i+1) {
			  /* Aqui imprime a linha do Menu */
			  $texto = $menu[$i]['texto'];
			  $link = $menu[$i]['link'];
			  print ("<LI><A HREF=\"$link\">$texto</A></LI>\n");
			  }
		?>
		</UL>
	</BODY>
</HTML>
```

O menu em si é a parte pintada de azul. Assim, para adaptar essa página ao uso da diretiva include, ela fica da seguinte forma:

aula15d.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP   include "menu.php" ?>
	</BODY>
</HTML>
```

E, no arquivo **menu.php**, colocamos todas as linhas que retiramos do código **aula15c.php**, que estavam marcadas em azul:

menu.php

```
<UL>
	<?PHP
		/* Define as opções do menu em uma "tabela" */
		$menu[0]['texto'] = "Home";
		$menu[0]['link'] = "http://www.caetano.eng.br/";
		$menu[1]['texto'] = "Ajuda";
		$menu[1]['link'] = "http://www.caetano.eng.br/ajuda/";
		$menu[2]['texto'] = "Contato";
		$menu[2]['link'] = "mailto:daniel@caetano.eng.br";

		/* Imprime as opções do menu */
		$num_linhas = count($menu);
		for ($i = 0; $i < $num_linhas; $i = $i+1) {
		  /* Aqui imprime a linha do Menu */
		  $texto = $menu[$i]['texto'];
		  $link = $menu[$i]['link'];
		  print ("<LI><A HREF=\"$link\">$texto</A></LI>\n");
		  }
	?>
</UL>
```

É importante lembrar que o arquivo a ser incluído pode ser um outro arquivo .php, um arquivo .html ou mesmo um .txt.

Um fato muito importante é que o nome do arquivo do include **pode conter variáveis.** Assim, se criarmos um parâmetro GET chamado "m", que conterá o tipo de menu a ser carregado, podemos incluir arquivos diferentes para menu.

O código a seguir faz uso desta característica para carregar diferentes menus de acordo com o valor do parâmetro "m".

aula15e.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP
			/* Pega o valor do parâmetro "m" na variável $tipom */
			@$tipom = $_GET['m'];
			/* Garante que o valor da variávei $tipom está entre 1 e 3 */
			if ($tipom < 1 || $tipom > 3) $tipom = 1;
			/* Cria nome do arquivo de menu */
			$menuarq = "menu" . $tipom . ".php";
			/* Inclui o arquivo correto */
			include $menuarq;
		 ?>
	</BODY>
</HTML>
```

Neste caso, se m = 1, o arquivo **menu1.php** será incluído. Se m = 2, o arquivo **menu2.php** será incluído. Se m = 3, o arquivo **menu3.php** será incluído. Para qualquer outro valor de m, o arquivo **menu1.php** será incluído.

NOTA: a @ na frente do nome da variável serve para suprimir um "aviso" de que o valor do parâmetro "m" pode não estar definido. Isso é esperado!

Isso é incomum para selecionar o menu, mas é muito comum para selecionar o conteúdo da página.

Por exemplo:

aula15f.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP
			/* Inclui Menu */
			include "menu1.php";

			/* Pega o valor do parâmetro "pag" na variável $pagina */
			@$pagina = $_GET['pag'];
			/* Garante que o valor da variável $pagina está entre 1 e 2 */
			if ($pagina < 1 || $pagina > 2) $pagina = 1;
			/* Cria nome do arquivo de conteudo */
			$conteudo = "pag" . $pagina . ".php";
			/* Inclui o arquivo correto */
			include $conteudo;
		 ?>
	</BODY>
</HTML>
```

menu1.php

```
<!--  Menu que carrega diferentes conteúdos, selecionando conteúdo pelo
      parâmetro "pag" //-->
<UL>
      <LI><A HREF="?pag=1">Home</A></LI>
      <LI ><A HREF="?pag=2">Produtos</A></LI>
</UL>
```

pag1.php

```
<H1>Home Page</H1>
```

pag2.php

```
<H1>Lista de Produtos</H1>
```

2.4. A Função Strlen (OPCIONAL)

É relativamente comum a necessidade de identificar a quantidade de caracteres de um texto (uma string). A função **strlen** (de STRing LENgth) realiza esta tarefa. A forma de uso é:

int strlen (string $string)

Nos exemplos abaixo, o primeiro imprime o valor 6, e o segundo o valor 7:

aula15g.php

```
<HTML>
      <HEAD></HEAD>
      <BODY>
            <?PHP   $texto = 'abcdef';
                    echo strlen($texto);
            ?>
      </BODY>
</HTML>
```

aula15g.php

```
<HTML>
      <HEAD></HEAD>
      <BODY>
            <?PHP   $texto = ' ab de g';
                    echo strlen($texto);
            ?>
      </BODY>
</HTML>
```

2.5. A Função strcmp (OPCIONAL)

Em algumas circunstâncias queremos saber se os textos de duas variáveis são iguais, ou seja, se as duas strings são iguais. Para esta necessidade existe a função **strcmp**. Sua sintaxe da função strcmp é a seguinte:

int strcmp (string $string1, string $string2)

O que esta função "retorna" é um valor que depende do resultado da comparação:

0 => $string1 = $string2
<0 => $string1 < $string2
>0 => $string1 > $string2

Em geral o valor <0 é "-1" e o valor >0 é 1; em todo caso, nas comparações "if" deve-se usar o critério <0 e >0, uma vez que os valores -1 e 1 podem mudar no futuro.

O código a seguinr, por exemplo, deverá imprimir 3 números: 0, 1 e -1:

aula15h.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP
			$texto1 = 'ab cd ';
			$texto2 = 'ab cd ';
			$texto3 = 'aa cd ';
			$texto4 = 'zb cd ';
			echo strcmp($texto1, $texto2);
			echo "<br>";
			echo strcmp($texto2, $texto3);
			echo "<br>";
			echo strcmp($texto3, $texto4);
		?>
	</BODY>
</HTML>
```

2.6. A Função substr (OPCIONAL)

Algumas vezes queremos copiar um pedaço de um texto. Para esta finalidade, existe a função **substr** e sua sintaxe é:

string substr (string $string, int $início [, int $comprimento])

Por exemplo, o código abaixo deverá imprimir: cdef:

aula15i.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP	$texto = 'abcdefgh';
			echo substr($texto, 2, 4);
		?>
	</BODY>
</HTML>
```

3. COMO CRIAR UMA FUNÇÃO?

Em alguns momentos, vamos querer criar nossas próprias funções, automatizando algumas atividades realizadas com freqüência. Muitas vezes colocamos várias funções úteis dentro de um único arquivo chamado, por exemplo, **functions.php**, e incluímos (include) este arquivo em todas as páginas que usarem uma ou mais daquelas funções.

O processo para criar uma função é bastante simples: basta usar a diretiva **function**. A forma correta de usá-la é:

```
function nome_da_função($parametro1, $parametro2, ..., $parâmetro n)
	{
	}
```

Por exemplo:

aula15i.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP
			/* Cria a Função */
			function ImprimeDataCompleta() {
			  $agora = date('d-m-Y H:i:s');
			  print ("<P>Agora &eacute; $agora</P>\n");
			  }

			/* Usa a Função */
			ImprimeDataCompleta();
		?>
	</BODY>
</HTML>
```

3.1. Exemplos de Funções Úteis (OPCIONAL)

Uma função muito útil é aquela que substitui um caractere por outro, em um texto. Por exemplo, uma funão que substitui espaços por "underline", para corrigir nomes de arquivos digitados pelo usuário.

NOTA: Antes de tratar esta função, é importante apresentar um conceito: sempre que uma função do PHP for **alterar** o conteúdo de uma variável passada como parâmetro, o nome desta variável deve vir precedida do caractere **&**. O uso deste caractere será apresentado no exemplo seguinte.

O código desta função é assim:

aula15j.php

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP
			/* Define Função */
			function LimpaEspaco($texto_entrada, &$texto_saida) {
			  $texto_saida = "";
			  $comprimento = strlen($texto_entrada);
			  for ($i = 0; $i < $comprimento; $i = $i + 1) {
			    $temp = substr($texto_entrada, $i, 1);
			    if ($temp == " ") $temp="_";
			    $texto_saida = $texto_saida . $temp;
			    }
			  }

			/* Usa Função */
			$texto = "Com espa&ccedil;o no texto.";
			print ("<P>$texto</P>\n");
			LimpaEspaco($texto, $texto2);
			print ("<P>$texto2</P>\n");
		?>
	</BODY>
</HTML>
```

Uma outra função bastante útil é aquela que corrige texto digitado pelo usuário, convertendo-o para a codificação HTML automaticamente:

aula15k.php

```
<?PHP
function TextToHTML (&$text)
   {
   $texthtml = "";
   /* Se texto de entrada está vazio, retorna sem fazer nada */
   if (($text == "") || ($text == NULL)) return;
```

```
    /* Loop: executa processo para cada letra do texto */
    for ($i = 0; $i < strlen($text); $i = $i + 1) {
      /* Pega letra atual, indicada por $i */
      $tempstr = substr($text,$i,1);
      /* Verifica se alguma substituição é necessária e a faz */
      if ($tempstr == "á") $tempstr = "&aacute;";
      else if ($tempstr == "é") $tempstr = "&eacute;";
      else if ($tempstr == "í") $tempstr = "&iacute;";
      else if ($tempstr == "ó") $tempstr = "&oacute;";
      else if ($tempstr == "ú") $tempstr = "&uacute;";
      else if ($tempstr == "â") $tempstr = "&acirc;";
      else if ($tempstr == "ê") $tempstr = "&ecirc;";
      else if ($tempstr == "î") $tempstr = "&icirc;";
      else if ($tempstr == "ô") $tempstr = "&ocirc;";
      else if ($tempstr == "û") $tempstr = "&ucirc;";
      else if ($tempstr == "à") $tempstr = "&agrave;";
      else if ($tempstr == "è") $tempstr = "&egrave;";
      else if ($tempstr == "ì") $tempstr = "&igrave;";
      else if ($tempstr == "ò") $tempstr = "&ograve;";
      else if ($tempstr == "ù") $tempstr = "&ugrave;";
      else if ($tempstr == "ä") $tempstr = "&auml;";
      else if ($tempstr == "ë") $tempstr = "&euml;";
      else if ($tempstr == "ï") $tempstr = "&iuml;";
      else if ($tempstr == "ö") $tempstr = "&ouml;";
      else if ($tempstr == "ü") $tempstr = "&uuml;";
      else if ($tempstr == "ã") $tempstr = "&atilde;";
      else if ($tempstr == "õ") $tempstr = "&otilde;";
      else if ($tempstr == "ç") $tempstr = "&ccedil;";
      else if ($tempstr == "Á") $tempstr = "&Aacute;";
      else if ($tempstr == "É") $tempstr = "&Eacute;";
      else if ($tempstr == "Í") $tempstr = "&Iacute;";
      else if ($tempstr == "Ó") $tempstr = "&Oacute;";
      else if ($tempstr == "Ú") $tempstr = "&Uacute;";
      else if ($tempstr == "À") $tempstr = "&Agrave;";
      else if ($tempstr == "È") $tempstr = "&Egrave;";
      else if ($tempstr == "Ì") $tempstr = "&Igrave;";
      else if ($tempstr == "Ò") $tempstr = "&Ograve;";
      else if ($tempstr == "Ù") $tempstr = "&Ugrave;";
      else if ($tempstr == "Ä") $tempstr = "&Auml;";
      else if ($tempstr == "Ë") $tempstr = "&Euml;";
      else if ($tempstr == "Ï") $tempstr = "&Iuml;";
      else if ($tempstr == "Ö") $tempstr = "&Ouml;";
      else if ($tempstr == "Ü") $tempstr = "&Uuml;";
      else if ($tempstr == "Ã") $tempstr = "&Atilde;";
      else if ($tempstr == "Õ") $tempstr = "&Otilde;";
      else if ($tempstr == "Ç") $tempstr = "&Ccedil;";
      /* Acrescenta caractere, modificado ou não, no texto de saída */
      $texthtml= $texthtml . $tempstr;
      }
    /* Copia texto de saída na variável de entrada */
    $text = $texthtml;
    }
?>
```

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP
			/* Usa Função */
			$texto = "Este é uma sentença com acentos á é à õ ü.";
			print ("<P>$texto</P>\n");
			TextToHTML($texto);
			print ("<P>$texto</P>\n");
		?>
	</BODY>
</HTML>
```

A última função que veremos é uma que imprime a data na forma completa e em língua portuguesa.

Neste caso usaremos um formato que retorna o resultado de uma maneira diferente, usando a instrução **return**.

aula15l.php

```
<?PHP
function MakeStringFromTime($time) {
  /* Define Textos das Datas */
  $week[0]="Domingo";
  $week[1]="Segunda";
  $week[2]="Ter&ccedil;a";
  $week[3]="Quarta";
  $week[4]="Quinta";
  $week[5]="Sexta";
  $week[6]="S&aacute;bado";
  $month[1]="Janeiro";
  $month[2]="Fevereiro";
  $month[3]="Mar&ccedil;o";
  $month[4]="Abril";
  $month[5]="Maio";
  $month[6]="Junho";
  $month[7]="Julho";
  $month[8]="Agosto";
  $month[9]="Setembro";
  $month[10]="Outubro";
  $month[11]="Novembro";
  $month[12]="Dezembro";
  /* Pega dia, mês e dia da semana atuais */
 $mon=date ("n", $time);
 $day=date ("j", $time);
 $wee=date ("w", $time);
  /* Pega ano, hora, minuto e segundo atuais */
  $enddate=date ("Y, G:i:s",$time);
  /* Imprime resultado */
  return  $week[$wee] . ", " . $day . " de " . $month[$mon] . " de " . $enddate;
 }
?>
```

```
<HTML>
	<HEAD></HEAD>
	<BODY>
		<?PHP
			/* Usa Função */
			$time = time();
			print(MakeStringFromTime($time));
		?>
	</BODY>
</HTML>
```

Vale lembrar que todas estas funções podem ser colocadas em um único arquivo "funcoes.php", que será incluído no código com uma diretiva **include "funcoes.php";**

6. BIBLIOGRAFIA

PHP: Manual do PHP, disponível em: < http://br.php.net/manual/pt_BR/ >. Visitado em 13/04/2009.

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 16: PHP - Recebendo e Tratando dados POST

Prof. Daniel Caetano

Objetivo: Apresentar a forma POST de passar parâmetros e um preliminar tratamento dos dados recebidos.

Bibliografia: PHP, 2009; MUTO,2006; RATSCHILLER,2000.

INTRODUÇÃO

Conceitos Chave:
- Parâmetros pela URL
  - * $_GET['...']
  - - Deselegante
  - - Limitado
- Como receber os dados que foram enviados por um formulário?
  - * $_POST['...']
- Processar direto?
  - * Validar!
- E depois?
  - * Usar os dados!

Já foi apresentada, anteiormente, a forma de pegar parâmetros passados pela URL, usando o método GET. Entretanto, apesar de ser uma forma extremamente simples de passar parâmetros, esta é uma maneira deselegante e limitada.

É deselegante porque permite que o usuário veja toda a "sujeira" por trás do funcionamento de sua página, permitindo inclusive que ele altere facilmente alguns valores que não deveria modificar.

Além disso, é limitada, porque impede que alguns tipos de dados sejam passados: o endereço web tem limitação de tamanho. Por esta razão, imagens e outros arquivos grandes não podem ser transmitidos pelo método GET.

A solução para evitar estes problemas será o envio de dados por formulários, com o uso do método POST para recuperá-los.

1. UM FORMULÁRIO EXEMPLO

Antes de começarmos a trabalhar com o método POST, precisamos de um formulário para enviar os dados. Vamos usar como base um formulário bastante simples, aquele implementado na atividade passada, que está no arquivo **pag3.php**:

pag3.php

```
<H1>Contato</H1>
<FORM ACTION="?cont=8" METHOD="post" NAME="form1" ID="form1">
  <FIELDSET>
    <LEGEND>Informa&ccedil;&otilde;es Pessoais</LEGEND>
    Nome: <INPUT TYPE="text" MAXLENGTH="40" SIZE="40" NAME="nome" ID="nome" ><BR>
    <FIELDSET>
      <LEGEND>Sexo</LEGEND>
      <INPUT TYPE="radio" NAME="sexo" ID="masc" VALUE="mas" CHECKED>Masculino
      <INPUT TYPE="radio" NAME="sexo" ID="fem" VALUE="fem">Feminino
      </FIELDSET><BR>
    CPF: <INPUT TYPE="text" MAXLENGTH="11" SIZE="11" NAME="cpf" ID="cpf"><BR>
    </FIELDSET>
  <INPUT TYPE="submit" VALUE="Enviar">
  </FORM>
<P ID="help"></P>
```

Este formulário tem três campos:

a) *nome*: TextBox com o nome da pessoa
b) *sexo*: RadioBox com sexo da pessoa
c) *cpf*: TextBox com o cpf da pessoa

O nome destes campos será importante, pois eles definem os **nomes dos parâmetros** que serão recebidos no arquivo de processamento, que será o **pag8.php**.

2. COMO RECEBER PARÂMETROS POST

Consideremos que o formulário foi aberto por um usuário e que ele digitou algumas informações. Quando este formulário for enviado, os dados dos campos serão transferidos para o servidor e então o programa/página **pag8.php** será carregado.

O nosso arquivo pag8.php, inicialmente, irá apenas imprimir os valores digitados pelo usuário. Para isso, iremos modificar o arquivo pag8.php. Para recordar:

pag8.php

```
<P>Obrigado por entrar em contato!</P>
```

Iremos agora modificar este arquivo, para que ele imprima os valores digitados:

pag8.php

```
<P>Obrigado por entrar em contato!</P>
<?PHP
        /* Imprime Nome */
        print("<P>Nome: $usr_nome</P>\n");
        /* Imprime Sexo */
        print("<P>Sexo: $usr_sexo</P>\n");
        /* Imprime CPF */
        print("<P>CPF: $usr_cpf</P>\n");
?>
```

Se ainda não o fez, copie todos os arquivos da página, incluindo o **pag8.php** modificado, para o diretório raiz do webserver (por exemplo: C:\WAMP\WWW). Carregue a página:

http://localhost/

O arquivo index.php será carregado automaticamente, mostrando nossa página. Clique no link "Contato" para abrir o formlário. Preencha os campos e clique em "Enviar". O resultado deverá ser algo parecido com a seguinte página:

Além dos "avisos" (Notice) de que as variáveis não foram definidas, os valores digitados não foram apresentados! Por que isso ocorreu?

A resposta é simples: os dados foram enviados para o servidor, mas nosso programa PHP não pegou estes dados!

A coleta destes dados pode ser feita desta forma:

$variavel **= $_POST['*parametro*'];**

Onde o "parâmetro" é o nome que demos ao campo (usando o modificador NAME lá no HTML). No caso, por exemplo, no nome da pessoa:

```
<INPUT TYPE="text" MAXLENGTH="40" SIZE="40" NAME="nome" ID="nome" />
```

A instrução que precisamos no PHP seria:

$usr_nome **= $_POST['*nome*'];**

Outro exemplo, para o campo onde é indicado o sexo:

```
<INPUT TYPE="radio" NAME="sexo" ID="masc" VALUE="mas" CHECKED />
<INPUT TYPE="radio" NAME="sexo" ID="fem" VALUE="fem" />
```

A instrução que precisamos no PHP seria:

$usr_sexo **= $_POST['*sexo*'];**

Finalmente, no caso do campo do CPF:

```
<INPUT TYPE="text" MAXLENGTH="11" SIZE="11" NAME="cpf" ID="cpf" />
```

A instrução que precisamos no PHP seria:

$usr_cpf **= $_POST['*cpf*'];**

É natural que "peguemos" o valor destas variáveis com o servidor **antes** de usá-las em na página e, assim, o melhor lugar para fazer estas declarações é no início do arquivo **pag8.php**:

pag8.php

```
<P>Obrigado por entrar em contato!</P>
<?PHP
    /* PHP 'pega dados' com o Servidor Web */
    $usr_nome = $_POST['nome'];
    $usr_sexo = $_POST['sexo'];
    $usr_cpf = $_POST['cpf'];
```

```
        /* Imprime Nome */
        print("<P>Nome: $usr_nome</P>\n");
        /* Imprime Sexo */
        print("<P>Sexo: $usr_sexo</P>\n");
        /* Imprime CPF */
        print("<P>CPF: $usr_cpf</P>\n");
?>
```

Grave este arquivo novamente (no diretório do servidor) e recarregue a página:

http://localhost/

Clicando, agora, no link "Contato". Preencha alguns valores nos campos e clique em "Enviar". O que aconteceu agora?

Se tudo correu bem, os dados digitados/selecionados devem ter sido recuperados e impressos na tela, como indicado abaixo. Isso significa que conseguimos receber os parâmetros com sucesso!

NOTA: Você deve ter percebido que, neste caso, não precisamos usar o @ na frente do $_POST, como fazíamos antes com o $_GET. Isso ocorre porque um formulário **sempre** manda um valor nos parâmetros, ainda que o valor seja **nulo**.

3. VALIDANDO DADOS NO PHP

Como foi dito anteriormente, não é possível confiar na validação do JavaScript, já que ele pode ser desligado. Assim, é preciso validar os dados também no "programa" **pag8.php**, antes de usá-los.

Como o JavaScript já está validando o preenchimento de nosso formulário, vamos desligar temporariamente a validação do JavaScript, simulando um navegador sem JavaScript. Para isso, vá nas opções do seu navegador e desligue o JavaScript.

No FireFox, abra a janela de opções (Ferramentas > Opções), clique na aba Conteúdo e desmarque a opção "Liga JavaScript" ou "Habilita JavaScript".

No Internet Explorer, abra a janela de opções (Ferramentas > Opções da Internet), clique na aba Segurança, selecione o ícone "Sites Restritos", clique no botão "Sites" e adicione o endereço "localhost" na lista.

Um método alternativo é modificar o arquivo **veriform.js**, comentando a linha "window.onload = init;", ou seja, modificando esta linha para:

```
// window.onload = init;
```

Teste o formulário sem preencher nada e veja se funcionou, isto é, se o JavaScript parou de validar o formulário. Vamos, agora, fazer uma primeira validação simples no PHP, que é a verificação de preenchimento do nome, que deve ser maior ou igual a 5 caracteres:

pag8.php

```
<P>Obrigado por entrar em contato!</P>
<?PHP
        /* PHP 'pega dados' com o Servidor Web */
        $usr_nome = $_POST['nome'];
        $usr_sexo = $_POST['sexo'];
        $usr_cpf = $_POST['cpf'];

        /* Valida nome */
        if (strlen($usr_nome) < 5) {
          print ("<P>O nome deve ter pelo menos 5 caracteres!</P>\n");
          print ("<P><A HREF=\"?cont=3\">Voltar</A></P>\n");
          }

        /* Imprime Nome */
        print("<P>Nome: $usr_nome</P>\n");
        /* Imprime Sexo */
        print("<P>Sexo: $usr_sexo</P>\n");
        /* Imprime CPF */
        print("<P>CPF: $usr_cpf</P>\n");
?>
```

Preencha o formulário com um nome menor que 5 caracteres e veja que a página irá reclamar do nome muito curto.

NOTA: Essa verificação foi feita no servidor, e exigiu que uma nova página inteira fosse enviada. Para evitar isso usamos a validação com o JavaScript, mas se o JavaScript for alterado ou desligado, a única alternativa é verificar no servidor.

Nossa validação ainda tem um problema: mesmo quando temos um erro, o resultado está agradecendo o envio dos dados e imprimindo os dados; nós queremos que ele só faça essas coisas quando os dados **passarem** pela validação. O código abaixo corrige este comportamento:

pag8.php

```
<P>Obrigado por entrar em contato!</P>
<?PHP
        /* PHP 'pega dados' com o Servidor Web */
        $usr_nome = $_POST['nome'];
        $usr_sexo = $_POST['sexo'];
        $usr_cpf = $_POST['cpf'];

        /* Valida nome */
        if (strlen($usr_nome) < 5) {
          print ("<P>O nome deve ter pelo menos 5 caracteres!</P>\n");
          print ("<P><A HREF=\"?cont=3\">Voltar</A></P>\n");
          }
        else {
          /* Imprime mensagem de agradecimento */
          print ("<P>Obrigado por entrar em contato!</P>\n");
          /* Imprime Nome */
          print("<P>Nome: $usr_nome</P>\n");
          /* Imprime Sexo */
          print("<P>Sexo: $usr_sexo</P>\n");
          /* Imprime CPF */
          print("<P>CPF: $usr_cpf</P>\n");
          }
?>
```

Verifique que, agora, as coisas estão funcionando como deveriam. Vamos, agora, verificar o CPF. Por simplicidade, vamos verificar apenas se ele tem exatamente 11 caracteres:

pag8.php

```
<?PHP /* PHP 'pega dados' com o Servidor Web */
        $usr_nome = $_POST['nome'];
        $usr_sexo = $_POST['sexo'];
        $usr_cpf = $_POST['cpf'];

        /* Valida nome */
        if (strlen($usr_nome) < 5) {
          print ("<P>O nome deve ter pelo menos 5 caracteres!</P>\n");
          print ("<P><A HREF=\"?cont=3\">Voltar</A></P>\n");
          }
        /* Valida CPF */
        else if (strlen($usr_cpf) != 11) {
          print ("<P>O cpf deve ter exatamente 11 caracteres!</P>\n");
          print ("<P><A HREF=\"?cont=3\">Voltar</A></P>\n");
          }
```

```
        else {
          /* Imprime mensagem de agradecimento */
          print ("<P>Obrigado por entrar em contato!</P>\n");
          /* Imprime Nome */
          print("<P>Nome: $usr_nome</P>\n");
          /* Imprime Sexo */
          print("<P>Sexo: $usr_sexo</P>\n");
          /* Imprime CPF */
          print("<P>CPF: $usr_cpf</P>\n");
          }
?>
```

Adicionalmente, podemos exigir que o CPF seja numérico, o que pode ser feito com a função ***is_numeric($variavel)***, que retorna TRUE se a variável for numérica ou FALSE se não for. A implementação está no código a seguir. Depois de testar, você pode ligar novamente o JavaScript.

pag8.php

```
<?PHP /* PHP 'pega dados' com o Servidor Web */
        $usr_nome = $_POST['nome'];
        $usr_sexo = $_POST['sexo'];
        $usr_cpf = $_POST['cpf'];

        /* Valida nome */
        if (strlen($usr_nome) < 5) {
          print ("<P>O nome deve ter pelo menos 5 caracteres!</P>\n");
          print ("<P><A HREF=\"?cont=3\">Voltar</A></P>\n");
          }
        /* Valida CPF */
        else if (strlen($usr_cpf) != 11) {
          print ("<P>O cpf deve ter exatamente 11 caracteres!</P>\n");
          print ("<P><A HREF=\"?cont=3\">Voltar</A></P>\n");
          }
        else if (is_numeric($usr_cpf) == FALSE) {
          print ("<P>O cpf deve ser num&eacute;rico!</P>\n");
          print ("<P><A HREF=\"?cont=3\">Voltar</A></P>\n");
          }
        else {
          /* Imprime mensagem de agradecimento */
          print ("<P>Obrigado por entrar em contato!</P>\n");
          /* Imprime Nome */
          print("<P>Nome: $usr_nome</P>\n");
          /* Imprime Sexo */
          print("<P>Sexo: $usr_sexo</P>\n");
          /* Imprime CPF */
          print("<P>CPF: $usr_cpf</P>\n");
          }
?>
```

4. VALIDAÇÃO DE DADOS PROFISSIONAL (OPCIONAL: AVANÇADO)

Como pode ser verificada, a solução acima não é muito profissional: a mensagem de erro deveria ser apresentada na mesma página do formulário; além disso, os dados previamente digitados não deveriam ser perdidos!

Para que isso seja possível, é preciso incluir as verificações do arquivo **pag8.php** dentro do arquivo **pag3.php**. Na prática, isso será feito com um include e algumas modificações em ambos. Começaremos renomeando o arquivo **pag8.php** para **veriform.php** e, no arquivo **conteudo.php**, faremos a seguinte modificação:

conteudo.php

```
<?PHP
        /* Pega número de Página sendo Mostrado */
        @$arq = $_GET['cont'];
        if ($arq<1 || $arq > 7) $arq = 1;

        /* Imprime Data nas Páginas 1 ou 7 */
        if ($arq == 1 || $arq == 7) imprimeData();

        /* Inclui arquivo de conteúdo da página atual */
        $pag = "pag" . $arq . ".php";
        include $pag;
?>
```

Isso libera a página 8 para ser usada para outra coisa no futuro. Agora, começaremos as nossas modificações no arquivo **pag3.php**, que deverá incluir o arquivo **veriform.php** antes do formulário, além de, agora, enviar os dados do formulário para si própria:

pag3.php

```
<H1>Contato</H1>
<?PHP
        include "veriform.php";
?>
<FORM ACTION="?cont=3" METHOD="post" NAME="form1" ID="form1">
  <FIELDSET>
    <LEGEND>Informa&ccedil;&otilde;es Pessoais</LEGEND>
    Nome: <INPUT TYPE="text" MAXLENGTH="40" SIZE="40" NAME="nome" ID="nome" ><BR>
    <FIELDSET>
      <LEGEND>Sexo</LEGEND>
      <INPUT TYPE="radio" NAME="sexo" ID="masc" VALUE="mas" CHECKED>Masculino
      <INPUT TYPE="radio" NAME="sexo" ID="fem" VALUE="fem">Feminino
      </FIELDSET><BR>
    CPF: <INPUT TYPE="text" MAXLENGTH="11" SIZE="11" NAME="cpf" ID="cpf"><BR>
    </FIELDSET>
  <INPUT TYPE="submit" VALUE="Enviar">
  </FORM>
<P ID="help"></P>
```

Atualizando os arquivos modificados no servidor, recarregue a página: http://localhost/ . Clicando no link "Contato", observe que surgiram alguns problemas:

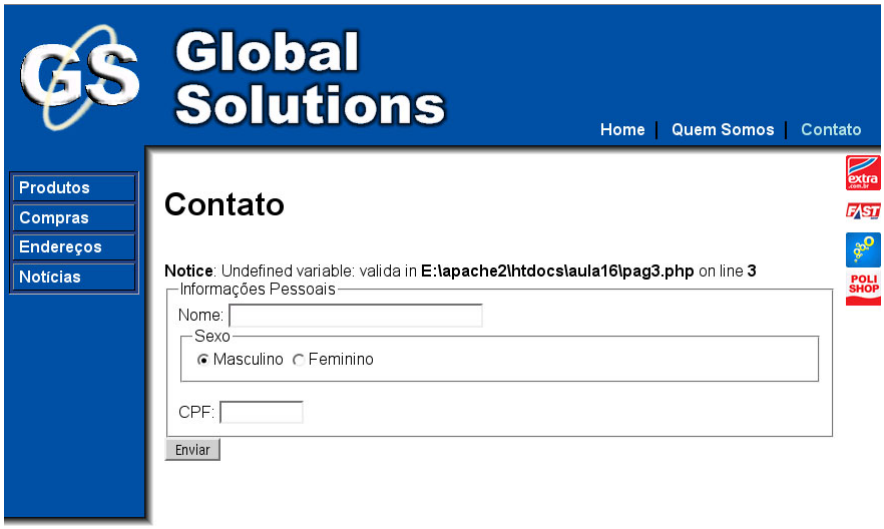
Isso ocorreu pelo seguinte fato: o nosso programa **veriform.php** está tentando validar os dados enviados pelo formulário... só que nenhum dado foi enviado! Vamos modificar o código da página **pag3.php** para que só inclua o código do validador caso seja necessário.

Essa modificação será feita da seguinte forma: será verificado o valor da variável **$valida** e, se ele for igual a 1 (UM), o arquivo veriform.php será incluído:

pag3.php

```
<H1>Contato</H1>
<?PHP
        if ($valida==1) include "veriform.php";
?>
<FORM ACTION="?cont=3" METHOD="post" NAME="form1" ID="form1">
  <FIELDSET>
    <LEGEND>Informa&ccedil;&otilde;es Pessoais</LEGEND>
    Nome: <INPUT TYPE="text" MAXLENGTH="40" SIZE="40" NAME="nome" ID="nome" ><BR>
    <FIELDSET>
      <LEGEND>Sexo</LEGEND>
      <INPUT TYPE="radio" NAME="sexo" ID="masc" VALUE="mas" CHECKED>Masculino
      <INPUT TYPE="radio" NAME="sexo" ID="fem" VALUE="fem">Feminino
      </FIELDSET><BR>
    CPF: <INPUT TYPE="text" MAXLENGTH="11" SIZE="11" NAME="cpf" ID="cpf"><BR>
    </FIELDSET>
  <INPUT TYPE="submit" VALUE="Enviar">
  </FORM>
<P ID="help"></P>
```

Gravando o arquivo atualizado no servidor e entrando novamente na página Contato, vemos que não eliminamos todos os nossos problemas, mas eles melhoraram, como pode ser visto na figura abaixo:



O problema agora ocorre porque o valor da variável **$valida** não foi definido.

Ora, só queremos que este valor seja definido quando o formulário for enviado... Então, nada mais razoável que definir este valor no próprio formulário, com o tipo de campo **hidden**, ou seja, invisível. Adicionalmente, pegaremos o valor do campo do formulário com o $_POST:

pag3.php

```
<H1>Contato</H1>
<?PHP
        @$valida = $_POST['validar'];
        if ($valida==1) include "veriform.php";
?>
<FORM ACTION="?cont=3" METHOD="post" NAME="form1" ID="form1">
  <INPUT TYPE="hidden" NAME="validar" ID="validar" VALUE="1">
  <FIELDSET>
    <LEGEND>Informa&ccedil;&otilde;es Pessoais</LEGEND>
    Nome: <INPUT TYPE="text" MAXLENGTH="40" SIZE="40" NAME="nome" ID="nome" ><BR>
    <FIELDSET>
      <LEGEND>Sexo</LEGEND>
      <INPUT TYPE="radio" NAME="sexo" ID="masc" VALUE="mas" CHECKED>Masculino
      <INPUT TYPE="radio" NAME="sexo" ID="fem" VALUE="fem">Feminino
      </FIELDSET><BR>
    CPF: <INPUT TYPE="text" MAXLENGTH="11" SIZE="11" NAME="cpf" ID="cpf"><BR>
    </FIELDSET>
  <INPUT TYPE="submit" VALUE="Enviar">
  </FORM>
<P ID="help"></P>
```

NOTA: É necessário o uso do @ neste $_POST pois, na primeira vez em que a página for carregada, o formulário **não** terá enviado o valor do parâmetro "validar", e teremos um aviso se não usarmos a @.

O nosso validador está quase funcionando, mas ele ainda imprime a mensagem "Voltar", que não é mais necessária. Para eliminar isso, editaremos o código **veriform.php**:

veriform.php

```
<?PHP /* PHP 'pega dados' com o Servidor Web */
        $usr_nome = $_POST['nome'];
        $usr_sexo = $_POST['sexo'];
        $usr_cpf = $_POST['cpf'];

        /* Valida nome */
        if (strlen($usr_nome) < 5) {
          print ("<P>O nome deve ter pelo menos 5 caracteres!</P>\n");
          print ("<P><A HREF=\"?cont=3\">Voltar</A></P>\n");
          }
        /* Valida CPF */
        else if (strlen($usr_cpf) != 11) {
          print ("<P>O cpf deve ter exatamente 11 caracteres!</P>\n");
          print ("<P><A HREF=\"?cont=3\">Voltar</A></P>\n");
          }
        else if (is_numeric($usr_cpf) == FALSE) {
          print ("<P>O cpf deve ser num&eacute;rico!</P>\n");
          print ("<P><A HREF=\"?cont=3\">Voltar</A></P>\n");
          }
        else {
          /* Imprime mensagem de agradecimento */
          print ("<P>Obrigado por entrar em contato!</P>\n");
          /* Imprime Nome */
          print("<P>Nome: $usr_nome</P>\n");
          /* Imprime Sexo */
          print("<P>Sexo: $usr_sexo</P>\n");
          /* Imprime CPF */
          print("<P>CPF: $usr_cpf</P>\n");
          }
?>
```

Isso deixa nosso validador quase perfeito e profissional, com excessão de dois poréns:

1) Quando há erro, os dados são apagados e é necessário preenchê-los novamente.
2) Quando os dados são preenchidos corretamente, o formulário aparece novamente.

O primeiro item é fácil corrigir: basta pegar os dados enviados do formulário no arquivo **pag3.php** e imprimi-los nos valores dos campos já existentes. Isso pode ser feito como indicado no código a seguir:

NOTA: Não use os nomes de variáveis definidos no arquivo veriform.php, pois assim você estaria "amarrando" muito o código do formulário ao código da validação.

pag3.php

```
<H1>Contato</H1>
<?PHP
        @$valida = $_POST['validar'];
        @$nome = $_POST['nome'];
        @$sexo = $_POST['sexo'];
        @$cpf = $_POST['cpf'];
        if ($valida==1) include "veriform.php";
?>
<FORM ACTION="?cont=3" METHOD="post" NAME="form1" ID="form1">
  <INPUT TYPE="hidden" NAME="validar" ID="validar" VALUE="1">
  <FIELDSET>
    <LEGEND>Informa&ccedil;&otilde;es Pessoais</LEGEND>
    Nome: <INPUT TYPE="text" MAXLENGTH="40" SIZE="40"
                <?PHP print ("VALUE=\"$nome\""); ?>
                NAME="nome" ID="nome" ><BR>
    <FIELDSET>
      <LEGEND>Sexo</LEGEND>
      <INPUT TYPE="radio" NAME="sexo"
                <?PHP if ($sexo!="fem") print ("CHECKED"); ?>
                ID="masc" VALUE="mas">Masculino
      <INPUT TYPE="radio" NAME="sexo"
                <?PHP if ($sexo=="fem") print ("CHECKED"); ?>
                ID="fem" VALUE="fem">Feminino
      </FIELDSET><BR>
    CPF: <INPUT TYPE="text" MAXLENGTH="11" SIZE="11"
                <?PHP print ("VALUE=\"$cpf\""); ?>
                NAME="cpf" ID="cpf"><BR>
    </FIELDSET>
  <INPUT TYPE="submit" VALUE="Enviar">
  </FORM>
<P ID="help"></P>
```

O segundo problema vai depender de fazermos uma pequena modificação no nosso arquivo **veriform.php**, transformando a validação em uma função chamada **validaForm()**, que deve retornar **false** caso a validação tenha sucesso (para NÃO mostrar o formulário) ou **true** caso tenha algum erro (para MOSTRAR o formulário), como no código a seguir:

veriform.php

```
<?PHP
function validaForm() {
        /* PHP 'pega dados' com o Servidor Web */
        $usr_nome = $_POST['nome'];
        $usr_sexo = $_POST['sexo'];
        $usr_cpf = $_POST['cpf'];

        /* Valida nome */
        if (strlen($usr_nome) < 5) print ("<P>O nome deve ter pelo menos 5 caracteres!</P>\n");
        /* Valida CPF */
        else if (strlen($usr_cpf) != 11) print ("<P>O cpf deve ter exatamente 11 caracteres!</P>\n");
        else if (is_numeric($usr_cpf) == FALSE) print ("<P>O cpf deve ser num&eacute;rico!</P>\n");
```

```
        else {
          /* Imprime mensagem de agradecimento */
          print ("<P>Obrigado por entrar em contato!</P>\n");
          /* Imprime Nome */
          print("<P>Nome: $usr_nome</P>\n");
          /* Imprime Sexo */
          print("<P>Sexo: $usr_sexo</P>\n");
          /* Imprime CPF */
          print("<P>CPF: $usr_cpf</P>\n");
          return false;
          }
        return true;
        }
?>
```

E, finalmente, uma mudança no arquivo pag3.php, que deve, agora, chamar esta função:

pag3.php

```
<H1>Contato</H1>
<?PHP
        /* Pega dados no servidor */
        @$valida = $_POST['validar'];
        @$nome = $_POST['nome'];
        @$sexo = $_POST['sexo'];
        @$cpf = $_POST['cpf'];
        /* Inicialmente, $mostraForm vale true */
        $mostraForm = true;
        /* Executa validação e corrige o valor de $mostraForm, se for o caso */
        if ($valida==1) {
                include "veriform.php";
                $mostraForm = validaForm();
                }
/* Só imprime o HTML a seguir se $mostraForm for true! */
if ($mostraForm) {
?>
<FORM ACTION="?cont=3" METHOD="post" NAME="form1" ID="form1">
  <INPUT TYPE="hidden" NAME="validar" ID="validar" VALUE="1">
  <FIELDSET>
    <LEGEND>Informa&ccedil;&otilde;es Pessoais</LEGEND>
    Nome: <INPUT TYPE="text" MAXLENGTH="40" SIZE="40"
                <?PHP print ("VALUE=\"$nome\""); ?>
                NAME="nome" ID="nome" ><BR>
    <FIELDSET>
      <LEGEND>Sexo</LEGEND>
      <INPUT TYPE="radio" NAME="sexo"
                <?PHP if ($sexo!="fem") print ("CHECKED"); ?>
                ID="masc" VALUE="mas">Masculino
      <INPUT TYPE="radio" NAME="sexo"
                <?PHP if ($sexo=="fem") print ("CHECKED"); ?>
                ID="fem" VALUE="fem">Feminino
      </FIELDSET><BR>
    CPF: <INPUT TYPE="text" MAXLENGTH="11" SIZE="11"
```

```
                <?PHP print ("VALUE=\"$cpf\""); ?>
                NAME="cpf" ID="cpf"><BR>
    </FIELDSET>
  <INPUT TYPE="submit" VALUE="Enviar">
  </FORM>
<P ID="help"></P>
<?PHP
}
?>
```

E pronto, nosso formulário com validação server-side profissional está finalizado!

5. BIBLIOGRAFIA

PHP: Manual do PHP, disponível em: < http://br.php.net/manual/pt_BR/ >. Visitado em 13/04/2009.
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 17: PHP - Integrando PHP com MySQL

Prof. Daniel Caetano

Objetivo: Apresentar o uso de bancos de dados em páginas web.

Bibliografia: PHP, 2009; MUTO,2006; RATSCHILLER,2000.

INTRODUÇÃO

Conceitos Chave:
- Páginas Dinâmicas
  - * Como armazenar dados de usuários?
  - * Bancos de Dados!
    - - Dados cadastrais!
    - - Busca!
- Integração com PHP
  - * Abstração de banco de dados
  - * Possibilita implementação de buscas com facilidade
- MySQL v5

Já foi visto, anteriormente, uma base sobre como usar o PHP para construir páginas web dinâmicas. Entretanto, ainda não foi possível armazenar ou recuperar os dados digitados pelos usuários. Para que isso possa ocorrer de maneira consistente e independente de navagador, dependemos do uso de bancos de dados.

O PHP fornece muitas formas de acesso a banco de dados, a diversos tipos de bancos de dados diferentes. Em especial, fornece até mesmo um sistema de abstração de bancos de dados. Neste curso veremos como usar o banco de dados MySQL que, em sua versão 5.x, deixa muito pouco a desejar para grandes bancos de dados comerciais, com a vantagem de ser um produto de código aberto e uso livre para aplicações não comerciais.

O objetivo desta aula não é o aprendizado do SQL, uma vez que este será apresentado, com detalhes, em momento oportuno. A idéia é apresentar como é feita a integração entre um banco de dados existente e páginas web. Será apresentado o mínimo necessário sobre bancos de dados.

Convém ressaltar que os bancos de dados não servem apenas para armazenamento de dados de cadastro, embora este seja o foco desta aula. O uso de banco de dados pode tornar absolutamente dinâmico o comportamento de uma página, permitindo desde logins e carregamento de características visuais para cada usuário até permitir a substituição do sistema de "includes" de conteúdo pelo armazenamento completo dos conteúdos em bancos de dados, facilitando a atualização das páginas e também proporcionando a fácil e rápida execução de busca de conteúdo.

1. ACESSO AO MySQL COM PHP

Conceitos Chave:
- Conexão com o SQL
- Seleção do Banco de Dados
- Buscas
- Desconexão

Em qualquer aplicação que use o banco de dados, o processo é o mesmo:

1) Conexão com o Servidor SQL
2) Seleção de Banco de Dados
3) Buscas (queries)
4) Desconexão

Com aplicações PHP não é diferente, sendo também um processo bastante simples. Vejamos como cada um deles é executado.

1.1. Conexão com o Servidor SQL

Conceitos Chave:
- mysql_connect
  - * *recurso* **mysql_connect** ([string $server [, string $user [, string $pass]]])
  - * **mysql_connect**("*servidor*", "*usuário*", "*senha*")
- mysql_error
  - * $texto_do_erro = **mysql_error** ([resource $sqlcon])
  - * echo **mysql_error** ()

Antes de mais nada, obviamente, precisamos conectar com o servidor. O PHP já possui uma função pronta para realizar essa conexão: basta que forneçamos os dados para que a conexão ocorra. O nome desta função é **mysql_connect**. A sintaxe básica desta função é:

recurso **mysql_connect** ([string $servidor [, string $nome_usuário [, string $password]]])

O significado de servidor, nome_usuário e password são mais ou menos óbvios: são os parâmetros da conexão. Entretanto, o retorno *recurso* exige alguma explicação. Quando chamamos esta função, uma conexão é estabelecida entre o micro onde o PHP roda e o servidor SQL. Este *recurso* retornado é o *número identificador* desta conexão. Caso ocorra algum erro de conexão, o retorno será o valor *FALSE*. Assim, o uso mais comum seria:

```
$sqlcon = mysql_connect("localhost", "usuario_web", "senha");
```

Feito isso, considerando que exista um usuário "**usuario_web**" cuja senha seja "**senha**" e que ele pode acessar o banco de dados a partir do computador local (**localhost**), **$sqlcon** conterá o identificador da conexão com o SQL atual.

Quando se realiza apenas UMA conexão, não é necessário manipular o número da conexão ($sqlcon) e, por isso, usaremos uma versão simplificada da instrução:

```
mysql_connect("localhost", "usuario_web", "senha");
```

Entretanto, como pode ocorrer um erro de conexão, é comum que este erro seja tratado e informado ao usuário. Para imprimir este (e outros erros), podemos usar a função **mysql_error**, que sempre devolverá o erro ocorrido na última instrução para o MySQL. Seu uso, normalmente, é feito da seguinte forma:

```
$texto_do_erro = mysql_error ( [resource $sqlcon] )
print ("Erro no SQL: " . $texto_do_erro);
```

Novamente, quando temos apenas uma conexão, não é necessário o uso do valor $sqlcon, simplificando o uso do mysql_error. Assim, a maneira mais usual de se fazer uma conexão com o banco e tratar os possíveis erros é:

```
if ( mysql_connect("localhost", "usuario_web", "senha") != FALSE ) {
        [ ... código de acesso ao banco de dados ... ]
        }
else {
        $msg_erro = mysql_error();
        print ("Erro no SQL: " . $msg_erro);
        }
```

Considerando que a conexão ocorreu com sucesso, é hora de selecionar o banco de dados, já que no sistema MySQL podem existir vários bancos ao mesmo tempo.

1.2. Seleção de Banco de Dados

Conceitos Chave:

- mysql_select_db
  - * *bool* **mysql_select_db** (string $banco [, resource $sqlcon])
  - * *bool* **mysql_select_db** ("*nome_do_banco*")

A seleção de banco de dados é simples, também, já que há uma função pronta para realizar esta atividade no PHP, chamada **mysql_select_db**. Sua sintaxe básica é:

bool **mysql_select_db** (string **$nome_do_banco** [, resource **$sqlcon**])

O "**nome_do_banco**" refere-se exatamente ao banco de dados que queremos selecionar e o **sqlcon** é justamente o identificador retornado pela função mysql_connect. Caso tenhamos apenas uma conexão ativa, não é necessário especificar o **sqlcon**.

Mais uma vez, quando se usa apenas uma conexão, o indicador $sqlcon é dispensável, podendo ser feita a seleção simplesmente assim:

```
$resultado = mysql_select_db("nome_do_banco");
```

O retorno desta função (**bool**), armazenado acima na variável **$resultado** é um valor "falso" ou "verdadeiro", indicando o fracasso ou sucesso da seleção, respectivamente. A seleção pode falhar por diversos fatores, como por exemplo: banco de dados não existe, usuário não tem acesso àquele banco de dados etc.

```
if ( mysql_select_db("nome_do_banco") == TRUE ) {
        [ ... código de acesso ao banco de dados ... ]
        }
else {
        $msg_erro = mysql_error();
        print ("Erro no SQL: " . $msg_erro);
        }
```

Considerando que a seleção do banco de dados ocorreu com sucesso, é hora de realizar um acesso, seja ele de busca ou adição de dados.

1.3. Buscas (queries)

Conceitos Chave:

- mysql_query
  - * *resource* **mysql_query** (string $query [, resource $sqlcon])
- mysql_fetch_array
  - * *linha_de_tabela* **mysql_fetch_array** (*resource* $resultado)
  - * $oper = "...";
  - * $resultado = mysql_query($oper);
  - * $linha = mysql_fetch_array ($resultado);

O ponto fundamental de um banco de dados são as buscas (SELECTs), ou seja, a recuperação das informações. Entretanto, precisamos de informações para poder recuperá-las. Isso é feito através dos comendos de adição de dados (INSERTs).

Em ambos os casos, usaremos o PHP para enviar um comando ao banco de dados e a função que envia o comandos para o banco de dados e pega sua resposta é a função **mysql_query**. Sua sintaxe básica é:

resource **mysql_query** (string $query [, resource $sqlcon])

O retorno desta função é uma ponteiro para os resultados e, assim, deve ser armazenado. Mais uma vez, o $sqlcon é dispensável quando temos apenas uma conexão.

Assim, o uso mais comum desta função é:

```
$resultado = mysql_query ($oper);
```

Onde $oper é o texto do comando a ser enviado para o banco de dados MySQL.

Caso a busca encontre um erro, o valor de **$resultado** será o booleano "FALSE". Caso contrário, precisamos arrumar uma maneira de ler os dados da resposta, pois eles estão em um formato de tabela que o PHP não entende.

Para converter o formato lendo uma linha da tabela de cada vez, usamos a função **mysql_fetch_array**. Sua sintaxe básica é:

linha_de_tabela **mysql_fetch_array** ($resultado)

```
$resultado = mysql_query ($oper);
$linha = mysql_fetch_array ($resultado);
```

Os nomes das colunas da linha de tabela são os nomes das colunas do banco de dados. Assim, se nossa tabela do banco de dados tivesse colunas "cpf", "nome" e "sexo", poderíamos acessá-los assim:

```
$linha['cpf']
$linha['nome']
$linha['sexo']
```

Mas, o que ocorre quando não há mais linhas na tabela? Simples: o valor de $linha será igual a NULL.

Finalizada a consulta, deve ser desfeita a conexão com o Banco de Dados.

1.4. Desconexão

Conceitos Chave:
- mysql_close
    * *bool* **mysql_close**([resource $sqlcon])

Assim que todas as consultas são finalizadas, devemos desconectar do banco de dados para liberar a conexão. Isso pode ser feito usando a função **msql_close**, que tem a seguinte sintaxe:

bool **mysql_close** ([resource $identificador_sql])

O parâmetro de identificador especifica qual conexão você quer fechar (se houver mais de uma). O retorno é se a conexão foi fechada com sucesso ou não, indicando TRUE ou FALSE, respectivamente.

A forma comum de usar esta intrução é:

```
mysql_close();
```

2. TUTORIAL DE CADASTRO DE USUÁRIOS

Na aula passada implementamos um formulário de cadastro de usuários que executava a validação do lado do cliente e do lado do servidor. O último passo necessário é armazenar estes dados no banco. Para isso, iremos fazer algumas modificações no código da página mas, antes, precisamos preparar o banco de dados.

Relembremos: o formulário tinha três campos: Nome, CPF e sexo. Assim, para armazenar estes dados, precisamos de uma tabela do seguinte tipo:

CPF	Nome	Sexo
...	...	...

O CPF foi colocado em primeiro porque ele é exclusivo para cada usuário; assim, ele será chamado de CHAVE PRIMÁRIA (Primary Key ou PK).

CPF : PK	Nome	Sexo
...	...	...

O CPF será uma sequência de caracteres (CHAR) com 11 posições, conforme os limites do cadastro. O nome é um a sequência de caracteres de tamanho variável (VARCHAR) com até 40 caracteres, conforme limites do cadastro. O sexo é uma sequência de caracteres (CHAR) com 3 posições: **mas** ou **fem**:

CPF : PK, CHAR(11)	Nome : VARCHAR(40)	Sexo : CHAR(3)
...	...	...

Antes de mais nada, precisaremos criar um banco de dados que contenha essa tabela.

2.1. Criando o Banco de Dados

Para criar o banco de dados, entre no prompt do MySQL, solicitando esta função pelo menu do WAMP Server (ícone próximo ao relógio na barra de tarefas do Windows) ou indo até o diretório \BIN do MySQL e digitando:

```
mysql -uroot
```

Já dentro do prompt do MySQL, vamos primeiro criar o banco de dados, que terá o nome "siteweb":

```
CREATE DATABASE siteweb;
```

Agora, vamos autorizar um usuário "webuser" a usar este banco de dados, com permissão total, acessando pelo computador local, com password "webpass":

```
GRANT ALL PRIVILEGES ON siteweb.* TO 'webuser'@'localhost' IDENTIFIED BY 'webpass';
```

Agora, vamos selecionar o banco de dados que criamos e criar uma tabela:

```
USE siteweb;

CREATE TABLE cadastro (cpf CHAR(11) NOT NULL, nome VARCHAR(40), sexo CHAR(3), PRIMARY KEY(cpf));
```

Agora vamos inserir dados de teste neste banco de dados:

```
INSERT INTO cadastro VALUES ("01234567890", "Fulano", "mas");
```

Você pode verificar se está tudo ok com o select abaixo:

```
SELECT * FROM cadastro;
```

Verificando que todos os dados estão lá, basta apagar os dados e sair:

```
DELETE FROM cadastro;
EXIT
```

2.2. Armazenando dados do Formulário no Cadastro

Vamos, agora, alterar o arquivo **pag3.php**, das aulas anteriores, para armazenar os dados no Banco de Dados que acabamos de criar. Recuperando o arquivo **pag3.php**, ele deve estar assim:

pag3.php

```
<H1>Contato</H1>
<?PHP
        /* Pega dados no servidor */
        @$valida = $_POST['validar'];
        @$nome = $_POST['nome'];
        @$sexo = $_POST['sexo'];
        @$cpf = $_POST['cpf'];
        /* Inicialmente, $mostraForm vale true */
        $mostraForm = true;
        /* Executa validação e corrige o valor de $mostraForm, se for o caso */
        if ($valida==1) {
                include "veriform.php";
                $mostraForm = validaForm();
                }
/* Só imprime o HTML a seguir se $mostraForm for true! */
if ($mostraForm) {
?>
<FORM ACTION="?cont=3" METHOD="post" NAME="form1" ID="form1">
  <INPUT TYPE="hidden" NAME="validar" ID="validar" VALUE="1">
  <FIELDSET>
    <LEGEND>Informa&ccedil;&otilde;es Pessoais</LEGEND>
    Nome: <INPUT TYPE="text" MAXLENGTH="40" SIZE="40"
                <?PHP print ("VALUE=\"$nome\""); ?>
                NAME="nome" ID="nome" ><BR>
    <FIELDSET>
      <LEGEND>Sexo</LEGEND>
      <INPUT TYPE="radio" NAME="sexo"
                <?PHP if ($sexo!="fem") print ("CHECKED"); ?>
                ID="masc" VALUE="mas">Masculino
      <INPUT TYPE="radio" NAME="sexo"
                <?PHP if ($sexo=="fem") print ("CHECKED"); ?>
                ID="fem" VALUE="fem">Feminino
```

```
    </FIELDSET><BR>
  CPF: <INPUT TYPE="text" MAXLENGTH="11" SIZE="11"
            <?PHP print ("VALUE=\"$cpf\""); ?>
            NAME="cpf" ID="cpf"><BR>
  </FIELDSET>
 <INPUT TYPE="submit" VALUE="Enviar">
 </FORM>
<P ID="help"></P>
<?PHP
}
?>
```

Antes de continuarmos, faremos uma modificação neste código. Observem que os campos do formulário são obtidos do servidor neste arquivo **pag3.php** e, se vocês se lembram, eles eram **relidos** no arquivo **veriform.php**. Isso não faz muito sentido. Como já temos o valor dos campos em variáveis do PHP, ao invés de reler os campos na função validaForm(), vamos passar os campos para esta função, da seguinte forma:

pag3.php

```
<H1>Contato</H1>
<?PHP
    /* Pega dados no servidor */
    @$valida = $_POST['validar'];
    @$nome = $_POST['nome'];
    @$sexo = $_POST['sexo'];
    @$cpf = $_POST['cpf'];
    /* Inicialmente, $mostraForm vale true */
    $mostraForm = true;
    /* Executa validação e corrige o valor de $mostraForm, se for o caso */
    if ($valida==1) {
            include "veriform.php";
            $mostraForm = validaForm($nome, $cpf, $sexo);
            }
/* Só imprime o HTML a seguir se $mostraForm for true! */
if ($mostraForm) {
?>
<FORM ACTION="?cont=3" METHOD="post" NAME="form1" ID="form1">
 <INPUT TYPE="hidden" NAME="validar" ID="validar" VALUE="1">
 <FIELDSET>
  <LEGEND>Informa&ccedil;&otilde;es Pessoais</LEGEND>
  Nome: <INPUT TYPE="text" MAXLENGTH="40" SIZE="40"
            <?PHP print ("VALUE=\"$nome\""); ?>
            NAME="nome" ID="nome" ><BR>
  <FIELDSET>
   <LEGEND>Sexo</LEGEND>
   <INPUT TYPE="radio" NAME="sexo"
            <?PHP if ($sexo!="fem") print ("CHECKED"); ?>
            ID="masc" VALUE="mas">Masculino
   <INPUT TYPE="radio" NAME="sexo"
            <?PHP if ($sexo=="fem") print ("CHECKED"); ?>
            ID="fem" VALUE="fem">Feminino
   </FIELDSET><BR>
```

```
    CPF: <INPUT TYPE="text" MAXLENGTH="11" SIZE="11"
                <?PHP print ("VALUE=\"$cpf\""); ?>
                NAME="cpf" ID="cpf"><BR>
    </FIELDSET>
  <INPUT TYPE="submit" VALUE="Enviar">
  </FORM>
<P ID="help"></P>
<?PHP
}
?>
```

E, é claro, precisaremos modificar o arquivo **veriform.php**:

veriform.php

```
<?PHP
/* Valida dados de cadastro */
/* Entrada: Nome, CPF e Sexo */
function validaForm($usr_nome, $usr_cpf, $usr_sexo) {
        /* PHP 'pega dados' com o Servidor Web */
        $usr_nome = $_POST['nome'];
        $usr_sexo = $_POST['sexo'];
        $usr_cpf = $_POST['cpf'];

        /* Valida nome */
        if (strlen($usr_nome) < 5) print ("<P>O nome deve ter pelo menos 5 caracteres!</P>\n");
        /* Valida CPF */
        else if (strlen($usr_cpf) != 11) print ("<P>O cpf deve ter exatamente 11 caracteres!</P>\n");
        else if (is_numeric($usr_cpf) == FALSE) print ("<P>O cpf deve ser num&eacute;rico!</P>\n");
        else {
          /* Imprime mensagem de agradecimento */
          print ("<P>Obrigado por entrar em contato!</P>\n");
          /* Imprime Nome */
          print("<P>Nome: $usr_nome</P>\n");
          /* Imprime Sexo */
          print("<P>Sexo: $usr_sexo</P>\n");
          /* Imprime CPF */
          print("<P>CPF: $usr_cpf</P>\n");
          return false;
          }
        return true;
        }
?>
```

Isso feito, vamos modificar o nosso arquivo pag3.php para, caso os dados passem pela validação, que o arquivo **armazena.php** seja incluído e a função **guardaDados($cpf, $nome, $sexo)** seja chamada:

pag3.php

```
<H1>Contato</H1>
<?PHP
        /* Pega dados no servidor */
        @$valida = $_POST['validar'];
        @$nome = $_POST['nome'];
        @$sexo = $_POST['sexo'];
        @$cpf = $_POST['cpf'];
        /* Inicialmente, $mostraForm vale true */
        $mostraForm = true;
        /* Executa validação e corrige o valor de $mostraForm, se for o caso */
        if ($valida==1) {
                include "veriform.php";
                $mostraForm = validaForm($nome, $cpf, $sexo);
                /* Se $mostraForm for FALSE, é porque passou na validação */
                /* Então guarda dados no banco de dados! */
                if ($mostraForm == FALSE) {
                        include "armazena.php";
                        guardaDados($cpf, $nome, $sexo);
                        }
                }
/* Só imprime o HTML a seguir se $mostraForm for true! */
if ($mostraForm) {
?>
<FORM ACTION="?cont=3" METHOD="post" NAME="form1" ID="form1">
  <INPUT TYPE="hidden" NAME="validar" ID="validar" VALUE="1">
  <FIELDSET>
    <LEGEND>Informa&ccedil;&otilde;es Pessoais</LEGEND>
    Nome: <INPUT TYPE="text" MAXLENGTH="40" SIZE="40"
                <?PHP print ("VALUE=\"$nome\""); ?>
                NAME="nome" ID="nome" ><BR>
    <FIELDSET>
      <LEGEND>Sexo</LEGEND>
      <INPUT TYPE="radio" NAME="sexo"
                <?PHP if ($sexo!="fem") print ("CHECKED"); ?>
                ID="masc" VALUE="mas">Masculino
      <INPUT TYPE="radio" NAME="sexo"
                <?PHP if ($sexo=="fem") print ("CHECKED"); ?>
                ID="fem" VALUE="fem">Feminino
      </FIELDSET><BR>
    CPF: <INPUT TYPE="text" MAXLENGTH="11" SIZE="11"
                <?PHP print ("VALUE=\"$cpf\""); ?>
                NAME="cpf" ID="cpf"><BR>
    </FIELDSET>
  <INPUT TYPE="submit" VALUE="Enviar">
  </FORM>
<P ID="help"></P>
<?PHP
}
?>
```

Agora nossa página já está preparada para armazenar os dados. Precisamos, agora, criar o código do arquivo **armazena.php**, criando a função **guardaDados($cpf, $nome, $sexo)**. Iniciemos com a criação do arquivo armazena.php com a função vazia:

armazena.php

```
<?PHP
function guardaDados($cpf, $nome, $sexo) {
        }
?>
```

Deve-se iniciar conectando ao banco de dados, lembrando que o servidor é **localhost**, o nome de usuário é **webuser** e a senha é **webpass**. Vamos inserir a desconexão também:

armazena.php

```
<?PHP
function guardaDados($cpf, $nome, $sexo) {
        mysql_connect("localhost", "webuser", "webpass");
        mysql_close();
        }
?>
```

Isso está quase bom, mas precisamos verificar se a conexão ocorreu com sucesso:

armazena.php

```
<?PHP
function guardaDados($cpf, $nome, $sexo) {
        if ( mysql_connect("localhost", "webuser", "webpass") != FALSE ) {
                print ("<P>Conexão com banco de dados feita com sucesso!<P>\n");
                mysql_close();
                }
        else {
                print ("<P><STRONG>Erro</STRONG> de conexão com o banco de dados!</P>\n");
                print ("<P>" . mysql_error() . "</P>\n");
                }
        }
?>
```

Isso está quase bom, mas ainda teremos um "Warning" caso a conexão não consiga ser feita porque o servidor está fora do ar. Para previnir este problema, usamos a @:

armazena.php

```
<?PHP
function guardaDados($cpf, $nome, $sexo) {
        if ( @mysql_connect("localhost", "webuser", "webpass") != FALSE ) {
                print ("<P>Conexão com banco de dados feita com sucesso!<P>\n");
                mysql_close();
                }
```

```
        else {
                print ("<P><STRONG>Erro</STRONG> de conexão com o banco de dados!</P>\n");
                print ("<P>" . mysql_error() . "</P>\n");
                }
        }
?>
```

É claro que na versão final não iremos querer mostrar todos estes erros e mensagens, mas vamos deixá-las aí durante o desenvolvimento, para facilitar a identificação de erros.

Considerando que a conexão foi feita com sucesso, agora devemos selecionar o banco de dados **siteweb**:

armazena.php

```
<?PHP
function guardaDados($cpf, $nome, $sexo) {
        if ( @mysql_connect("localhost", "webuser", "webpass") != FALSE ) {
                print ("<P>Conexão com banco de dados feita com sucesso!<P>\n");
                if ( @mysql_select_db("siteweb") != FALSE ) {
                        print ("<P>Banco de dados selecionado com sucesso!<P>\n");
                        }
                else {
                        print ("<P><STRONG>Erro</STRONG> na seleção de banco!</P>\n");
                        print ("<P>" . mysql_error() . "</P>\n");
                        }
                mysql_close();
                }
        else {
                print ("<P><STRONG>Erro</STRONG> de conexão com o banco de dados!</P>\n");
                print ("<P>" . mysql_error() . "</P>\n");
                }
        }
?>
```

Com a seleção de banco de dados feita, falta agora **inserir** os dados. Anteriormente, no teste, inserimos dados da seguinte forma:

```
INSERT INTO cadastro VALUES ("01234567890", "Fulano", "mas");
```

Usaremos exatamente esta sintaxe, mas usaremos a função **mysql_query** para enviar esta requisição ao banco de dados, substituindo os dados pelos valores das variáveis:

armazena.php

```
<?PHP
function guardaDados($cpf, $nome, $sexo) {
        if ( @mysql_connect("localhost", "webuser", "webpass") != FALSE ) {
                print ("<P>Conexão com banco de dados feita com sucesso!<P>\n");
                if ( @mysql_select_db("siteweb") != FALSE ) {
```

```
                        print ("<P>Banco de dados selecionado com sucesso!<P>\n");
                        $comando = "INSERT INTO cadastro VALUES ('$cpf','$nome','$sexo')";
                        if (mysql_query($comando) != FALSE) {
                                print ("<P>Dados inseridos com sucesso!<P>\n");
                                }
                        else {
                                print ("<P><STRONG>Erro</STRONG> ao inserir dados!</P>\n");
                                print ("<P>" . mysql_error() . "</P>\n");
                                }
                        }
                else {
                        print ("<P><STRONG>Erro</STRONG> na seleção de banco!</P>\n");
                        print ("<P>" . mysql_error() . "</P>\n");
                        }
                mysql_close();
                }
        else {
                print ("<P><STRONG>Erro</STRONG> de conexão com o banco de dados!</P>\n");
                print ("<P>" . mysql_error() . "</P>\n");
                }
        }
?>
```

Experimente inserir o nosso amigo "Fulano", de CPF 01234567890 e sexo masculino. Veja que agora o sistema permite que ele seja inserido e todas as mensagens são de sucesso.

Se tentarmos inserir novamente a mesma pessoa, receberemos um erro: "Duplicate Entry", que significa que estamos tentando inserir uma nova linha com o mesmo CPF, o que é proibido no sistema como o fizemos.

Se quisermos que seja possível ALTERAR a linha anterior quando um mesmo CPF for digitado, devemos substituir o comando INSERT pelo comando REPLACE. A sintaxe é exatamente a mesma:

armazena.php

```
<?PHP
function guardaDados($cpf, $nome, $sexo) {
        if ( @mysql_connect("localhost", "webuser", "webpass") != FALSE ) {
                print ("<P>Conexão com banco de dados feita com sucesso!<P>\n");
                if ( @mysql_select_db("siteweb") != FALSE ) {
                        print ("<P>Banco de dados selecionado com sucesso!<P>\n");
                        $comando = "REPLACE INTO cadastro VALUES ('$cpf','$nome','$sexo')";
                        if (mysql_query($comando) != FALSE) {
                                print ("<P>Banco de Dados atualizado com sucesso!<P>\n");
                                }
                        else {
                                print ("<P><STRONG>Erro</STRONG> ao inserir dados!</P>\n");
                                print ("<P>" . mysql_error() . "</P>\n");
                                }
                        }
                else {
                        print ("<P><STRONG>Erro</STRONG> na seleção de banco!</P>\n");
```

```
                print ("<P>" . mysql_error() . "</P>\n");
                }
            mysql_close();
            }
        else {
            print ("<P><STRONG>Erro</STRONG> de conexão com o banco de dados!</P>\n");
            print ("<P>" . mysql_error() . "</P>\n");
            }
        }
?>
```

Vamos, agora, como teste, criar uma nova função **mostraDados()**, que irá imprimir todos os dados do banco de dados quando a atualização ocorrer com sucesso. A função vai considerar que a conexão já está aberta e que existe uma única conexão de banco de dados:

armazena.php

```
<?PHP
function guardaDados($cpf, $nome, $sexo) {
        if ( @mysql_connect("localhost", "webuser", "webpass") != FALSE ) {
            print ("<P>Conexão com banco de dados feita com sucesso!<P>\n");
            if ( @mysql_select_db("siteweb") != FALSE ) {
                print ("<P>Banco de dados selecionado com sucesso!<P>\n");
                $comando = "REPLACE INTO cadastro VALUES ('$cpf','$nome','$sexo')";
                if (mysql_query($comando) != FALSE) {
                    print ("<P>Banco de Dados atualizado com sucesso!<P>\n");
                    mostraDados();
                    }
                else {
                    print ("<P><STRONG>Erro</STRONG> ao inserir dados!</P>\n");
                    print ("<P>" . mysql_error() . "</P>\n");
                    }
                }
            else {
                print ("<P><STRONG>Erro</STRONG> na seleção de banco!</P>\n");
                print ("<P>" . mysql_error() . "</P>\n");
                }
            mysql_close();
            }
        else {
            print ("<P><STRONG>Erro</STRONG> de conexão com o banco de dados!</P>\n");
            print ("<P>" . mysql_error() . "</P>\n");
            }
        }

function mostraDados() {
        $comando = "SELECT cpf, nome, sexo FROM cadastro";
        $resultado = mysql_query($comando);
        if ($resultado == FALSE) {
            print ("<P><STRONG>Erro</STRONG> lendo banco de dados!</P>\n");
            print ("<P>" . mysql_error() . "</P>\n");
            }
        /* Aqui se conseguiu ler... pega linha! */
        $linha = mysql_fetch_array($resultado);
```

```
        print ("<P>CPF: " . $linha['cpf']);
        print (" Nome: " . $linha['nome']);
        print (" Sexo: " . $linha['sexo'] . "</P>\n");
        }
?>
```

Essa função funciona bem, mas mostra **apenas a primeira linha** da tabela. Para imprimir todas as linhas, é preciso chamar várias vezes a instrução **mysql_fetch_array**, até que ela retorne o valor NULL. Isso pode ser feito usando a instrução WHILE:

armazena.php

```
<?PHP
function guardaDados($cpf, $nome, $sexo) {
        if ( @mysql_connect("localhost", "webuser", "webpass") != FALSE ) {
                print ("<P>Conexão com banco de dados feita com sucesso!<P>\n");
                if ( @mysql_select_db("siteweb") != FALSE ) {
                        print ("<P>Banco de dados selecionado com sucesso!<P>\n");
                        $comando = "REPLACE INTO cadastro VALUES ('$cpf','$nome','$sexo')";
                        if (mysql_query($comando) != FALSE) {
                                print ("<P>Banco de Dados atualizado com sucesso!<P>\n");
                                mostraDados();
                                }
                        else {  print ("<P><STRONG>Erro</STRONG> ao inserir dados!</P>\n");
                                print ("<P>" . mysql_error() . "</P>\n");
                                }
                        }
                else {  print ("<P><STRONG>Erro</STRONG> na seleção de banco!</P>\n");
                        print ("<P>" . mysql_error() . "</P>\n");
                        }
                mysql_close();
                }
        else {  print ("<P><STRONG>Erro</STRONG> de conexão com o banco de dados!</P>\n");
                print ("<P>" . mysql_error() . "</P>\n");
                }
        }
function mostraDados() {
        $comando = "SELECT cpf, nome, sexo FROM cadastro";
        $resultado = mysql_query($comando);
        if ($resultado == FALSE) {
                print ("<P><STRONG>Erro</STRONG> lendo banco de dados!</P>\n");
                print ("<P>" . mysql_error() . "</P>\n");
                }
        /* Aqui se conseguiu ler... pega linha! */
        $linha = mysql_fetch_array($resultado);
        while ( $linha!=NULL ) {            /* Loop até que não haja mais linhas na tabela */
                print ("<P>CPF: " . $linha['cpf']);
                print (" Nome: " . $linha['nome']);
                print (" Sexo: " . $linha['sexo'] . "</P>\n");
                $linha = mysql_fetch_array($resultado);
                }
        }
?>
```

Para finalizar, iremos apenas eliminar as linhas de mensagem de sucesso, além de não mais mostrarmos todas as entradas do banco de dados, mostrando apenas a linha recém inserida:

armazena.php

```
<?PHP
function guardaDados($cpf, $nome, $sexo) {
        if ( @mysql_connect("localhost", "webuser", "webpass") != FALSE ) {
                print ("<P>Conexão com banco de dados feita com sucesso!<P>\n");
                if ( @mysql_select_db("siteweb") != FALSE ) {
                        print ("<P>Banco de dados selecionado com sucesso!<P>\n");
                        $comando = "REPLACE INTO cadastro VALUES ('$cpf','$nome','$sexo')";
                        if (mysql_query($comando) != FALSE) {
                                print ("<P>Banco de Dados atualizado com sucesso!<P>\n");
                                mostraDados($cpf);
                                }
                        else {  print ("<P><STRONG>Erro</STRONG> ao inserir dados!</P>\n");
                                print ("<P>" . mysql_error() . "</P>\n");
                                }
                        }
                else {  print ("<P><STRONG>Erro</STRONG> na seleção de banco!</P>\n");
                        print ("<P>" . mysql_error() . "</P>\n");
                        }
                mysql_close();
                }
        else {  print ("<P><STRONG>Erro</STRONG> de conexão com o banco de dados!</P>\n");
                print ("<P>" . mysql_error() . "</P>\n");
                }
        }

function mostraDados($cpf) {
        $comando = "SELECT cpf, nome, sexo FROM cadastro WHERE CPF = '$cpf'";
        $resultado = mysql_query($comando);
        if ($resultado == FALSE) {
                print ("<P><STRONG>Erro</STRONG> lendo banco de dados!</P>\n");
                print ("<P>" . mysql_error() . "</P>\n");
                }
        /* Aqui se conseguiu ler... pega linha! */
        $linha = mysql_fetch_array($resultado);
        while ( $linha!=NULL ) {                /* Loop até que não haja mais linhas na tabela */
                print ("<P>CPF: " . $linha['cpf']);
                print (" Nome: " . $linha['nome']);
                print (" Sexo: " . $linha['sexo'] . "</P>\n");
                $linha = mysql_fetch_array($resultado);
                }
        }
?>
```

3. TUTORIAL DE BUSCA (OPCIONAL)

Se quisermos implementar um campo de busca em uma página, uma boa solução é colocar todo o conteúdo em um banco de dados. Para este fim, o banco de dados mais simples deve conter:

id : PK, INT	titulo: VARCHAR(255)	texto : BLOB
...	...	...

O **id** é um identificador inteiro único para cada página. O **titulo** é um campo texto de tamanho até 255 caracteres, que conterá o título da página. O **texto** será o conteúdo da página (código HTML, inclusive). BLOB significa "Binary Language OBject", ou seja, qualquer tipo de dado que um computador possa armazenar.

3.1. Criando o Banco de Dados

Para criar o banco de dados, entre no prompt do MySQL, solicitando esta função pelo menu do WAMP Server ou indo até o diretório \BIN do MySQL e digitando:

```
mysql -uroot
```

Já dentro do prompt do MySQL, vamos primeiro criar o banco de dados, se ele já não existir:

```
CREATE DATABASE siteweb;
```

Agora, se ainda não fizemos isso, vamos autorizar um usuário "webuser" a usar este banco de dados, com permissão total acessando pelo computador local, com password "webpass":

```
GRANT ALL PRIVILEGES ON webpage.* TO 'webuser'@'localhost' IDENTIFIED BY webpass;
```

Agora, vamos selecionar o banco de dados que criamos e criar uma tabela:

```
USE siteweb;

CREATE TABLE webdata (id INT NOT NULL AUTO_INCREMENT, titulo VARCHAR (255), texto BLOB, PRIMARY KEY(id));
```

Agora vamos inserir uns dados neste banco de dados:

```
INSERT INTO webdata VALUES (NULL, "Pagina 1", "<P>Texto da pagina 1</P>");
INSERT INTO webdata VALUES (NULL, "Pagina 2", "<P>Texto da pagina 2</P>");
INSERT INTO webdata VALUES (NULL, "Outro titulo", "<P>Um texto completo</P>");
```

Você pode verificar se está tudo ok com o select abaixo:

```
SELECT * FROM webdata;
```

Verificando que todos os dados estão lá, basta sair do banco de dados:

```
EXIT
```

3.2. Acessando o banco de dados pela página Web de Busca

O código a seguir depende da existência de um formulário com um campo chamado "busca", onde o usuário terá digitado o texto a ser procurado, e que envie os dados ao programa **resultado_busca.php**, como indicado a seguir:

busca.html

```
<FORM METHOD="post" ACTION="resultado_busca.php">
        <INPUT TYPE="text" NAME="busca">
        <INPUT TYPE="submit" VALUE="Procurar">
</FORM>
```

O código de **resultado_busca.php** deve ter o seguinte aspecto (código simplificado, sem as verificações de erros):

resultado_busca.php

```
/* Pega o valor indicado pelo usuário na página para a busca. */
$busca = $_POST['busca'];
/* Conecta no Banco de Dados */
mysql_connect("localhost", "webuser", "webpass");
mysql_select_db("webdata");

/* Define o comando de busca */
$comando = "SELECT id, titulo, texto FROM webdata WHERE texto LIKE '%$busca%'";
$resultado = mysql_query ($comando);      /* Colhe os resultados */
$linha = mysql_fetch_array ($resultado);  /* Recupera a primeira linha da resposta */
while ($linha != NULL)                    /* Enquanto houver linhas na resposta */
        {
        /* Imprime os dados coletados */
```

```
                print ("ID: " . $linha['id'] . "<BR>\n");
                print ("T&iacute;tulo: " . $linha['titulo'] . "<BR>\n");
                print ("Texto: " . $linha['text'] . "\n");
                print ("<BR>\n");
                $linha = mysql_fetch_array ($resultado);
                }
        mysql_close();
```

Coloque estes arquivos no diretório do seu servido WAMP (C:\WAMP\WWW, por exemplo) e abra o formulário **busca.html** no navegador, com **http:\\localhost\busca.html**. Tente buscas diferentes, e veja os resultados. Dica: use as palavras:

texto
pagina
completo

3.3. Criando um programa que mostra as páginas que estão no Banco

Perceba que você pode usar este banco de dados para imprimir suas páginas. É possível fazer uma página da seguinte forma:

showpage.php

```
<?PHP
        /* Pega número da página, passado por Get (na URL) */
        @$cont = $_GET['cont'];
        if ($cont < 1 || $cont > 9999) $cont = 1;

        /* Conecta no Banco de Dados */
        mysql_connect("localhost", "webuser", "webpass");
        mysql_select_db("webdata");

        /* Define o comando de busca */
        $comando = "SELECT titulo, texto FROM webdata WHERE id = $cont";
        $resultado = mysql_query ($comando);        /* Colhe os resultados */
        $linha = mysql_fetch_array ($resultado);    /* Recupera a linha da resposta */
        /* Imprime a página */
        print ("T&iacute;tulo: " . $linha['titulo'] . "<BR>\n");
        print ("Texto: " . $linha['text'] . "<BR>\n");

        /* Fecha Banco de Dados */
        mysql_close();
?>
```

Coloque este arquivo no servidor e teste!

Algo interessante que pode ser feito é mostrar um menu, com link para todas as páginas do banco de dados. Para que isso seja feito, serão necessárias várias mudanças no arquivo já apresentado.

showpage.php

```
<?PHP
        /* Pega número da página, passado por Get (na URL) */
        @$cont = $_GET['cont'];
        if ($cont < 1 || $cont>9999) $cont = 1;

        /* Conecta no Banco de Dados */
        mysql_connect("localhost", "webuser", "webpass");
        mysql_select_db("webdata");

        /* Define o comando de busca */
        $comando = "SELECT titulo, texto FROM webdata WHERE id = $cont";
        $resultado = mysql_query ($comando);        /* Colhe os resultados */
        $linha = mysql_fetch_array ($resultado);    /* Recupera a linha da resposta */

        /* Armazena Informações em variáveis*/
        $titulo = $linha['titulo'];
        $texto = $linha['texto'];

        /* Imprime cabeçalho e título */
        print ("<HTML><HEAD></HEAD><BODY>\n");
        print ("<H1>$titulo</H1>\n");

        /* Imprime Menu */
        /* Busca todos os títulos e IDs de páginas existentes */
        $comando = "SELECT id, titulo FROM webdata";
        $resultado = mysql_query ($comando);    /* Colhe os resultados */
        $linha = mysql_fetch_array ($resultado);  /* Recupera uma linha */
        if ($linha != NULL) {
                print ("<UL>\n");
                while ($linha != NULL) {
                        /* Imprime link para cada linha recuperada */
                        print ("<LI>");                                    /* Inicia elemento de lista */
                        print ("<A HREF=\"?cont=" . $linha['id'] . "\">"); /* Abre link */
                        print ($linha['titulo']);                          /* Texto do link */
                        print ("</A>");                                    /* Fecha link */
                        print ("</LI>\n");                                 /* Fecha el. de lista */
                        $linha = mysql_fetch_array ($resultado);           /* Recupera outra linha */
                        }
                print ("</UL>\n");
                }

        /* Imprime Restante da Página */
        print ($texto);
        print ("</BODY></HTML>\n");

        /* Fecha Banco de Dados */
        mysql_close();
?>
```

Combinando os elementos transmitidos nas últimas aulas, é possível criar páginas extremamente interessantes. Na realidade, grande parte do conteúdo da web é feito exatamente da maneira apresentada neste curso!

4. EXERCÍCIO

a) Crie um site web* com pelo menos duas páginas, usando PHP e HTML. O conteúdo da página deve ser selecionado pelo valor do parâmetro **page**, passado pela URL (use GET!). O conteúdo deve vir do banco de dados, conforme apresentado anteriormente.

b) Implemente um mecanismo de busca que imprima quais páginas do seu site contém a palavra procurada, incluindo um link para a página.

(*) Você pode fazer esse exercício usando a página do projeto, se quiser.

5. BIBLIOGRAFIA

PHP: Manual do PHP, disponível em: < http://br.php.net/manual/pt_BR/ >. Visitado em 13/04/2009.

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 18: Programação Web Segura

Prof. Daniel Caetano

Objetivo: Apresentar e discutir problemas usuais onde técnicas de segurança são necessárias na programação Web e como solucioná-los.

Bibliografia: TERADA, 2000, TANENBAUM, 2003.

INTRODUÇÃO

Conceitos Chave:
- Programação Web
  * Como Fazer...
- Quais cuidados tomar?

Muitas informações já foram apresentadas sobre como construir páginas web atuais e com vários recursos de web dinâmica.

Entretanto, é preciso lembrar que a Internet é um meio bastante propenso a ataques por conta de crackers, sendo preciso tomar muito cuidado com a configuração/manipulação de bancos de dados por nossas páginas, visando impedir que usuários inescrupulosos tenham a possibilidade de realizar um ataque destrutivo.

Esta unidade apresenta uma breve visão sobre algumas destas questões.

1. PROTEÇÃO DOS DADOS

Conceitos Chave:
- Dados Transmitidos x Dados no Servidor
- Criptografia dos Dados Transmitidos
  * Nem sempre necessária
  * Secure Socket Layer (SSL)
  * Transport Layer Socket (TLS) => Novo nome do SSL
  * Implantados diretamente em servidores Web e Navegadores
  * Uso "automático" => Registro em Entidade (VeriSign).
  * Porta 80 x 443
  * Proteção Fim a Fim (não protege servidor)

- Segurança dos Dados no Servidor
  * SEMPRE necessária
  * Problemas:
    = Modificação/Destruição de bancos de dados
    = Acesso (sem autorização) a seções restritas de website
    = Slave Machine x Zombie Machine
    = Roubo de Informação

Existem dois tipos de segurança quando se fala em comunicação Web: uma é com relação à criptografia dos dados transmitidos, para que ninguém consiga identificar as informações trocadas entre cliente e servidor. A outra é relativa à segurança dos dados do servidor, quanto à sua integridade e confidencialidade.

1.1. Criptografia dos Dados

Existem várias maneiras de realizarmos a criptografia dos dados; entretanto, existe uma solução completa - e transparente - desenvolvida na década de 90 pela Netscape Communications, com o nome de Secure Socket Layer (SSL). Esta tecnologia evoluiu um pouco nos últimos anos e teve seu nome alterado para Transport Layer Socket (TLS), para ficar com um nome mais de acordo com sua função dentro dos modelos teóricos.

É uma grande vantagem usar estes sistemas prontos pois eles estão implementados em praticamente todos os Servidores Web e Navegadores existentes, bastando para seu uso a correta configuração do Servidor Web e o cadastro em um servidor de assinaturas digitais (VeriSign, por exemplo).

A programação do site web é feita normalmente, sendo a única diferença que a porta padrão de comunicação passa a ser a port 443, ao invés da porta 80.

Vale ressaltar que a criptografia dos dados transitados não impede que um cracker atinja o site que hospeda a página: ela apenas protege as informações transitadas "fim a fim".

1.2. Protegendo o Servidor contra Ataques

A segurança das informações transitadas nem sempre é necessária. Por esta razão, apenas em casos específicos é usado o esquema TLS. Por outro lado, a segurança da integridade e confidencialidade das informações do servidor é sempre necessária.

Através de um site web mal programado, um cracker pode modificar ou apagar bancos de dados inteiros, acessar seções restritas de site sem ter autorização, usar um determinado equipamento como escravo (para cometer crimes) e até mesmo roubar informações de arquivos existentes no disco rígido do Servidor Web.

Serão apresentados, nas próximas seções, alguns cuidados básicos que são necessários quando se trabalha com programação web.

2. CUIDADOS COM VARIÁVEIS GLOBAIS

Conceitos Chave:
- Variáveis do Servidor (estáticas)
- Variáveis GET (pela linha de endereço, ou seja, pela URL)
- Variáveis POST (por formulários de outra página)
- Exemplo: http://www.caetano.eng.br/exemplo.php?pagina=37
    * PHPs Antigos: $pagina
    * PHPs recentes: $pag = $_GET['pagina']
    * Exemplo: http://www.servidor.com.br/pagina.php?AUTH=1

O primeiro cuidado (e talvez um dos mais importantes) diz respeito às variáveis globais. Para entender isso, é preciso primeiro entender como funciona a passagem de variáveis de uma página web para outra. São basicamente três tipos:

- Variáveis do Servidor (estáticas)
- Variáveis GET (pela linha de endereço, ou seja, pela URL)
- Variáveis POST (por formulários de outra página)

Em algumas linguagens (notadamente versões antigas do PHP) qualquer um destes tipos de variáveis era definido automaticamente em um programa de uma página dinâmica. Por exemplo, considere a URL abaixo:

http://www.caetano.eng.br/exemplo.php?pagina=37

Neste caso temos um parâmetro enviado na forma GET (enviada pela URL). Ao chamar uma página desta forma, automaticamente o PHP geraria uma variável com o mesmo nome do parâmetro, ou seja, com o nome **$pagina**, que seria inicializada com o valor 37. O mesmo ocorreria se o parâmetro tivesse sido definido em um formulário (tipo POST) ou estivesse definido no servidor.

Esta característica, chamada de "Variáveis Globais", era bastante prática. Entretanto, isso propiciava uma baixa segurança. Mas por quê?

Imagine que, dentro de um software de uma página Web, o programador defina que a variável **$AUTH** identifica que um usuário está autenticado. Ou seja, quando o usuário entrar com um nome e senha adequados, o programa faz com que **$AUTH = 1**. Se o usuário der logoff ou errar ou não digitar o nome e login, o programa faz com que **$AUTH = 0**.

Ora, se um cracker descobrir o nome desta variável $AUTH (há alguns nomes bastante comuns) e chamar a URL da seguinte forma:

http://www.servidor.com.br/pagina.php?AUTH=1

Nós podemos ter um problema se o servidor estiver com variáveis globais ligadas! Neste caso, o servidor marcará o valor da variável $AUTH como sendo 1, e o cracker terá acesso à região em que apenas usuários autorizados possuem acesso, mesmo sem ter realizado login algum!

Por esta razão, a maioria das linguagens web permite **desligar** as variáveis globais e, em alguns casos (como as versões recentes do PHP) este comportamento vem desligado "de fábrica". É por isso que, quando estudamos leitura de variáveis da URL, foi usando o método $_GET:

$variavel = $_GET['pagina'];

Que permitia ler o valor do parâmetro **pagina** na variável **$variavel**, em uma URL como a apresentada abaixo:

http://www.caetano.eng.br/exemplo.php?pagina=37

Isso significa que só serão transformados em variáveis os parâmetros que quisermos, impedindo que alguém defina um valor de uma variável qualquer.

Desta forma, mesmo que o invasor descubra que a variável $AUTH é usada para definir se usuário já concluiu o login com sucesso, ele **não** vai conseguir mudar o valor desta variável usando uma linha como esta:

http://www.servidor.com.br/pagina.php?AUTH=1)

Pois em nossa página **não existirá** esta linha:

$AUTH = $_GET['AUTH'];

Pois não queremos que o valor desta variável seja mudado pela URL!

3. CUIDADOS COM VARIÁVEIS DE SESSÃO

Conceitos Chave:
- Armazenam dados da conexão atual
- Hashs Reversíveis
    * Exemplo
- Dados sensíveis => BANCO DE DADOS NO SERVIDOR!

Uma forma comum de armazenar informações de login e outros via Web é através do uso da URL. Entretanto, isto torna o sistema muito vulnerável, por informações importantes sobre o funcionamento interno seriam reveladas por uma simples inspeção da URL.

Uma maneira de evitar este problema é codificar de uma forma complexa a informação ao ser colocada na URL, "comprimindo-a" em um número chamado "HASH reversível". Este tipo de variável, que também pode ser armazenada em um cookie, também é chamada de Variável de Sessão, sendo um número único que define um determinado acesso de um usuário.

Por exemplo: se quisermos guardar a data em um único número, podemos fazer o seguinte:

```
$session = $ano * 500 + $mes * 35 + $dia;
```

Sabendo que $dia nunca terá um valor maior que 31, o fato de multiplicarmos o número do mês por 35 permitirá que possamos separar o número de meses do número final. Igualmente, 35*12+31 dá um total de 451 e, multiplicando o ano por 500, isolaremos este número do mês e dia.

Assim, caso definamos a data 15 de novembro de 1889 (que data é essa?), teremos:

```
$dia = 15;
$mes = 11;
$ano = 1889;
$session = $ano * 500 + $mes * 35 + $dia;
```

Ao imprimir o valor de $session, teremos:

1889*500 + 11*35 + 15 = 944500 + 385 + 15 = **944900**

Para reverter, basta usar operações de resto de divisão inteira e resto divisão:

```
// $ses vai conter $ano*500
$ses = $session - ($session % 500);
$ano = $ses/500;
// $ses vai conter $mes*35+dia
$ses = $session - $ses;
$dia = $ses%35;
// $ses vai conter $mes*35
$ses = $ses - $dia;
$mes = $ses/35;
```

4. CUIDADOS NO ACESSO AO BANCO DE DADOS

Conceitos Chave:
- SQL Injection
- SELECT * FROM usernames WHERE name LIKE "$nome";
- $nome = AAAAA"; DROP DATABASE; SELECT "A
- Resultado:
  SELECT * FROM usernames WHERE name LIKE "AAAAA";
  DROP DATABASE;
  SELECT "A";
- Eliminar aspas / Substituir aspas / URL Encode

Em praticamente todo web site dinâmico, existe um uso extensivo de um SGBD. Embora este uso seja muito prático e interessante, são necessários alguns cuidados para evitar problemas com possíveis ataques.

Um ataque comum a banco de dados é aquele que visa apagar todo seu conteúdo. Mais grave ainda, pode-se ter um ataque que revele todas as informações do banco de dados ao atacante. Se estas informações forem sensíveis e não criptografadas, tal revelação não é considerada aceitável.

A principal fonte de problemas ao se programar bancos de dados está no uso de variáveis dentro de queries SQL onde existem aspas (sejam elas simples ou duplas). Por exemplo:

```
SELECT * FROM usernames WHERE name LIKE "$nome";
```

Se o invasor consegue adulterar o conteúdo de $nome para algo como:

```
AAAAA"; DROP DATABASE; SELECT "A
```

Observe como será composto o query:

```
SELECT * FROM usernames WHERE name LIKE "AAAAA";
DROP DATABASE;
SELECT "A";
```

Apesar de parecer algo fora do comum, é um ataque bastante praticado e seu nome é "SQL Injection" (Injeção SQL). E, para piorar, muitos sites não estão preparados para evitar este tipo de "truque". Mas como se proteger disso?

A primeira forma é não permitir "aspas" nos valores das variáveis que são usadas internamente às queries. Para isso, basta criar uma função simples que remova as aspas de

qualquer variável que seja usada dentro das queries. Algumas linguagens até mesmo fornecem este tipo de função pronta.

A segunda forma, usada quando as aspas são necessárias no texto, é substituir as aspas por algum outro caractere ao inserir o texto no banco de dados, lembrando de converter de volta, quando futuramente o banco de dados for lido. Uma forma alternativa é o uso de "URL Encode" nos textos, que converte todos os caracteres especiais (incluindo aspas) para a forma do HTML, que também não causará problemas com as queries. Muitas linguagens também vêm com funções para converter o texto de uma variável na sua forma "URL Encoded".

5. CUIDADOS NO USO DE INCLUDES

Conceitos Chave:
- Repetições nas páginas
  * include "nome_da_pagina.php"
  * include "pagina".$page.".php"
- Redefinindo $page => adicionar qualquer código à pagina
  * Código externo rodando *no servidor*
- Desabilitar inclusão de páginas de outro servidor
- Verificar link antes de incluir seu conteúdo

Quando são criadas páginas dinâmicas, muitos dos trechos de código e/ou conteúdo podem ser repetidos em vários lugares. Nestes casos, estes trechos são gravados em arquivos específicos e são adicionados nas páginas necessárias usando as diretivas "include":

include "nome_da_pagina.php"

Esse recurso pode ser combinado com o uso de variáveis. Por exemplo, se uma variável $page diz o número da página atual, é possível "incluir" na página atual o conteúdo do arquivo da página pagina1.php com a seguinte instrução (exemplo em PHP):

include "pagina" . $page . ".php";

Entretanto, mais uma vez, se o invasor tiver acesso à modificar o conteúdo da variável $page de alguma maneira (se ela for definida na URL, por exemplo), ele pode conseguir incluir código malicioso em sua página, que será executado como se fosse parte da página original.

Este é um tipo de ataque muito comum, onde normalmente o invasor faz com que sua página inclua código de alguma outra página (criada por ele, em outro servidor). Assim, uma forma bastante eficaz contra este tipo de ataque é através de medidas para desabilitar a inclusão de conteúdo de servidores externos ao que está executando a página.

De qualquer forma, isso nem sempre é possível, já que algumas páginas podem precisar incluir conteúdos legítimos de sites externos. Nestes casos, é preciso tomar um cuidado adicional, criando um sistema que verifica o link a ser incluído, para que possam ser excluídos links maliciosos.

6. CAPTCHAS

Conceitos Chave:
- Palavras alfanuméricas distorcidas
- Evitar SPAM / Múltiplos cadastros automatizados etc
- Uso com variável de sessão em um DB, para verificação
  * Variável de Sessão / Texto do Captcha
  * Obriga leitura da imagem
- Evolução dos Captchas:
  * Captchas gerados por seres humanos (fotos)
    = Quantos leões existem de boca aberta?
  * "Gatos"

A maioria das pessoas que usam a Internet já viram um captcha, mas poucos sabem que aquilo se chama "captcha". Captchas são aquelas pequenas "palavras" alfanuméricas, apresentadas em caracteres distorcidos, que alguns serviços solicitam quando tentamos acessá-los:

No exemplo, o conteúdo é "ADCM". A finalidade dos captchas é impedir que programas automáticos sejam usados para criar contas em sites, enviar mensagens SMS e outros, impedindo o uso destes serviços para divulgação de SPAM, por exemplo.

Os captchas são, normalmente, gerados como uma seqüência aleatória de caracteres, escritos em um pedaço de imagem usando alguma linguagem como PHP e, posteriormente, esta imagem é distorcida, com algumas linhas movimentadas, pontos coloridos aleatórios desenhados, retas traçadas por sobre o texto, filtros de embaçamento de imagem, etc.

Antes de ser enviada a imagem para o usuário, é também estabelecida uma variável de sessão aleatória e, em um banco de dados local, é indicado o texto do captcha para aquela variável de sessão:

Variável de Sessão	Texto do Captcha
6a786e65fb55a4667dea234f	ADCM

Isso é feito desta forma para que, quando o usuário digitar o texto e enviar pelo formulário, seja possível verificar no banco de dados se o valor apresentado e o valor digitado batem. Esta é a única forma 100% segura para que não seja possível identificar o texto sem precisar analisar a figura.

8. CONTROLE DE ACESSO EM BANCO DE DADOS (OPCIONAL)

Conceitos Chave:

- Controle importante: leitura e escrita no BD
- SGBDs usuais: controle de acesso integrado
  - * Uso de banco de dados específico
  - * Consultado sempre que acesso ao BD é feito
- Instalação inicial
  - * Apenas root: senha?
- Primeiros passos:
  - * Colocar senha complexa para o root
  - * Configurá-lo apenas para acesos local (ou SSH)
  - * Criar usuários restritos para as funções normais
- Controle de Acesso SQL-97

```
* GRANT tipo_de_privilégio [lista_de_colunas] [, tipo_de_privilégio [lista_de_colunas] ...]
        ON { nome_da_tabela | * | *.* | nome_do_banco.* }
        TO nome_do_usuário [IDENTIFIED BY 'password']
                [, nome_do_usuário [IDENTIFIED BY 'password'] ...]
        [WITH GRANT OPTION]
* REVOKE tipo_de_privilégio [lista_de_colunas] [, tipo_de_privilégio [lista_de_colunas] ...]
        ON { nome_da_tabela | * | *.* | nome_do_banco.* }
        FROM nome_do_usuário [, nome_do_usuário ...]
```

- usuário@nome.do.host x usuário@numero_do_host
- usuario@"%.usp.br" x usuário@"200.168.%"

Um aspecto muito importante da segurança da informação é relativo ao acesso das informações contidas em um banco de dados, seja este acesso para a leitura ou, mais importante, para a escrita.

Praticamente todo Sistema Gerenciador de Banco de Dados (SGBD) contém um sistema de controle de acesso integrado. As permissões de acesso são registradas em um banco de dados específico, o qual é checado sempre que uma requisição de mudança ou seleção em qualquer dos bancos de dados gerenciados é solicitada.

Na instalação padrão, muitos dos SGBDs são vulneráveis, existindo apenas uma conta de usuário (root), com todas as permissões possíveis e sem qualquer tipo e senha. A primeira providência a ser tomada é a de configurar este usuário de uma forma muito mais restritiva, colocando uma senha longa e também permitindo que o usuário "root" só possa gerenciar

bancos de dados a partir da máquina local, ou seja, para que alguém possa realizar operações como usuário root, precisa estar fisicamente na máquina do servidor de banco de dados (ou conectado a ele por Telnet/SSH).

Feito isso, o próximo passo é criar "contas" específicas para cada uso e usuário do SGBD, estabelecendo restrições rígidas a partir do princípio que "tudo é proibido a menos que expressamente permitido".

1.1. Acesso em Banco de Dados SQL

Nos bancos de dados que seguem o padrão SQL-97, o controle de acessos é configurado com os comandos GRANT e REVOKE. O comando GRANT concede permissão de acesso a um usuário e o comando REVOKE revoga a permissão de acesso a um usuário.

A sintaxe do comando GRANT é:

```
GRANT tipo_de_privilégio [lista_de_colunas] [, tipo_de_privilégio [lista_de_colunas] ...]
    ON { nome_da_tabela | * | *.* | nome_do_banco.* }
    TO nome_do_usuário [IDENTIFIED BY 'password']
        [, nome_do_usuário [IDENTIFIED BY 'password'] ...]
    [WITH GRANT OPTION]
```

Ou seja:

1) Indica-se o comando GRANT.
2) Indica-se o tipo de privilégio e para quais colunas este privilégio será aplicado. Os tipos de privilégio são:

ALL PRIVILEGES	FILE	RELOAD
ALTER	INDEX	SELECT
CREATE	INSERT	SHUTDOWN
DELETE	PROCESS	UPDATE
DROP	REFERENCES	USAGE

3) Indica-se em que tabela estão aquelas colunas, com a diretiva ON.
4) Indica-se o usuário autorizado, com a diretiva TO. Não é obrigatório, mas é interessante que cada usuário seja identificado por um password, com a diretiva IDENTIFIED BY.
5) Finalmente, é indicado se aquele usuário terá permissão de autorizar outros usuários as ações às quais ele é permitido, com a diretiva WITH GRANT OPTION.

A sintaxe do comando REVOKE é mais simples:

```
REVOKE tipo_de_privilégio [lista_de_colunas] [, tipo_privilégio [lista_de_colunas] ...]
    ON { nome_da_tabela | * | *.* | nome_do_banco.* }
    FROM nome_do_usuário [, nome_do_usuário ...]
```

1) Indica-se o comando REVOKE.

2) Indica-se o tipo de privilégio que será removido e de quais colunas.

3) Indica-se em que tabela estão aquelas colunas, com a diretiva ON.

4) Indica-se o usuário de quem estão sendo revogadas as permissões, com a diretiva FROM.

Alguns SGBDs permitem que um usuário fique restrito a um único host (equipamento de acesso). Isto é alcançado com algumas modificações no nome do usuário, usando a seguinte sintaxe:

nome_do_usuário@nome_do_host

ou

nome_do_usuário@número_do_host

Se o nome do usuário ou o nome do host contiverem caracteres especiais, como por exemplo o "-" ou o "%", eles devem estar entre aspas. O caractere "%" tem o significado de "qualquer elemento", como por exemplo:

"144.155.166.%"

Se refere a qualquer host de 144.155.166.0 a 144.155.166.255. Da mesma forma:

"%.usp.br"

Se refere a qualquer host terminado por .usp.br, como por exemplo "ftp.usp.br", "www.usp.br", "sistemas.poli.usp.br" etc.

9. BIBLIOGRAFIA

TANENBAUM, A. S. **Redes de Computadores**. Editora Campus, 2003.

TERADA, R. **Segurança de Dados:** Criptografia em Redes de Computador. São Paulo: Edgard Bücher, 2000.

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*

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.

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
ondbclick	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"
clip	Determina o formato de um elemento
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