

INSTITUTO FEDERAL SUL-RIO-GRANDENSE

UNIVERSIDADE ABERTA DO BRASIL

Programa de Fomento ao Uso das

TECNOLOGIAS DE COMUNICAÇÃO E INFORMAÇÃO NOS CURSOS DE GRADUAÇÃO - TICS

TICS

LINGUAGEM DE PROGRAMAÇÃO WEB

Anubis Graciela de Moraes Rossetto

Ministério da
Educação



Copyright© 2011 Universidade Aberta do Brasil
Instituto Federal Sul-rio-grandense

Apostila de Linguagem de Programação Web

ROSSETTO, Anubis Graciela de Moraes

2012/1

Produzido pela Equipe de Produção de Material Didático da
Universidade Aberta do Brasil do Instituto Federal Sul-rio-grandense

TODOS OS DIREITOS RESERVADOS

PRESIDÊNCIA DA REPÚBLICA

Dilma Rousseff

PRESIDENTE DA REPÚBLICA FEDERATIVA DO BRASIL

MINISTÉRIO DA EDUCAÇÃO

Fernando Haddad

MINISTRO DO ESTADO DA EDUCAÇÃO

Luiz Cláudio Costa

SECRETÁRIO DE EDUCAÇÃO SUPERIOR - SESU

Eliezer Moreira Pacheco

SECRETÁRIO DA EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA

Luís Fernando Massonetto

SECRETÁRIO DA EDUCAÇÃO A DISTÂNCIA – SEED

Jorge Almeida Guimarães

PRESIDENTE DA COORDENAÇÃO DE APERFEIÇOAMENTO DE PESSOAL DE NÍVEL SUPERIOR - CAPES

**INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E
TECNOLOGIA SUL-RIO-GRANDENSE [IFSUL]**

Antônio Carlos Barum Brod

REITOR

Daniel Espírito Santo Garcia

PRÓ-REITOR DE ADMINISTRAÇÃO E DE PLANEJAMENTO

Janete Otte

PRÓ-REITORA DE DESENVOLVIMENTO INSTITUCIONAL

Odeli Zanchet

PRÓ-REITOR DE ENSINO

Lúcio Almeida Hecktheuer

PRÓ-REITOR DE PESQUISA, INOVAÇÃO E PÓS-GRADUAÇÃO

Renato Louzada Meireles

PRÓ-REITOR DE EXTENSÃO

IF SUL-RIO-GRANDENSE

CAMPUS PELOTAS

José Carlos Pereira Nogueira

DIRETOR-GERAL DO CAMPUS PELOTAS

Clóris Maria Freire Dorow

DIRETORA DE ENSINO

João Róger de Souza Sastre

DIRETOR DE ADMINISTRAÇÃO E PLANEJAMENTO

Rafael Blank Leitzke

DIRETOR DE PESQUISA E EXTENSÃO

Roger Luiz Albernaz de Araújo

CHEFE DO DEPARTAMENTO DE ENSINO SUPERIOR

IF SUL-RIO-GRANDENSE

DEPARTAMENTO DE EDUCAÇÃO A DISTÂNCIA

Luis Otoni Meireles Ribeiro

CHEFE DO DEPARTAMENTO DE EDUCAÇÃO A DISTÂNCIA

Beatriz Helena Zanotta Nunes

COORDENADORA DA UNIVERSIDADE ABERTA DO BRASIL – UAB/IFSUL

Marla Cristina da Silva Sopena

COORDENADORA ADJUNTA DA UNIVERSIDADE ABERTA DO BRASIL – UAB/IFSUL

Cinara Ourique do Nascimento

COORDENADORA DA ESCOLA TÉCNICA ABERTA DO BRASIL – E-TEC/IFSUL

Ricardo Lemos Sainz

COORDENADOR ADJUNTO DA ESCOLA TÉCNICA ABERTA DO BRASIL – E-TEC/IFSUL

IF SUL-RIO-GRANDENSE

UNIVERSIDADE ABERTA DO BRASIL

Beatriz Helena Zanotta Nunes

COORDENADORA DA UNIVERSIDADE ABERTA DO BRASIL – UAB/IFSUL

Marla Cristina da Silva Sopena

COORDENADORA ADJUNTA DA UNIVERSIDADE ABERTA DO BRASIL – UAB/IFSUL

Mauro Hallal dos Anjos

GESTOR DE PRODUÇÃO DE MATERIAL DIDÁTICO

**PROGRAMA DE FOMENTO AO USO DAS TECNOLOGIAS
DE COMUNICAÇÃO E INFORMAÇÃO NOS CURSOS DE
GRADUAÇÃO –TICS**

Raquel Paiva Godinho

GESTORA DO EDITAL DE TECNOLOGIAS DE INFORMAÇÃO E COMUNICAÇÃO – TICS/IFSUL

Ana M. Lucena Cardoso

DESIGNER INSTRUCIONAL DO EDITAL TICS

Lúcia Helena Gadret Rizzolo

REVISORA DO EDITAL TICS

EQUIPE DE PRODUÇÃO DE MATERIAL DIDÁTICO – UAB/IFSUL

Lisiane Corrêa Gomes Silveira
GESTORA DA EQUIPE DE DESIGN

Denise Zarnottz Knabach
Felipe Rommel
Helena Guimarães de Faria
Lucas Quaresma Lopes
Tabata Afonso da Costa
EQUIPE DE DESIGN

Catiúcia Klug Schneider
GESTORA DE PRODUÇÃO DE VÍDEO

Gladimir Pinto da Silva
PRODUTOR DE ÁUDIO E VÍDEO

Marcus Freitas Neves
EDITOR DE VÍDEO

João Eliézer Ribeiro Schaun
GESTOR DO AMBIENTE VIRTUAL DE APRENDIZAGEM

Giovani Portelinha Maia
GESTOR DE MANUTENÇÃO E SISTEMA DA INFORMAÇÃO

Carlo Camani Schneider
Efrain Becker Bartz
Jeferson de Oliveira Oliveira
Mishell Ferreira Weber
EQUIPE DE PROGRAMAÇÃO PARA WEB

SUMÁRIO



GUIA DIDÁTICO	9
UNIDADE A - INTRODUÇÃO AO DESENVOLVIMENTO PARA WEB	13
A internet e a World Wide Web	15
Linguagens de Programação para Web client-side e server-side	19
Tendências Web	21
Atividades	23
Introdução ao HTML	24
Atividades	33
UNIDADE B - LINGUAGEM DO LADO CLIENTE	37
Introdução a linguagem JavaScript	39
Objetos Javascript	47
Eventos Javascript	51
Funções	52
Atividades - parte 1	53
Arrays	59
Strings	56
Data e Hora	57
Atividades - parte 2	58
Objeto Window	63
Objeto Document	65
Solução passo-a-passo	66
Atividades - parte 3	69
UNIDADE C - INTRODUÇÃO A LINGUAGEM PHP	75
Linguagem PHP	77
Características da linguagem	77
Ferramentas necessárias	77
Estrutura da linguagem	79
Atividades - parte 1	89
Arrays	90
Strings	93
Datas	94
Atividades - parte 2	98
Funções	101
Atividades - parte 3	106
Sessões	107
Cookies	109
Atividades - parte 4	112
UNIDADE D - LINGUAGEM PHP COM ACESSO A BANCO DE DADOS	113
PHP com Banco de Dados	115
Atividades - parte 1	122
Área administrativa	123
Atividades - parte 2	134
Autenticação de usuário para área administrativa	135
Atividades - parte 3	140

APRESENTAÇÃO

Prezado(a) aluno (a),

Bem-vindo (a) ao espaço de estudo da Disciplina de Linguagem de Programação de Programação para Web.

O desenvolvimento de aplicações para web evidenciou uma significativa expansão nos últimos anos, possibilitando que os seus usuários tenham acesso a aplicações a partir do navegador de uma máquina com acesso à internet. A crescente aplicação nesta área é originada, sobretudo, dos avanços obtidos nas tecnologias de rede e da redução dos custos inerentes às tecnologias. Dada a dinâmica intrínseca da área e a importância que tem alcançado nos mais diferentes segmentos, a demanda por aplicações para esse ambiente tem crescido sobremaneira, ao mesmo tempo em que crescem as necessidades de resolução de problemas cada vez mais complexos.

O desenvolvimento de aplicações para a ambiente Web envolve a utilização de diferentes tecnologias, a fim de se alcançar um resultado de qualidade com recursos atuais e importantes para o usuário.

Nesta disciplina, serão desenvolvidos conceitos com utilização de linguagens de programação adequadas à criação de aplicações voltadas para Web. Para tanto, serão utilizadas linguagens de programação client-side e server-side com acesso a banco de dados.

Nas unidades, serão abordados os seguintes conteúdos: Linguagem HTML: confecção e formatação de páginas, formulários com métodos GET e POST; recursos da linguagem JavaScript: estruturas de controle, funções e eventos, validação de formulários, manipulação de janelas; recursos da linguagem PHP: estruturas de controle, manipulação de arrays, strings e datas, funções, sessões e cookies; desenvolvimento com banco de dados: conexão, consultas, inserção, alteração, exclusão e controle de acesso de usuário.

Esperamos que, através dos conteúdos e das atividades propostas, você possa estabelecer subsídios para a compreensão dos recursos de programação para a Web. E, para tal, você pode contar com toda a equipe.

Bom trabalho!

Objetivo Geral

Ao final desta disciplina, o aluno será capaz de atuar no desenvolvimento de aplicações para a Web, empregando tecnologias emergentes, visando suprir as necessidades do mundo do trabalho.

Habilidades

- Reconhecer as tecnologias e ferramentas disponíveis para o desenvolvimento para web.
- Utilizar linguagens client-side e server-side no desenvolvimento de aplicações para web.
- Compreender, utilizar e controlar formulários em aplicações Web.
- Explorar os recursos de uma linguagem de script para validação de formulários, manipulação de janelas, uso de funções e eventos.

- Utilizar os recursos da linguagem PHP, como estruturas condicionais e de repetição, manipulação de arrays, strings e datas.
- Organizar o script utilizando a modularização e funções permitindo a reutilização do código.
- Explorar os recursos do uso de sessões e cookies com PHP.
- Criar aplicações com o uso de uma linguagem do lado servidor (PHP) com acesso a banco de dados.

Metodologia

A disciplina será desenvolvida em 100 horas através do Ambiente Virtual de Aprendizado Moodle, onde serão disponibilizados materiais a serem estudados para subsidiar a aprendizagem. O Moodle será o canal de comunicação direto entre discentes e tutores, com as seguintes possibilidades de interação:

- Disponibilizar aos discentes as tarefas a serem realizadas.
- Publicar os materiais de apoio e de leitura complementar.
- Acompanhar o desempenho dos discentes em relação às atividades propostas.
- Interagir com a turma através de fórum de discussão, salas de chat e correio eletrônico.
- Acessar e avaliar as tarefas realizadas pelos discentes.
- Estimular o trabalho cooperativo entre os discentes.
- Promover o estudo autônomo.
- Acompanhar a frequência de acesso ao ambiente pelos discentes.
- Acessar links interessantes e relacionados ao curso.

Avaliação

A avaliação do desenvolvimento e envolvimento do discente em todas as unidades curriculares considerará os seguintes elementos:

- A participação nas aulas à distância, através das ferramentas de comunicação do ambiente virtual.
- A realização e a entrega das atividades solicitadas, observando a relevância e pertinência aos conteúdos abordados e solicitados no trabalho.
- A avaliação final e presencial.

Programação

Primeira Semana

As atividades a serem desenvolvidas na primeira semana são:

1. Fórum: Apresentação do professor, da disciplina e questões gerais.
2. Leitura e estudo do conteúdo: Introdução ao Desenvolvimento para Web.
3. Participação do Fórum de discussão proposto pelo professor formador.

Segunda Semana

As atividades a serem desenvolvidas na segunda semana são:

1. Leitura e estudo do conteúdo: Introdução ao HTML.
2. Realização da atividade: Unidade A – Introdução ao HTML (exercícios).
3. Participação em Chat em horário marcado pelo professor formador para discutir questões relativas aos exercícios propostos.

Terceira Semana

As atividades a serem desenvolvidas na terceira semana são:

1. Leitura e estudo do conteúdo: Introdução à linguagem JavaScript – Parte 1.
2. Realização da atividade: Unidade B – Introdução à linguagem JavaScript (exercícios parte 1).
3. Participação em aula remotamente ministrada via Webconferência.

Quarta Semana

As atividades a serem desenvolvidas na quarta semana são:

1. Leitura e estudo do conteúdo: Introdução à linguagem JavaScript - Parte 2.
2. Realização da atividade: Unidade B – Introdução à linguagem JavaScript (exercícios parte 2).
3. Participação do Fórum de discussão proposto pelo professor formador.

Quinta Semana

As atividades a serem desenvolvidas na quinta semana são:

1. Leitura e estudo do conteúdo: Introdução à linguagem JavaScript - Parte 3.
2. Realização da atividade: Unidade B – Introdução à linguagem JavaScript (exercícios parte 3).
3. Participação em Chat em horário marcado pelo professor formador para discutir questões relativas aos exercícios propostos.

Sexta Semana

As atividades a serem desenvolvidas na sexta semana são:

1. Leitura e estudo do conteúdo: Introdução à linguagem PHP – parte 1.
2. Assistir a vídeo: Instalação e configuração de ferramentas.
3. Realização da atividade: Unidade C – Introdução à linguagem PHP (exercícios parte 1).
4. Participação do Fórum de discussão proposto pelo professor formador.

Sétima Semana

As atividades a serem desenvolvidas na sétima semana são:

1. Leitura e estudo do conteúdo: Introdução à linguagem PHP – parte 2.
2. Realização da atividade: Unidade C – Introdução à linguagem PHP (exercícios parte 2).
3. Participação do Fórum de discussão proposto pelo professor formador.

Oitava Semana

As atividades a serem desenvolvidas na oitava semana são:

1. Leitura e estudo do conteúdo: Introdução à linguagem PHP – parte 3.
2. Realização da atividade: Unidade C – Introdução à linguagem PHP (exercícios parte 3).
3. Participação em Chat em horário marcado pelo professor formador, para discutir questões relativas aos exercícios propostos.

Nona Semana

As atividades a serem desenvolvidas na nona semana são:

1. Leitura e estudo do conteúdo: Introdução à linguagem PHP – parte 4.
2. Realização da atividade: Unidade C – Introdução à linguagem PHP (exercícios parte 4).
3. Participação do Fórum de discussão proposto pelo professor formador.

Décima Semana

As atividades a serem desenvolvidas na décima semana são:

1. Leitura e estudo do conteúdo: Linguagem PHP com acesso a banco de dados – Parte 1.
2. Assistira vídeo: Instalação e configuração de ferramentas .
3. Realização da atividade: Unidade D – Linguagem PHP com acesso a banco de dados – (exercícios parte 1).
4. Participação em Chat em horário marcado pelo professor formador, para discutir questões relativas aos exercícios propostos.

Décima Primeira Semana

As atividades a serem desenvolvidas na décima primeira semana são:

1. Leitura e estudo do conteúdo: Linguagem PHP com acesso a banco de dados – Parte 2.
2. Realização da atividade: Unidade D – Linguagem PHP com acesso a banco de dados – (exercícios parte 2).
3. Participação do Fórum de discussão proposto pelo professor formador.

Décima Segunda Semana

As atividades a serem desenvolvidas na décima segunda semana são:

1. Leitura e estudo do conteúdo: Linguagem PHP com acesso a banco de dados – Parte 3.
2. Realização da atividade: Unidade D – Linguagem PHP com acesso a banco de dados – (exercícios parte 3).
3. Participação em Chat em horário marcado pelo professor formador, para discutir questões relativas aos exercícios propostos.

Referências:

NIEDERAUER, J. *Desenvolvendo websites com PHP*: Aprenda a criar websites dinâmicos e interativos. São Paulo: Novatec, 2004.

NIEDERAUER, Juliano. *PHP para quem conhece PHP*. São Paulo: Novatec, 2006.

SOARES, Wallace. *PHP 5*: Conceitos, Programação e Integração com Banco de Dados. 4 ed. São Paulo: Érica, 2007.

PHP. *Manual do PHP*. Disponível na Web em http://php.net/manual/pt_BR/index.php

Professor-Autor

Anubis Graciela de Moraes Rossetto

Graduada em Ciência da Computação pela Universidade de Passo Fundo em 1998; Especialização em Sistemas de Informação pela Universidade de Passo Fundo (2002). Mestre em Ciência da Computação pela Universidade Federal de Santa Catarina em 2007; Atualmente, doutoranda em Ciência da Computação pela Universidade Federal do Rio Grande do Sul. Tem 9 anos de experiência em docência e atualmente é professora do Instituto Federal Sul-Rio-Grandense. A área de atuação é em Sistemas Distribuídos, com interesse em sistemas para web e computação móvel.

<http://buscatextual.cnpq.br/buscatextual/visualizacv.jsp?id=K4127507Y1>



tics



Introdução ao desenvolvimento para Web

Unidade A
Linguagem de Programação Web

INTRODUÇÃO AO DESENVOLVIMENTO PARA WEB

Introdução

Atualmente, acessamos a Web para obter diversos serviços, tais como:

- comunicar com colegas e amigos via softwares com MSN, skype, gtalk ou pelas redes sociais
- fazer compras (de quase todos os produtos)
- ler notícias
- entregar a declaração de IR
- consultar dados bancários e/ou realizar transações
- consultar a lista de filmes em cartaz no cinema
- consultar a programação da televisão
- consultar a previsão do tempo
- fazer inscrição para o processo seletivo
- fazer inscrição para o Enem
- ouvir rádio, procurar emprego, ...

Enfim, utilizamos este recurso com inúmeros propósitos diferentes e a tendência é que cada vez mais serviços sejam disponibilizados. Bem, mas então precisamos entender melhor como funciona, os conceitos e as possibilidades de aplicação para que possamos ser profissionais qualificados na criação de soluções para o ambiente Web.

Para começar, precisamos saber diferenciar Internet de Web. Você sabe?

A Internet e a World Wide Web

Internet

Você já deve ter ouvido/lido que a internet tem mais de 50 anos. Pois bem, por que a Internet existe há tanto tempo, mas o uso dos seus recursos é bem mais recente?

Isso porque a Internet é

a grande rede de computadores, ou seja, o conjunto de computadores que estão conectados através de várias redes. Então podemos entender a Internet como a infraestrutura.

A Internet surgiu em meados de 1969 com um projeto do governo americano denominado ARPANET. O objetivo do projeto era interligar universidades e instituições de pesquisa militares.

Web

E a Web? Bem, ela é bem mais nova. Em 1991, Tim Berners-Lee, pesquisador do CERN, lançou a WWW (*World Wide Web* – rede de alcance mundial) ou simplesmente, Web, com o objetivo de compartilhar arquivos (HTML e outros), tendo o navegador como ferramenta de acesso. Assim, a Web é

é um serviço da Internet, um ambiente onde os documentos são publicados, disponibilizados e acessados.

Então, podemos dizer que a web usa a Internet, mas ela em si não é a Internet. Sempre que acessamos um site, estamos navegando na web, mas quando usamos o MSN, por exemplo, estamos usando um software que usa a Internet e não necessariamente a Web.

Os documentos disponibilizados na Web podem estar na forma de vídeos, sons, hipertextos e figuras. Para visualizar as informações, usamos o navegador (*browser*) para descarregar os documentos (ou “páginas”) de servidores web (ou “sítios”) e mostrá-los ao usuário. O usuário pode então seguir as hiperligações na página para outros documentos ou mesmo enviar informações de volta para o servidor para interagir com ele. Esta funcionalidade é baseada em três padrões: URI, HTTP e HTML. Vamos ver mais detalhadamente cada um deles.

Atenção:

Internet e internet possuem significados diferentes. Enquanto internet significa um conjunto de redes de computadores interligadas, a Internet se refere à internet global e pública. Assim, existem inúmeras internets espalhadas por redes particulares, seja interligando empresas, universidades ou residências. Porém, existe somente uma rede única e global, o conjunto de todas as redes, a Internet.

URI - Universal Resource Identifier (Identificador Universal de Recursos)

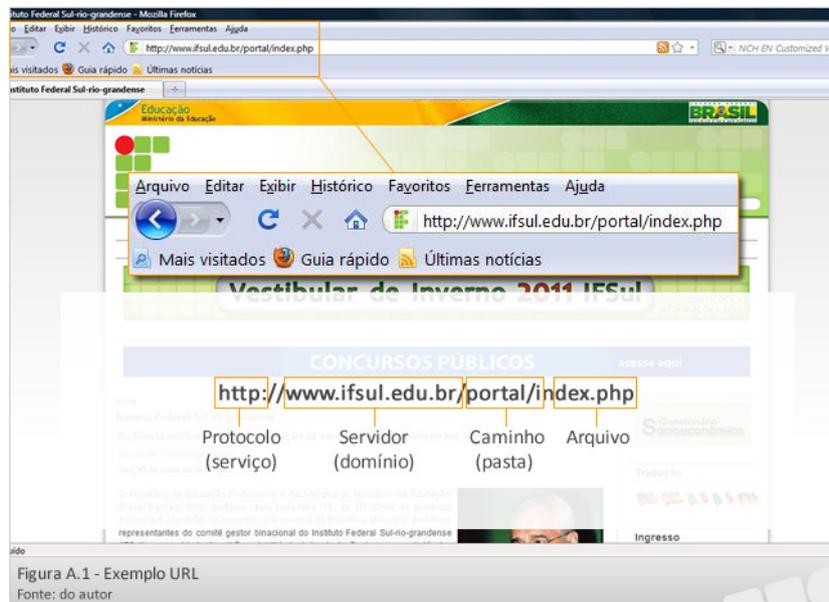
URI é a base do sistema de endereçamento da Web que usa uma cadeia de caracteres para identificar ou denominar um recurso na Internet. Um URI pode ser classificado como um localizador (URL) ou um nome (URN), ou ainda como ambos.

Um URN (*Uniform Resource Name* - Nome de Recursos Uniforme) é como o nome de uma pessoa, enquanto que um URL (*Uniform Resource Locator* - Localização de Recursos Uniforme) é como o seu endereço.

Os endereços que utilizamos atualmente na Web são os URLs, que possuem a seguinte estrutura:

- o protocolo de acesso ao recurso desejado (http)
- a máquina a ser conectada (www.ifsul.edu.br)
- o caminho de pastas até o recurso (portal/)
- o recurso (arquivo) a ser obtido (index.php)

Veja exemplo apresentado na figura A.1 abaixo:



HTTP (HyperText Transfer Protocol - Protocolo de Transferência de Hipertexto)

O HTTP é o protocolo que especifica como o navegador e o servidor web se comunicam. Cada vez que você aciona um link, seu navegador realiza uma comunicação com um servidor da Web através deste protocolo.

HTML (HyperText Markup Language - Linguagem de Marcação de Hipertexto)

O HTML é uma linguagem de marcação utilizada para produzir documentos que são interpretados por navegadores. O HTML mescla textos com códigos especiais (tags ou etiquetas).

As etiquetas (ou tags) permitem estruturar o conteúdo do documento e dar instruções ao navegador sobre o que fazer. As etiquetas são elementos entre parênteses angulares (< e >). A maioria das etiquetas tem sua correspondente de fechamento:

```
<etiqueta>...</etiqueta>
```

Exemplos:

```
<p> Introdução ao desenvolvimento para Web </p>
```

Acima com a etiqueta <p></p> o texto será apresentado em um novo parágrafo.

```
<b> Linguagem de Marcação de Hipertexto </b>
```

Com a etiqueta texto será apresentado em negrito.

Como podemos perceber vários novos conceitos vão sendo apresentados quando começamos a falar em Web e Internet. Por isso, esse documento se destina a apresentar várias informações importantes que devemos saber antes de iniciarmos o aprendizado específico das tecnologias para programação. A seguir vamos falar com pouco sobre: W3C, XHTML, CSS, linguagens do lado cliente e servidor e sobre tendências para a Web.

W3C

Conhecer o que é o W3C (*World Wide Web Consortium*) é muito importante, pois ele é responsável por desenvolver os padrões para criação e a interpretação de conteúdos para a Web. O W3C é um consórcio internacional com cerca de 300 membros, que agrega empresas, órgãos governamentais e organizações independentes, fundado por Tim Berners-Lee em 1994. O objetivo principal é desenvolver protocolos comuns e fóruns abertos que promovam a evolução da Web e assegurem a sua interoperabilidade. Assim, os sites que seguem esses padrões podem ser acessados e visualizados por qualquer pessoa ou tecnologia, independente de hardware ou software utilizados, de maneira rápida e compatível com os novos padrões e tecnologias que possam surgir.

Padrões seus como HTML, XHTML e CSS (que vamos mencionar mais adiante) são muito populares, contudo, em muitos casos são usados de forma errônea devido ao desconhecimento da especificação. É um dever de todo o desenvolvedor Web respeitar e seguir os padrões de acessibilidade do W3C, pois de outro modo poderá impor barreiras tecnológicas a diversas pessoas, desestimulando e até mesmo impedindo o acesso a suas páginas.

Saiba mais:

Para saber mais sobre o W3C acesse:

www.w3c.org

Uma iniciativa interessante do W3C é um validador de páginas HTML que foi disponibilizado com o objetivo de auxiliar os desenvolvedores na tarefa de verificar e corrigir os códigos criados por eles. O endereço é <http://validator.w3.org/>.

XHTML

XHTML? Por que este X? Bem, a XHTML (*EXtensible HyperText Markup Language* - Linguagem Extensível para Marcação de Hipertexto) foi criada para substituir a HTML, sendo uma linguagem mais “pura, clara e limpa”

A XHTML é uma reformulação da HTML, baseada em XML (*EXtensible Markup Language*). A XML tem como princípio que você pode criar seus próprios elementos de marcação e atributos para escrever seu documento. Isto significa que é você quem cria sua linguagem de marcação. A transformação de um documento existente de HTML para XHTML é uma tarefa bem simples, uma vez que a XHTML é baseada na HTML.

Mas o que muda de fato?

Esse processo de padronização tem em vista a exibição de páginas web em diferentes dispositivos (televisão, smartphone, celular, tablet).

As principais diferenças são:

- todas as tags devem ser escritas em letras minúsculas;
- as tags devem estar convenientemente aninhadas;
- os documentos devem ser bem formados;
- o uso de tags de fechamento é obrigatório;
- elementos vazios devem ser fechados;
- sintaxe para atributos.

CSS

Talvez você nunca tenha ouvido falar do CSS (*Cascading Style Sheets* - Folha de Estilos em Cascata) e nem tenha percebido a sua aplicação, mas praticamente todos os sites que você acessa usam este recurso. O CSS é também uma linguagem, mas esta é usada para criar estilos que definem o layout do documento HTML. Por exemplo, estilos para fontes, cores, margens, linhas, alturas, larguras, imagens de fundo, posicionamentos e muito mais.

O CSS é suportado por qualquer navegador e proporciona inúmeras opções de formatação para apresentação do layout de forma bastante sofisticada.

É fundamental que um desenvolvedor web saiba aplicar os recursos desta linguagem que foi proposta justamente como objetivo de separar o estilo do conteúdo da página.

Talvez você se pergunte se não pode fazer isso com HTML. Bem, a questão é que a recomendação W3C e os padrões de desenvolvimento indicam que o HTML deve ser usado para estruturar conteúdos e o CSS para formatar conteúdos estruturados.

Os benefícios do uso de CSS são:

- controle do layout de vários documentos a partir de uma simples folha de estilos;
- maior precisão no controle do layout;
- aplicação de diferentes layouts para servir diferentes mídias (tela, impressora, etc.);
- emprego de variadas, sofisticadas e avançadas técnicas de desenvolvimento que facilitam a manutenção.

Atenção:

Esta disciplina não prevê abordar a linguagem CSS. No entanto, é importante que o aluno compreenda sua aplicação, até porque na linguagem HTML nos deteremos as tags que tratam apenas da estrutura do documento.

Linguagens de Programação para Web: client-side e server-side

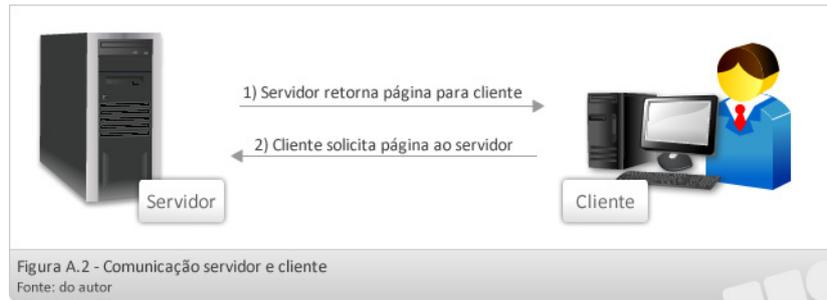
Nesta seção vamos falar sobre dois tipos de linguagens que são usadas no desenvolvimento para Web e diferenciar em que situações devemos usar cada uma delas. Primeiro, precisamos saber o que significa client-side e server-side:

Client-side (lado cliente):

O computador do usuário que está acessando um site por meio do navegador.

Server-side(lado servidor):

Servidor que hospeda o site.

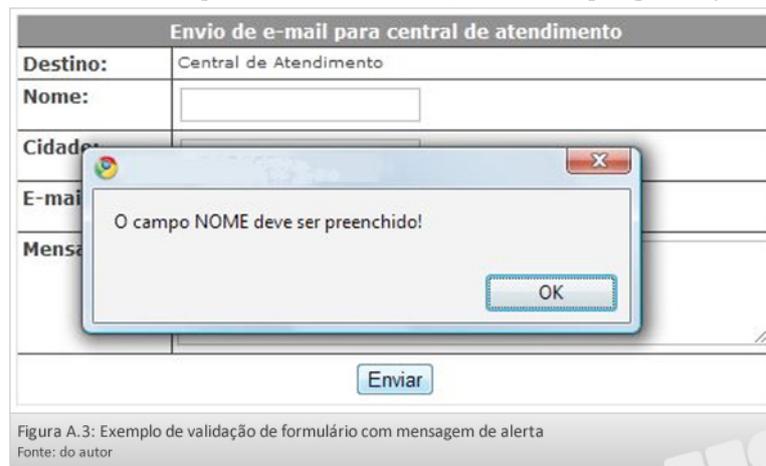


A figura A.2 apresenta o Cliente e o Servidor que interagem no processo de acesso a um site.

Então quando falamos de uma **linguagem de script** (linguagem interpretada) **do lado cliente**, significa que esta linguagem está sendo interpretada pelo navegador do cliente.

Já uma **linguagem de script do lado servidor** tem seu código interpretado no servidor, sendo que o cliente recebe apenas o resultado do seu processamento.

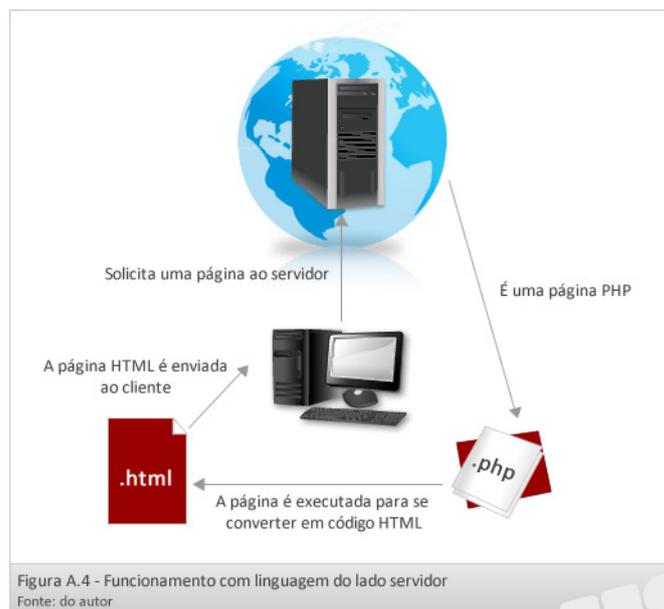
Com relação à linguagem do lado cliente, é utilizado o **JavaScript** para que seja possível manipular a página do usuário diretamente, tomando ações dinâmicas que vão desde emitir uma mensagem, validação de um formulário, abrir e configurar janelas, manipulação de objetos da página, detectar versão do navegador. A figura A.3 apresenta um exemplo de validação de formulário que apresenta mensagens ao usuário quando os campos obrigatórios não são preenchidos. Assim, para situações que necessitam de interação com o usuário e não dependem do servidor, usamos a programação JavaScript.



A linguagem do lado servidor é instalada no servidor e vai rodar “por trás dos panos”, fornecendo a lógica principal da aplicação.

A figura A.2 apresenta como ocorre o funcionamento:

- Quando o usuário faz uma requisição (entra numa página, clica num link, etc.), o pedido é enviado ao servidor.
- O servidor recebe a requisição e verifica se a página tem código que precisa ser interpretado. Em caso positivo faz o processamento.
- Depois, transforma o resultado final em um XHTML e envia ao navegador do cliente. Assim, o cliente já recebe tudo pronto.



Uma linguagem do lado servidor é necessária, por exemplo, quando precisamos interagir com um banco de dados (que está no servidor). Por exemplo, quando você acessa um site que publica notícias, muito provavelmente, este site armazena as notícias em uma base de dados e uma linguagem do lado servidor é utilizada para recuperar esses dados e gerar dinamicamente uma página HTML de visualização para o usuário.

Existem várias linguagens do lado servidor: PHP, JSP, ASP, Python. Nossa disciplina estudará a linguagem PHP (*Hypertext Preprocessor*) que é atualmente uma das linguagens mais utilizadas, open-source e independente de plataforma. Quando abordamos a linguagem falaremos mais sobre ela.

Tendências Web

Falaremos um pouco agora sobre alguns importantes conceitos que são tendências para a Web. É visível como as páginas Web evoluíram, melhoraram com o passar do tempo. Atualmente, os usuários têm mais liberdade, podem participar e interagir com blogs, redes sociais e outros serviços que estão cada vez mais presentes. Dois conceitos bem atuais relacionados a este novo cenário da Web é a Web 2.0 e a Web 3.0.

WEB 2.0

A Web 2.0 não é uma nova Web, mas sim um termo utilizado para denotar um novo conceito de aplicações, que reforça a ideia de troca de informações e colaboração dos internautas com sites e serviços virtuais. Neste conceito, o ambiente on-line se torna mais dinâmico e os usuários colaboram para a organização do conteúdo.

Alguns exemplos que estão alinhados a este novo conceito são a enciclopédia *Wikipedia*, cujas informações são disponibilizadas e editadas pelos próprios internautas, o Orkut, o Facebook, o Youtube, o Twitter, Flickr, LinkedIn, Blogs, entre outros.

Com a Web 2.0, o grande desafio é que os sites precisam lidar com um público cada vez mais ativo e presente, que deseja interagir. Por isso, a divulgação e a publicidade se tornaram cada vez mais necessários nesse cenário.

Associado a isso, outras tecnologias vêm surgindo e requerem que o desenvolvedor faça uso delas para aprimorar os sites. Vamos falar um pouquinho de uma tecnologia que os desenvolvedores para Web devem conhecer: AJAX.

AJAX (*Asynchronous Javascript and XML* - Assíncrono de Javascript e XML) é o uso metodológico de tecnologias como Javascript e XML com o objetivo de tornar páginas Web mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações. Assíncrono se refere ao método para se comunicar com um servidor web sem a necessidade de recorrer ao recarregamento de página, não precisa haver sincronia do carregamento de dados com o carregamento da página Web. Isso permite que a navegação seja mais interativa para o usuário. Segundo Jesse James Garrett (inventor do termo), "AJAX não é uma tecnologia. São, não verdade, várias tecnologias, juntando-se de maneiras novas e poderosas." A Google utiliza muito este recurso nas suas ferramentas, por exemplo, quando se pretende fazer um login e o usuário e/ou senha são inválidos. Para fazer essa verificação, a página faz uma solicitação ao servidor que retorna um resultado. A página trata esse resultado e mostra uma mensagem na página para o usuário, tudo isso sem recarregar a página (a figura A.4 apresenta um exemplo).



WEB 3.0

A Web 3.0 ainda não está presente, mas é também um novo conceito de Web que pretende ser a organização e o uso de maneira mais inteligente de todo o conhecimento já disponível na Web. Este paradigma surgiu a partir da necessidade de desenvolvimento de programas que entendam como fazer melhor uso da grande quantidade de dados. Assim, o foco principal da Web 3.0 é a Web semântica.

A Web Semântica tem como premissa uma web com toda sua informação organizada de forma que não somente seres humanos possam entendê-la, mas principalmente máquinas. O objetivo é tornar tarefas como a pesquisa sobre filmes e sobre comida mais rápidas e mais fáceis. Em vez de múltiplas pesquisas, você poderá digitar uma ou duas frases complexas no seu navegador e este irá analisar sua pergunta, pesquisar por todas as respostas possíveis e então, organizar os resultados para você.

Síntese

Bem pessoal, creio que foi possível observar que o desenvolvimento para Web requer o conhecimento de diversas tecnologias, cada uma com seu propósito de aplicação. Esta disciplina terá como enfoque principal preparar o aluno para a parte de programação de aplicações Web com linguagem do lado cliente (JavaScript) e linguagem do lado servidor (PHP). Assim, poderemos criar soluções que acompanham a tendência para Web com recursos interessantes e que possibilitam facilidade de uso ao usuário.

Atividades

Agora que você conhece as principais tecnologias que envolvem o desenvolvimento para Web, você está desafiado a olhar diferente para os sites que acessa.

Inicialmente, acesse o vídeo que foi disponibilizado sobre a “A evolução da Web”.

Após, acesse vários sites que você já conhece e faça uma análise para depois discutirmos sobre alguns aspectos conforme as questões norteadoras abaixo:

1. Considerando os conceitos e tecnologias abordados, você já possuía tais conhecimentos?
2. Quando você navega em sites identifica os recursos que foram apresentados?
3. Você compreende por que são necessárias várias tecnologias para o desenvolvimento e a aplicação de cada uma?
4. Consegue identificar situações em que o site aplica uma linguagem do lado cliente?
5. Consegue identificar se o site usa uma linguagem do lado servidor?
6. Pesquise mais informações sobre Web 2.0 e Web 3.0 e compartilhe com seus colegas.
7. Faça uma pesquisa sobre o que é RSS (também uma tendência da Web 2.0) e compartilhe com seus colegas.

Introdução ao HTML

Esta unidade tem objetivo fazer uma revisão da estrutura principal da linguagem (X)HTML, focando principalmente em formulários. Estes conhecimentos servirão de base para as próximas unidades.

Abordaremos a XHTML a fim de seguir as recomendações da W3C, pois desta forma você estará apto a construir páginas que tenham como características a rapidez no carregamento, a facilidade de manutenção, acessibilidade aos mais variados agentes de usuário e cuja estrutura esteja separada da apresentação e estilização.

A grafia (X)HTML é utilizada para se fazer referência tanto à HTML quanto à XHTML. Todos os elementos que serão apresentados são válidos para a HTML. No entanto, seguiremos a especificação da XHTML que é mais rigorosa. Por exemplo, ao escrevermos HTML, é correta a grafia dos elementos tanto em minúsculo como em maiúsculo, mas ao escrevermos em XHTML, só é válida a grafia de elementos em letras minúsculas.

As versão atual da HTML é a 4.01 e da XHTML é a 1.1. Porém está em elaboração e promete profundas alterações a especificação para a HTML 5.0.

Para criar documentos (X)HTML você pode usar um simples editor de texto como o Bloco de Notas. Existem ferramentas que auxiliam no desenvolvimento e dão suporte visual, os chamados editores WYSIWYG (What You See Is What You Get - O que você vê é o que você obtém). Um exemplo é a ferramenta Dreamweaver da Adobe bastante utilizada por desenvolvedores. Porém, cabe destacar que é uma ferramenta que precisa de licença de uso. Por hora vamos usar o bloco de notas para aprender a (X)HTML.

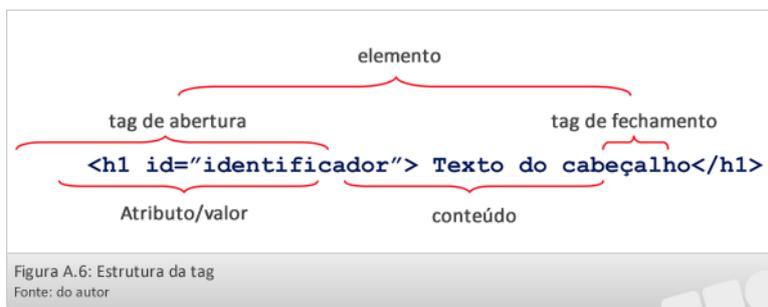
Estrutura do (X)HTML

Antes de qualquer coisa, é necessário conhecer o significado de alguns termos que vamos usar daqui para frente.

Termos	Significado
Renderização	Transformação da marcação (X)HTML do documento em algo capaz de ser lido e entendido pelo usuário. A renderização é feita pelo navegador. Quando visualizamos uma página é porque o navegador renderizou a página para nós.
Documento e página	Documento é o arquivo completo que contém toda a marcação. A página web ou simplesmente página é o resultado da renderização do documento.
Elemento	É a menor unidade de marcação. Cada elemento tem uma destinação específica, por exemplo, para marcar títulos, devemos obrigatoriamente usar o elemento previsto para marcação de títulos; para marcar parágrafos existe um elemento específico também, e assim por diante.
Elemento vazio	São representados apenas pela tag(etiqueta) de abertura, não existe a tag de fechamento. Estes elementos não possuem conteúdos para marcar e se destinam a fornecer informações adicionais.

Tags (ou etiquetas)

Os elementos de marcação são representados por um par de tags(ou etiquetas) que são os sinais < e >. Geralmente, na marcação usamos uma tag de abertura e uma tag de fechamento, entre as quais se insere o conteúdo. A tag de fechamento possui por sua vez uma / (**barra**) logo após o sinal de <. Veja na figura A.6 a estrutura:



Observe que dentro da tag de abertura foi inserido o atributo **id** com o valor “**identificador**”. Atributos são usados para definir uma propriedade de um elemento. Eles devem ser colocados sempre na tag de abertura, logo após o nome do elemento, precedido de um espaço. O atributo é composto de um nome, um sinal de igual (=) e um valor de atributo, cercado por aspas duplas (") ou simples ('). No caso do nosso exemplo, o atributo é o **id**, que serve para identificar, de maneira única, um elemento dentro de um documento (X)HTML.

Comentários

Muitas vezes é necessário incluir comentários no nosso documento (X)HTML. Nesse caso podemos usar da seguinte forma:

```
<!-- comentário -->
```

Para iniciar o comentário usamos o marcado <!-- e para fechar -->. Assim, o navegador irá ignorar o texto incluído entre as marcações de comentário.

Estrutura mínima de um documento

Um documento (X)HTML possui uma estrutura mínima dividida em três partes, ou seja, uma espécie de esqueleto do documento. Vejamos:

1. seção de declaração do tipo de (X)HTML
2. seção head (cabeçalho)
3. seção body (corpo)

A figura A.7 apresenta a estrutura mínima de um documento documento. A linha 1 contém a primeira parte do documento que declara o tipo de (X)HTML ou *doctype*. Após, na linha 2 está o elemento `<html>` que é aberto no início do documento e fechado apenas no fim. Dentro do elemento `<html>` estão as outras duas seções. A seção do cabeçalho do documento é aberta na linha 3 e fechada na linha 5, indicada pelo elemento `<head></head>`. Já a seção do corpo do documento é aberta na linha 7 e fechada na linha 9, indicado pelo elemento `<body></body>`.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 <html xmlns="http://www.w3.org/1999/xhtml">
3 <head>
4 <title>Exemplo</title>
5 </head>
6
7 <body>
8 texto do documento...|
9 </body>
10 </html>
```

Figura A.7 - Estrutura mínima de um documento (X)HTML
Fonte: do autor

Doctype

Doctype é a declaração do tipo de documento e é escrita na primeira linha do documento. O doctype da linha 1 da imagem A.5 quer dizer o seguinte: *a marcação para o presente documento está escrita em linguagem html pública, segundo as regras estabelecidas pelo W3C para XHTML 1.0 Transitional. O conjunto de elementos e atributos, bem como as regras de sintaxe empregadas neste documento, está de acordo com o que prescreve o documento hospedado no site do W3C no seguinte endereço: <http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd>*

Veja na tabela abaixo os tipos de *doctype*:

Strict	<i>doctype</i> mais rígido e possibilita um código mais limpo. É usada em conjunto com estilos CSS (<i>Cascading Style Sheets</i>). Não pode usar elementos e atributos declarados em desuso pelo W3C e nem marcação de frames.
Transitional	<i>doctype</i> mais flexível e deve ser usado para possibilitar uma navegação por navegadores mais antigos ou que não suportam CSS por exemplo. Não pode conter elementos ou atributos destinados a marcação de frames.
Frameset	deve ser usada quando o documento contém molduras (frames).

Atenção:

Sobre frames é importante dizer que o seu uso está sendo abandonado por ter restrições à acessibilidade.

Head

A seção *head* contém elementos com informações adicionais sobre o documento. Um elemento obrigatório é o título do documento (`<title></title>`). O conteúdo do elemento *title* será colocado na barra superior da janela do navegador. Os elementos podem aparecer no cabeçalho, alguns recomendados pela W3C, por exemplo, para informar qual o conjunto de caracteres usados na marcação.

Body

A seção *body* contém os elementos que marcam os conteúdos que serão visualizados no navegador. Tudo o que você vê no navegador está entre os elementos `<body></body>`.

Principais Elementos (X)HTML

Agora você irá conhecer os principais elementos do (X)HTML. A ideia não é apresentar todos, mas aqueles que são mais utilizados e que dizem respeito a parte estrutural do documento e seu conteúdo.

Elemento	Finalidade
<pre><h1> título de nível 1 </h1> <h2> título de nível 2 </h2> ... <h6> título de nível 6 </h6></pre>	abertura e fechamento de um cabeçalho. São 6 níveis: h1 (o maior) até h6 (o menor).
<pre><p> texto </p></pre>	inicia um parágrafo pulando uma linha.
<pre>
</pre>	quebra de linha

<code><hr/></code>	cria uma linha divisória horizontal.
<code><div> texto </div></code>	abertura e fechamento de uma divisão de página cujos elementos que estão dentro dela obedecerão às mesmas definições.
<code> texto do link </code>	Estabelece um texto ou imagem como link. Atributo
<code></code>	inserção de imagem. Atributos: src ⇒ caminho e nome da imagem width ⇒ largura da imagem em pixels height ⇒ altura da imagem em pixels alt ⇒ texto alternativo a imagem (importante para acessibilidade).

Atributos

Como já foi citado acima, os elementos podem conter atributos que referenciam características ou dados complementares do elemento. Cada elemento pode conter atributos diferentes. Um exemplo, é o atributo `align` que pode ser aplicado a vários elementos e tem por objetivo o alinhamento do conteúdo do elemento. As opções são *left*, *center*, *right*. Ele pode ser aplicado, por exemplo, aos elementos *p*, *h1*, *h2... h6*, *hr*, *div*, *table*.

Ex.:

```
<p align='center'>texto </p>
```

Links

Sobre a criação de links existem possibilidades diferentes de criação dependendo o destino do link:

- **links internos:** são aqueles que apontam para uma página dentro do próprio site. É necessário informar apenas o local e nome do arquivo dentro do site.

```
<a href="produtos.html">Página de produtos</a>
```

- **links externos:** são aqueles que apontam para fora do próprio site. Neste caso devemos usar o endereço completo da página destino(URL). Observe que foi usado o atributo `target='_blank'` para indicar que o endereço deve ser aberto em uma nova janela.

```
<a href="http://www.w3c.org" target="_blank">W3C</a>
```

- **link para e-mail:** são aqueles que apontam para um endereço de e-mail. Neste caso usamos a `mailto:` antes de indicar o e-mail.

```
<a href="mailto:teste@provedor.com.br">Email Teste</a>
```

Tabelas

Em (X)HTML utilizamos bastante tabelas, por isso é importante reconhecer toda a sua estrutura. Vejamos a tabela mostrada na figura A.8, como será o (X)HTML desta tabela?

linha 1 coluna 1	linha 1 coluna 2
linha 2 coluna 1	linha 2 coluna 2
linha 3 coluna 1	linha 3 coluna 2

Figura A.8 - Tabela
Fonte: do autor

A figura A.9 apresenta os elementos (X)HTML da tabela visualizada na figura A.8. Agora vejamos os detalhes:

- O elemento *table* indica a criação de uma tabela. Abre com `<table>` e fecha com `</table>`. Abre na linha 10 e fecha na linha 23.
- para cada linha da tabela (são 3) existe um elemento *tr*. `<tr>` abre linha e `</tr>` fecha a linha.
- dentro de cada linha está o elemento da coluna (*td*). Neste caso a tabela possui duas colunas, então dentro de cada linha existem dois elementos *td*. `<td>` abre a coluna e `</td>` fecha coluna. É importante destacar que os dados da tabela sempre estarão dentro do elemento *td*.

Dicas:

`tr` ⇒ table row (linha da tabela) | `td` ⇒ table data (dados da tabela)

```

10 <table width="600" border="1" align="center" cellpadding="2" cellspacing="0">
11 <tr>
12 <td>linha 1 coluna 1</td>
13 <td>linha 1 coluna 2</td>
14 </tr>
15 <tr>
16 <td>linha 2 coluna 1</td>
17 <td>linha 2 coluna 2</td>
18 </tr>
19 <tr>
20 <td>linha 3 coluna 1</td>
21 <td>linha 3 coluna 2</td>
22 </tr>
23 </table>
    
```

Figura A.9 - (X)HTML da tabela
Fonte: do autor

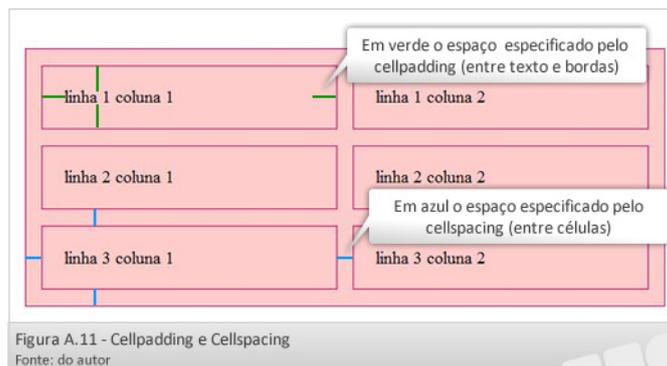
Vejamos novamente com algumas anotações:

```

10 <table width="600" border="1" align="center" cellpadding="2" cellspacing="0">
11 <tr>
12 <td>linha 1 coluna 1</td>
13 <td>linha 1 coluna 2</td>
14 </tr>
15 <tr>
16 <td>linha 2 coluna 1</td>
17 <td>linha 2 coluna 2</td>
18 </tr>
19 <tr>
20 <td>linha 3 coluna 1</td>
21 <td>linha 3 coluna 2</td>
22 </tr>
23 </table>
    
```

Figura A.10 - Estrutura do (X)HTML da tabela
Fonte: do autor

Vamos ver em mais detalhes agora os atributos cellpadding e cellspacing do elemento table. Veja a figura A.11:



Listas

Vamos trabalhar um pouco com listas. Podemos criar listas não numeradas ou numeradas. Veja no exemplo da figura A.12.

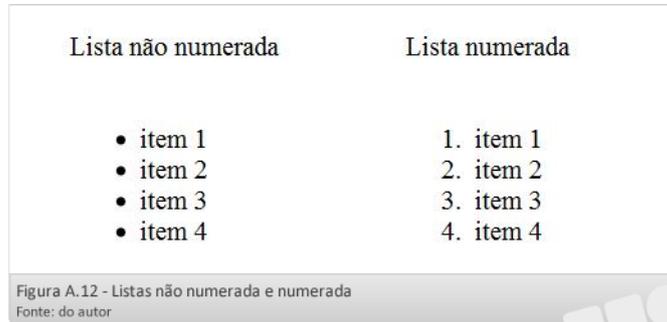


Figura A.12 - Listas não numerada e numerada
Fonte: do autor

Para criar uma lista não numerada usamos o elemento ``. Para cada item da lista usaremos o elemento ``. Veja na figura A.13 as marcações para criação da lista não numerada.

```

1 <ul>
2   <li>item 1</li>
3   <li>item 2</li>
4   <li>item 3</li>
5   <li>item 4</li>
6 </ul>

```

Figura A.13 - (X) HTML da lista não numerada
Fonte: do autor

Para criar uma lista numerada usamos o elemento ``. Para cada item da lista usaremos o elemento ``. Veja na figura A.14 as marcações para criação da lista numerada.

```

1 <ol>
2   <li>item 1</li>
3   <li>item 2</li>
4   <li>item 3</li>
5   <li>item 4</li>
6 </ol>

```

Figura A.14 - (X) HTML da lista numerada
Fonte: do autor

Dicas:

ul ⇒ unordered list | ol ⇒ ordered list | li ⇒ list item

Formulários em (X)HTML

Para que possamos interagir com o usuário e criar soluções que usem as linguagens do lado cliente e servidor, é fundamental trabalhar com formulários (X)HTML. Por isso, vamos ver os principais elementos de formulário. Para começar vejamos um exemplo de formulário na figura A.15. Um formulário tem como objetivo, geralmente, possibilitar a entrada de dados. Neste exemplo da figura há apenas uma caixa de texto e um botão.

Figura A.15 - Exemplo Formulário
Fonte: do autor

Vejamos o (X)HTML do formulário acima na figura A.16. Todo formulário inicia com o elemento

form. Todos os elementos vinculados ao formulário ficam entre `<form></form>`. Os elementos HTML basicamente usados para marcação de formulários são:

- *form* que é o “*container*” para os demais elementos;
- *input* que pode ser de variados tipos como: *text*, *submit*, *button*, *radio button* e *checkbox*;
- *textarea* para entrada de textos *multi-linhas*, tais como comentários;
- *select* também conhecido como *drop-down-list*, ou menu de lista;

```

1 <form name="form1" method="post" action="">
2
3   <label for="email">E-mail:</label>
4
5   <input name="email" type="text" id="email"
6     size="40" maxlength="50">
7
8   <input type="button" name="botao" id="botao"
9     value="ok">
10
11 </form>

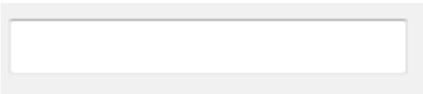
```

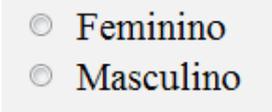
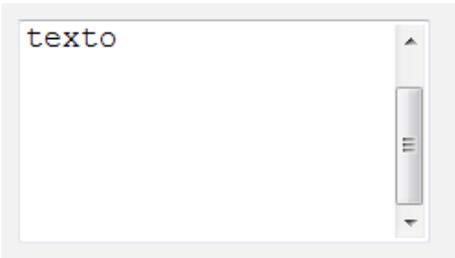
Figura A.16 - (X)HTML do formulário da figura A.15
Fonte: do autor

Todos os elementos de formulário devem possuir os atributos *id* e *name*. Eles serão necessários para manipulação dos valores tanto em programação do lado cliente (*Javascript*) quando em programação do lado servidor (PHP). O atributo *id* é o identificador do elemento, ou seja, sempre será um valor único para cada elemento do documento. Geralmente, o *id* e *name* possuirão o mesmo valor (exceção é para o tipo *radio*).

No exemplo da figura A.16, vimos o elemento *label* que é usado para especificar etiquetas para os controles. O atributo *for* associa a etiqueta de um campo de formulário. O valor do atributo *for* deve ser o valor do atributo *id* ou *name* do elemento que se deseja referenciar. No nosso exemplo o *label* é referente a caixa de texto chamada *email*.

A tabela abaixo apresenta as variações para os elementos *input*, *select* e *textarea*.

Elemento	Código	
<code>input type="text"</code>	<code><input type="text" name="texto" id="texto" /></code>	Caixa de texto: 
<code>input type="password"</code>	<code><input type="password" id="texto2" name="texto2" value="senha" /></code>	
<code>input type="hidden"</code>	<code><input type="hidden" id="oculto" name="oculto" value="1" /></code>	Campo oculto (não aparece). Mas possui <i>id</i> e valor. Usado para passar alguma informação que não precise ser visualizada.
<code>input type="button"</code>	<code><input type="button" name="botao" id="botao" value="ok"></code>	Botão: 

input type="submit"	<pre><input type="submit" name="botao2" id="botao2" value="Enviar" /></pre>	<p>Botão do tipo submit: envia o formulário executando a ação do form</p> 
input type="reset"	<pre><input type="reset" name="botao3" id="botao3" value="Limpar" /></pre>	<p>Botão do tipo reset: limpa os dados do form, voltando ao estado inicial.</p> 
input type="checkbox"	<pre><input type="checkbox" id="status" name="status" value="S" /></pre>	<p>Checkbox: quando selecionado conterá o valor do atributo value.</p> 
input type="radio"	<pre><input type="radio" name="sexo" value="F" id="sexo_0" /> Feminino
 <input type="radio" name="sexo" value="M" id="sexo_1" /> Masculino</pre>	<p>Botões de rádio: devem possuir o mesmo name para que ao marcar uma opção a outra seja desmarcada.</p> 
input type="file"	<pre><input type="file" name="arquivo" id="arquivo" /></pre>	<p>Arquivo: Seleção de um arquivo</p> 
select	<pre><select name="UF" id="UF"> <option value="RS">Rio Grande do Sul</ option> <option value="SC">Santa Catarina</option> <option value="PR">Paraná</ option> </select></pre>	<p>Menu de lista: para cada opção possui um elemento option. Quando selecionado conterá o valor do atributo value</p> 
textarea	<pre><textarea name="texto3" id="texto3" cols="20" rows="5"></textarea></pre>	<p>Área de texto: rows: número de linhas cols: largura em termos de colunas</p> 

Atributos do elemento form:

action = url

Este atributo indica o endereço de envio quando o formulário for processado. Geralmente uma outra página do site é especificada.

method = get|post

Este atributo indica sob qual forma será enviada as respostas: “POST” é o valor que corresponde ao envio de dados armazenados no corpo do pedido, enquanto “GET” corresponde ao envio dos dados codificados no URL e separados do endereço por um ponto de interrogação. O método POST é recomendado por ser uma forma mais segura de envio dos dados. No caso do método GET (envio dos dados através do URL), o URL assemelhar-se-á à uma cadeia do tipo:

<http://www.dominioteste.com.br?campo1=val1&campo2=val2>

enctype = content-type

Este atributo especifica o tipo de conteúdo usado para submeter o formulário no servidor (quando o valor de *method* for igual a “*post*”). O valor por padrão é “*application/x-www-form-urlencoded*”. O valor “*multipart/form-data*” deve ser usado em combinação com o elemento *input type="file"*

Síntese

Bem, agora já conhecemos a estrutura básica da linguagem e seus principais elementos. De agora em diante podemos começar a criar documentos (X)HTML. É importante lembrar que precisamos seguir com rigor a especificação da linguagem para que nossos documentos sejam considerados válidos. Vale lembrar também que existe bastante material disponível sobre (X)HTML, por isso, agora é exercitar e pesquisar sobre outros elementos e atributos.

Atividades

1. Como vimos, o XHTML é mais rigoroso que o HTML quanto a estrutura dos elementos. Dada a tabela abaixo, apresente a estrutura correta considerando as restrições do XHTML.

Errado	Correto
<code><DIV><P>Aqui está errado!</P></DIV></code>	
<code><div><p>texto em negrito </p></div></code>	
<code><p>um parágrafo <p> outro parágrafo</code>	
<code>
</code>	
<code><hr></code>	
<code><td width=300></code>	
<code><input checked/></code>	

2. Dado o código html abaixo, encontre e corrija os erros existentes.

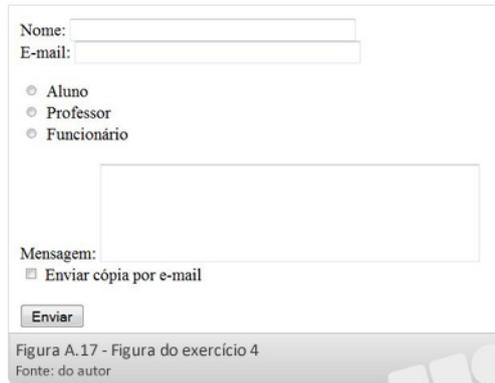
Linha	Código
1	<code><html></code>
2	<code><title>Atividade - LPW</title></code>
3	<code><head></code>
4	<code></head></code>
5	<code><body></code>
6	<code>IFSUL</code>
7	<code></code>
8	<code>
</code>
9	<code><table width="250" border="1" cellspacing="0" cellpadding="3"></code>
10	<code><tr></code>
11	<code><td>Código</code>
12	<code><td>Descrição</code>
13	<code><td>Preço</code>
14	<code></tr></code>
15	<code><tr></code>
16	<code><td>1</code>
17	<code><td>Impressora</code>
18	<code><td>R\$ 300,00</code>
19	<code></tr></code>
20	<code></table/></code>
21	<code></body></code>

3. Apresente o código para uma página HTML que contenha os seguintes elementos:

- Estrutura básica do documento (cabeçalho e corpo descritos corretamente)
- Título geral do documento (para exibição na barra superior do navegador)
- Título 1: Minha página;
- Um parágrafo de texto livre
- Título 2: Tabela de Calorias
- Tabela conforme abaixo:

Refrigerantes e energéticos	
Coca-cola	R\$ 2,50
Coca-cola light	R\$ 3,00
Fanta	R\$ 2,00

4. Apresente o código (X)HTML que corresponda a apresentação da página com o formato conforme a figura abaixo:



5. Faça uma pesquisa sobre os atributos que podem ser aplicados aos elementos abaixo relacionados e apresente exemplos de uso:

img

a

input

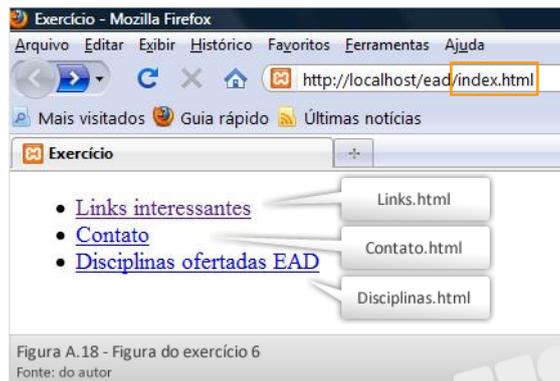
ul

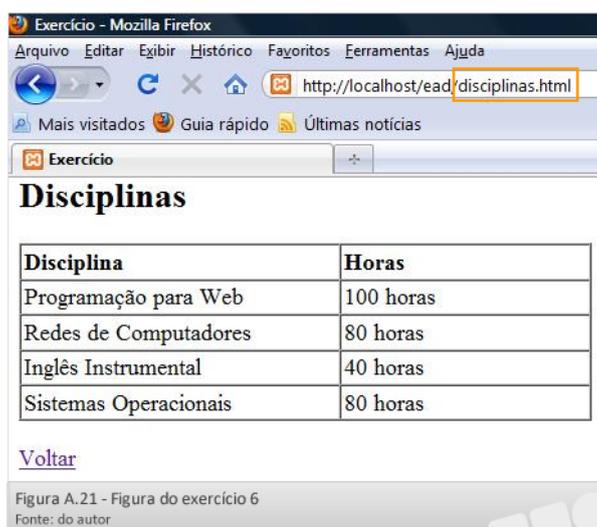
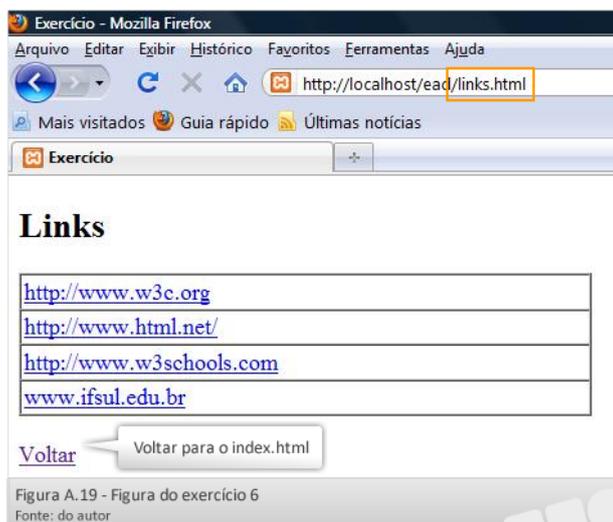
tr

6. Tomando por referência as páginas mostradas nas figuras abaixo crie os seguintes documentos (X)HTML:

- index.html
- links.html
- contato.html
- disciplinas.html

Obs.: os balões amarelos na imagens são apenas lembretes







tics

B

B

Linguagem do lado cliente

Unidade B
Linguagem de Programação Web

UNIDADE

B

LINGUAGEM DO LADO CLIENTE

Introdução

Esta unidade tem por objetivo estudar uma linguagem do lado cliente, que permite incorporar às páginas (X)HTML a possibilidade de interação com o usuário, ou seja, vai proporcionar o comportamento da página.

Na unidade anterior vimos que a (X)HTML possibilita, por meio de marcações, a estruturação do documento com seu conteúdo. Porém, não conseguimos introduzir lógica em um documento usando apenas (X)HTML, por exemplo:

- se acontecer isso, então faça aquilo...
- se o usuário interagir com a página desta forma, então faça aquilo...
- se o usuário está utilizando o navegador tal, então mostre tal objeto...

Por isso, precisamos conhecer a linguagem JavaScript para que possamos introduzir comportamento aos documentos que vamos construir. Assim, a seguir abordaremos a sintaxe básica da linguagem, como capturar eventos, criação de funções, interação com formulários (X)HTML, manipulação de janelas, entre outros. Então, mãos a obra...

Introdução a linguagem JavaScript

Histórico e características

JavaScript é uma linguagem criada especificamente para a Web, por isso, é leve e interpretada pelo navegador. Como já comentamos no nosso documento introdutório, ela é uma linguagem do lado cliente, e possibilita a interação com o usuário, uma vez que é possível controlar a página e seus elementos.

Atenção:

Não faça confusão da linguagem JavaScript com a linguagem JAVA. Elas podem até ter alguma semelhança de sintaxe em alguns comandos, mas não vai além disso. Vale lembrar que a linguagem JAVA permite a criação de aplicações para *desktop*, web e dispositivos, ou seja, é uma plataforma mais completa.

A linguagem foi desenvolvida inicialmente pela Netscape e foi chamada de LiveScript. Já no fim de 1995, a Sun (que também desenvolveu o Java) se incorporou ao projeto e a linguagem foi nomeada de JavaScript. A versão da JavaScript atual e disponível nos navegadores é a 1.5.

Vamos conhecer algumas características da linguagem JavaScript:

linguagem não-tipada:

Isto significa que as variáveis não precisam ser criadas com um tipo específico. Os tipos são associados

com valores atribuídos, não com variáveis. Por exemplo, a variável `x` poderia ser associada a um número e mais tarde associada a uma string.

Case sensitive:

Significa dizer, que o interpretador diferencia minúsculas de maiúsculas, tanto para variáveis/funções definidas pelo programador, como para todos os comandos da linguagem. Isso faz com que os comandos tenham que ser escritos da mesma forma que são apresentados. Caso isso não ocorra o comando passa a ser interpretado como uma variável.

Baseada em objetos:

É uma linguagem baseada em objetos nativos ou criados pelo programador, ou seja, trata todos os elementos de uma página como objetos, que normalmente são agrupados por tipo ou finalidade.

Sintaxe

Para começar, a programação JavaScript é incluída dentro dos documentos (X)HTML, porém precisamos indicar que estamos programando em JavaScript. Para isso, identifica-se da seguinte forma:

```
<script>
```

comandos...

```
</script>
```

Tudo que for escrito entre o `<script></script>` será considerado programação JavaScript. Porém, recomenda-se utilizar o atributo `type` que define o tipo de conteúdo do script. Ficando assim:

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

comandos...

```
</script>
```

Outro forma é gerar um documento com a programação JavaScript a parte e fazer referência a este arquivo. Para tanto se utiliza o atributo `src`. Isso é bastante comum e até recomendado. Veja o exemplo abaixo.

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

```
</script>
```

Neste caso só vai a referência ao arquivo e o elemento é fechado. Veja que o padrão para extensão de arquivos que contenham programação JavaScript é `js`. Também é recomendado que se crie uma pasta específica para estes arquivos.

Além disso, é importante observar que existe uma padronização quanto a inclusão da programação JavaScript no documento. A convenção específica que o elemento `script` deva estar dentro do cabeçalho do documento. Veja na figura B1.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml">
4 <head>
5 <title>Exemplo</title>
6 <script type="application/javascript">
7
8 </script>
9
10 </head>
11
12 <body>
13 </body>
14 </html>

```

Figura B.1 - Elemento script
Fonte: do autor

Comentários

A convenção JavaScript para comentários é apresentada abaixo:

Tudo o que está escrito entre o // e o fim da linha será ignorado.

```
// comentário de linha
```

Também é possível incluir comentários em diversas linhas como abaixo:

```
/* comentário em
diversas linhas */
```

Atenção:

Não confundir os comentários Javascript e os comentários (X)HTML (<!--comentário -->).

Dicas:

Faça uso dos comentários ao longo da sua programação, isso pode ajudar na manutenção posterior do código e ainda faz você compreender melhor os recursos da linguagem.

Variáveis

Trabalhar com variáveis na linguagem JavaScript vai parecer mais fácil que outras linguagens, pois não é necessário declarar o tipo da variável. Para fazer a declaração use a palavra var. O nome da variável pode ser de qualquer tamanho, conter números, dígitos e underlines ('_'). Exemplos:

```
<script type="text/javascript">
    var nome;
    var preco_unitario;
    var valor1;
    var email = 'aaaa@teste.com.br';

```

```
</script>
```

A variável assumirá um tipo de acordo com o valor que lhe for atribuído. Cabe ressaltar que a palavra **var** é opcional, porém é altamente recomendado que faça uso na declaração de forma a evitar erros.

Atenção:

Como JavaScript é case sensitive, a variável de nome teste é diferente da variável de nome Teste.

Dicas:

Na linguagem JavaScript você pode usar tanto aspas simples como duplas para indicar que um conteúdo é String.

Na linguagem JavaScript o ponto e vírgula (;) no fim da linha de comando é opcional, porém seu uso é recomendado.

Primeiro Script

Bem pessoal, vamos fazer o nosso “Olá mundo!” em JavaScript. Vocês podem usar o bloco de notas para codificar (não se esqueçam de salvar o arquivo com a extensão html). Veja o código da figura B.2:

- Na linha 11 foi criada a variável chamada teste com valor “olá mundo!!”.
- Na linha 12 estamos mostrando o valor da variável teste no navegador. Para isso, estamos usando o **document.write()**. Isto significa que estamos escrevendo no documento.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 <html xmlns="http://www.w3.org/1999/xhtml">
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5 <title>Exemplo</title>
6 </head>
7 <body>
8
9 <script type="application/javascript">
10
11     var teste = 'olá mundo!!';
12     document.write(teste);
13
14 </script>
15 </body>
16 </html>

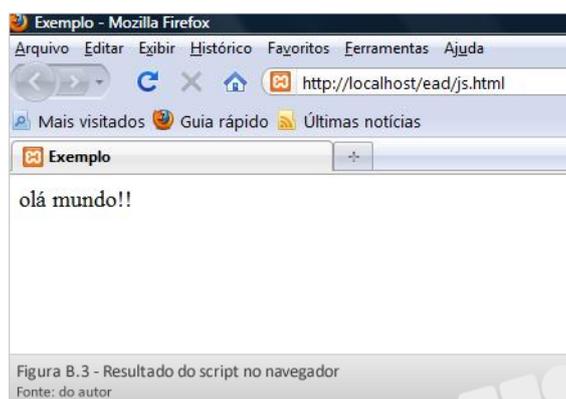
```

Figura B.2 - Primeiro Script
Fonte: do autor

Atenção:

Na figura B.2 o script está dentro do elemento body, isso porque estamos escrevendo no documento.

A figura B.3 mostra o resultado da página no navegador.



Operadores Lógicos

Operadores lógicos são operadores normalmente utilizados em comandos condicionais, como *if*, *for* e *while*

Operador lógico	Finalidade
==	Igual
!=	Diferente
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
&&	E
	Ou (pipeline)

Operadores Lógicos

Operadores Matemáticos

Operador Matemático	Finalidade
+	Este operador serve para adição de valores e ao mesmo tempo, este operador pode ser usado na concatenação de strings. (concatenação = junção ou união) Exemplo: "programação" + "1" //retorna programação1
-	Subtração de valores
*	Multiplicação de valores
/	Divisão de valores
%	Retorna o resto de uma divisão. Exemplo: 150 % 13 retornará 7 7 % 3 retornará 1

Operadores Matemáticos

Expressões Simples com operadores

Operador Matemático	Finalidade
<code>+=</code>	Concatena /adiciona ao string/valor já existente. Exemplo: <code>x += y</code> é o mesmo que <code>x = x + y</code> , da mesma forma podem ser utilizados: <code>--</code> , <code>*=</code> , <code>/=</code> ou <code>%=</code>
<code>++</code>	Acrescenta 1 no valor Exemplo: <code>X = 3;</code> <code>X++; // X valerá 4</code>
<code>--</code>	Decrementa 1 no valor Exemplo: <code>X = 3;</code> <code>X--; // X valerá 2</code>

Expressões Simples com operadores

Conversão de tipos

Em expressões que envolvam valores numéricos e strings com o operador `+`, os valores numéricos serão convertidos implicitamente para strings. Exemplo:

```
X = "A resposta é " + 35; // retorna "A resposta é 35"
```

```
Y = 35 + " é a resposta" // retorna "35 é a resposta"
```

```
Y = "37" + 7 // retorna 377
```

Assim, as vezes torna-se necessário fazer a conversão de valores que estão como string para que possamos fazer cálculos. Para converter explicitamente strings em números existem as funções predefinidas `parseInt` e `parseFloat`.

parseInt: Converte uma string em um número inteiro.

Ex.:

```
num = "3A";
```

```
x = parseInt(num); // x recebe 3
```

parseFloat: Converte uma string em um número real.

Ex.:

```
z = parseFloat("3.15"); // z recebe 3.15
```

Instruções

A seguir veremos as principais instruções condicionais e de repetição. Veremos que seguem uma sintaxe muito parecida de outras linguagens.

if

A instrução **if** é utilizada quando é necessário tomar decisões. Sua estrutura é:

```
if (<expressão>
    <Instrução>
```

A expressão é avaliada, caso seja *true* (verdadeira) a instrução é executada, caso contrário a instrução não é executada;

Se o bloco de instrução possuir mais de uma instrução, deve-se usar início de bloco e fim de bloco, ou seja {}.

```
if (<expressão>){
    <Instrução>
}
```

Caso exista instrução a ser executada quando a expressão retornar false(falso), pode ser usar o else.

```
if (<expressão>){
    <Instrução>
}else{
    <instrução>
}
```

switch

O switch também pode ser utilizado para testes condicionais e é indicado para situações que exijam vários testes.

Sintaxe:

```
switch (variavel_de_controle) {
    case opção1 :
        comandos ;
        break;
    case opção2 :
        comandos ;
        break;
    default :
        comandos;
}
```

É importante observar que a instrução *break* é utilizada ao final de cada opção. O *break* faz com que a execução pule para o fim do *switch*. Isto é importante, uma vez que quando uma opção verdadeira for encontrada o *switch* segue executando todas as instruções até o fim se não possuir um *break*.

for

Para situações em que precisamos realizar um bloco de instrução diversas vezes, uma opção de solução é o comando de repetição *for*. Vamos supor que você tenha que escrever em seu código (X)HTML os números de 1 até 20. Para tanto, você pode utilizar o laço *for*.

Sintaxe:

```
for (<inicialização>;<condição>;<atualização variável>){
    <instruções>
}
```

Veja na figura B.4 como fica a construção do laço *for* para mostrar os números de 1 a 20. Na linha 5 foi concatenado o elemento “
” para quebrar a linha cada vez que mostra o valor da variável *x*.



while

O *while* é semelhante ao comando *for*, porém às vezes o *while* é aplicado quando não sabemos determinar a quantidade de vezes que nosso laço vai executar as instruções. Neste laço, enquanto a condição do *while* for verdadeira as instruções serão executadas.

Sintaxe:

```
while(<condição>){
    <instruções>
}
```

Dicas:

Cuidado para que seu laço não seja infinito. A condição precisa ser falsa em algum momento para sair do laço ou pode usar o comando *exit* para sair do laço.

A figura B.5 apresenta o exemplo que mostra os valores de 1 até 10. A variável *i* é que controla o laço. Observe que dentro do comando que mostra o valor da variável *i*, ela já é incrementada (*i++*).

```

1 <script type="application/javascript">
2
3     var i=1;
4     while(i <= 10) {
5
6         document.write(i++ + "<br>");
7     }
8
9 </script>

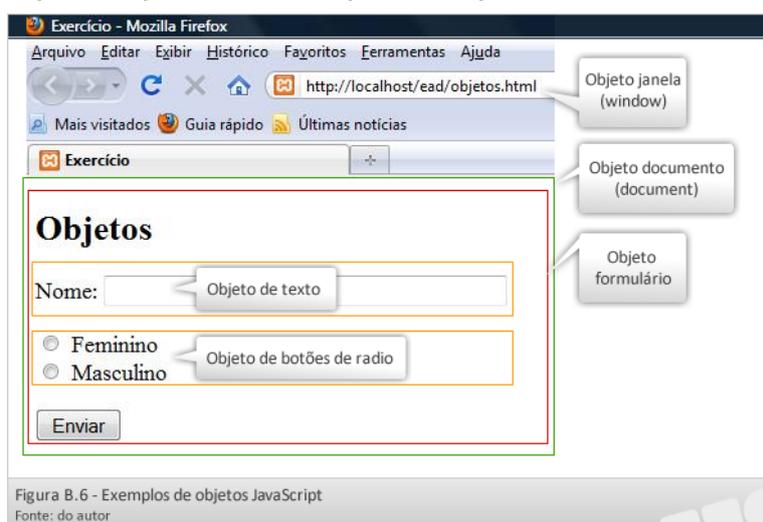
```

Figura B.5 - Exemplo laço while
Fonte: do autor

Objetos JavaScript

Em JavaScript é utilizado o conceito de objetos para manipulação de todos os elementos. Tudo pode ser considerado objeto no mundo (uma caneta, um animal, uma pessoa, uma mesa). Mas vamos ver como a linguagem JavaScript trabalha baseada em objetos. Na figura B.6 podemos ver vários objetos. A janela do navegador é um objeto, o próprio documento é considerado um objeto, um formulário, uma caixa de texto, e assim por diante. Assim, existe uma hierarquia para manipulação dos objetos:

Janela(window) → documento(document) → formulário → elemento



O objeto janela será acessado por meio do termo *window* e o objeto documento usando *document*. Para acessar o formulário e os demais elementos do formulário, é necessário saber a sua identificação. Vejamos o exemplo da figura B.7. Os objetos possuem um *name* e um *id*, que geralmente tem o mesmo valor (o radio é uma exceção). Nunca teremos dois objetos com o mesmo *id*, pois ele identifica de forma única o objeto. Já em relação a propriedade *name*, podemos ter mais de um objeto com o mesmo *name*. Observe nos balões o *id/name* de cada objeto.

The screenshot shows a web form with the following elements:

- Nome:** A text input field with the label "Nome:" and an ID of "nome".
- Sexo:** Two radio buttons labeled "Feminino" and "Masculino" with a common name of "sexo".
- Estado Civil:** A dropdown menu with the label "Estado Civil:" and an ID of "estado_civil".
- Mensagem:** A large text area with the label "Mensagem:" and an ID of "mensagem".
- Mail:** A checkbox labeled "Aceita receber e-mail" with an ID of "mail".
- Enviar:** A submit button with the label "Enviar" and an ID of "botao".

Figura B.7 - Formulário
Fonte: do autor

Só para lembrar vamos observar também o (X)HTML correspondente ao formulário acima. Veja na figura B.8. Observe que o id do formulário é *form1*.

```
<form id="form1" name="form1" method="post" action="">
  <label for="nome">Nome:</label>
  <input name="nome" type="text" id="nome" size="40" maxlength="40" /><br />
  <label> <input type="radio" name="sexo" value="F" id="sexo_0"
    Feminino</label><br />
  <label> <input type="radio" name="sexo" value="M" id="sexo_1" />
    Masculino</label> <br />
  <label for="estado_civil">Estado Civil:</label>
  <select name="estado_civil" id="estado_civil">
    <option></option>
    <option value="S">Solteiro</option>
    <option value="C">Casado</option>
    <option value="D">Divorciado</option>
    <option value="V">Viúvo</option>
    <option value="O">Outro</option>
  </select> <br />
  <label for="mensagem">Mensagem:</label>
  <textarea name="mensagem" id="mensagem" cols="45" rows="5"></textarea> <br />
  <input type="checkbox" name="mail" id="mail" />
  <label for="mail">Aceita receber e-mail</label> <br />
  <input type="submit" name="botao" id="botao" value="Enviar" />
</form>
```

Figura B.8 - Código (X)HTML do formulário
Fonte: do autor

O *id* e o *name* são propriedades dos objetos. Uma **propriedade** é um atributo, uma característica, uma descrição do objeto. Uma propriedade tem um nome e um valor associado. Para acessar as propriedades, utiliza-se a sintaxe:

```
nome_do_objeto.nome_da_propriedade
```

Vejamos um exemplo de propriedade para o objeto nome do nosso formulário da figura B.7. Se desejarmos “pegar” o valor que o usuário informar na caixa de texto nome, ou “atribuir” um valor a ele, usamos a propriedade value. Veja com o fica:

```
x = document.form1.nome.value; //pegando valor
```

```
document.form1.nome.value = 'teste';//atribuindo valor
```

Vamos ver como fica para os demais objetos do nosso formulário na tabela abaixo:

Objeto	Id	JavaScript para pegar valor
Nome: <input type="text"/>	nome	<code>document.form1.nome.value</code>
Estado Civil: <input type="text"/>	estado_civil	<code>document.form1.estado_civil.value</code> obs.: retorna o valor do item selecionado
Mensagem: <input type="text"/>	mensagem	<code>document.form1.mensagem.value</code>
<input type="checkbox"/> Aceita receber e-mail	mail	<code>document.form1.mail.value</code> Obs.: se estiver marcado retorna o valor senão retorna <i>false</i> . <ul style="list-style-type: none"> Para verificar se o campo está marcado usa-se a propriedade <i>checked</i> que retorna <i>true</i> se estiver marcado e <i>false</i> se estiver desmarcado: <code>document.form1.mail.checked</code>
<input type="radio"/> Feminino <input type="radio"/> Masculino	sexo	Obs.: os botões de radio funcionam um pouco diferente. Vejamos: <ul style="list-style-type: none"> Os botões de radio são utilizados para escolher apenas uma opção, e só uma, entre um conjunto. Por isso, todos os botões tem o mesmo nome, justamente para que seja possível desmarcar os outros botões de mesmo nome quando um for marcado. Assim, cada botão está relacionado a um índice do conjunto, começando com o índice zero(0). No nosso exemplo o primeiro botão "Feminino" é o índice zero e o botão "Masculino" o índice 1. É necessário verificar se o botão está marcado, para então pegar o valor dele. <pre> if (document.form1.sexo[0].checked) { x = document.form1.sexo[0].value }; </pre>

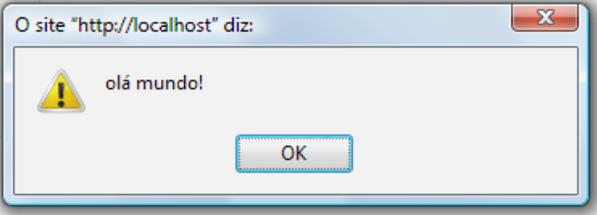
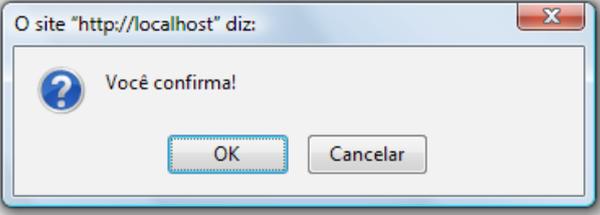
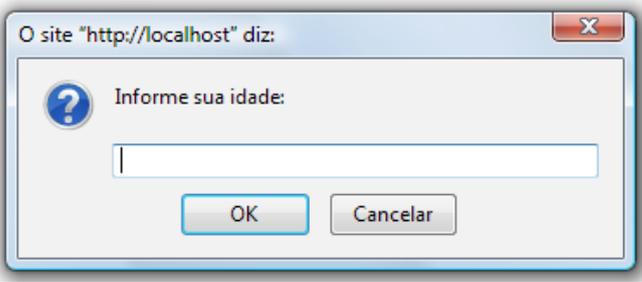
Atenção:

Lembre-se que JavaScript é *case sensitive*, ou seja diferencia maiúsculas e minúsculas.

Outro conceito importante relacionado a objetos é referente a **métodos**. Vimos que as propriedades são informações/características dos objetos, já **os métodos, são ações realizadas sobre os objetos** (gatos miam, carros correm, canetas escrevem, etc.). Vamos ver um exemplo já usado anteriormente. Sabemos que o nosso documento é considerado um objeto e vimos que podemos escrever no documento:

```
document.write('escrevendo no documento');
```

No exemplo acima, usamos o método `write()` do objeto `document` que mostra no documento o que foi passado como parâmetro, no nosso caso a `string` 'escrevendo no documento'. Vejamos outros exemplos de métodos:

Objeto/método	Finalidade
<code>window.alert()</code>	Mostra uma mensagem de alerta. <code>window.alert('olá mundo!');</code> 
<code>window.confirm()</code>	Mostra uma mensagem de confirmação com botões de OK e Cancelar. Se o botão OK for acionado retorna <code>true</code> , se acionar o botão Cancelar retorna <code>false</code> . <code>window.confirm('Você confirma!');</code> 
<code>window.prompt()</code>	Mostra uma caixa de diálogo para entrada de dados. O segundo parâmetro é referente ao valor padrão que aparece na caixa de texto. Ao clicar no botão OK o valor informado será retornado. <code>window.prompt('Informe sua idade:', '');</code> 

Métodos

Agora que conhecemos um pouco sobre métodos, podemos ver outra forma de manipular os objetos. Vamos ver o método `getElementById()` do objeto `document`. Este método retorna o objeto de qualquer elemento do documento que tenha o `id` passado por parâmetro. Por exemplo, para pegar o valor do objeto `nome` do nosso formulário da figura B.7, usamos:

```
var x = document.getElementById('nome').value;
```

Ou

```
var obj = document.getElementById('nome');
var x = obj.value;
```

Quando 'pegamos' o objeto podemos manipular seus atributos e métodos como mostrado no exemplo

acima, com a variável *obj* que recebeu o objeto *nome*.

Eventos JavaScript

Eventos são disparados a partir de algumas ações que ocorrem com os objetos, por exemplo, quando se clica em algo, quando passa o mouse sobre um objeto, quando algum objeto recebe o foco. Nestes casos existem situações que precisamos fazer algo, por isso existem os eventos. Vejamos alguns eventos importantes:

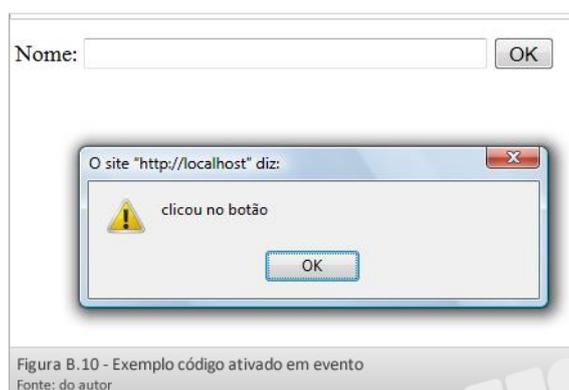
Eventos	
click	Quando clica sobre um botão, um <i>link</i> ou outros elementos.
load	Quando a página é carregada pelo navegador (aplicado ao elemento <i>body</i>).
unload	Quando saia da página (aplicado ao elemento <i>body</i>).
mouseover	Quando coloca o ponteiro do mouse sobre um <i>link</i> ou outro elemento.
mouseout	Quando o ponteiro do <i>mouse</i> sai de um <i>link</i> ou outro elemento.
focus	Quando um elemento de formulário tem o foco, isto é, está ativo.
blur	Quando um elemento de formulário perde o foco, isto é, quando o deixa de estar ativo.
change	Quando o valor de um campo de formulário é modificado.
select	Quando seleciona um campo dentro de elemento <i>select</i> de um formulário.
submit	Quando clica sobre o botão <i>Submit</i> para enviar um formulário.

Vamos ver um exemplo na figura B.9. O elemento botão, cujo id é *botao*, está com um código no evento *onclick*. Veja que está se programando em JavaScript dentro das aspas duplas. Isto pode ser feito dentro de um evento que o navegador vai interpretar corretamente.

```
<form id="form1" name="form1" method="post" action="">
<label for="nome">Nome:</label>
<input name="nome" type="text" id="nome" size="40" maxlength="40" />
<input type="submit" name="botao" id="botao" value="OK"
onclick="alert('clicou no botão');" />
</form>
```

Figura B.9 - Código (X)HTML do formulário
Fonte: do autor

O resultado será uma mensagem de alerta quando o botão for clicado. Veja na figura B.10.



Eventos são bastante utilizados em JavaScript, principalmente para chamada de funções predefinidas pelo programador que veremos a seguir.

Funções

Uma função é um conjunto de instruções destinado a uma tarefa bem específica e que podemos utilizar várias vezes. A utilização de funções melhora bastante a leitura do script. Em Javascript existem dois tipos de funções: as funções próprias do Javascript, que chamamos de “métodos”, como já visto, e as funções escritas pelo programador. É este último tipo que nos interessa agora.

Para declarar ou definir uma função, utiliza-se a palavra `function`:

```
function nome_da_funcao(parâmetros) {  
  
    // instruções  
  
}
```

Atenção:

Lembre-se que JavaScript é case sensitive e que o nome de uma função deve ser único, ou seja não podemos ter duas funções com o mesmo nome

Os parâmetros são opcionais, mas os parênteses devem sempre aparecer. Uma função só é executada quando chamamos a função, ou seja, apenas a sua definição não executa as instruções pertencentes a ela. A chamada ou invocação de uma função se faz pelo nome da função com parênteses, por exemplo:

```
nome_da_funcao();
```

Atenção:

A função deve estar definida antes de ser chamada, caso contrário um erro será gerado

Uma função pode retornar um valor, neste caso usa-se a palavra reservada `return`. Vamos a um exemplo. A figura B.11 apresenta o código JavaScript que cria uma função que recebe dois valores por parâmetro. A função soma os dois valores e retorna o resultado. A definição da função começa na linha 9 e termina na linha 14. A seguir na linha 16 é feita a chamada da função passando os dois valores para cálculo. Como a função retorna o valor total, será mostrado no navegador o resultado conforme figura B.12.

Dicas:

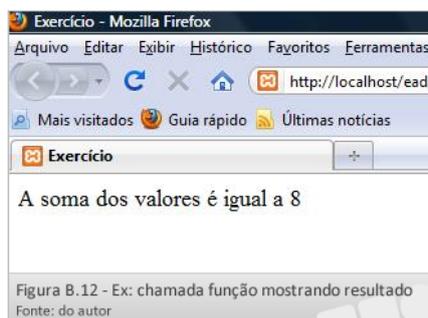
É convenção inserir todas as declarações de funções no cabeçalho da página, isto é entre os elementos `<head>...</head>`

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml">
4 <head>
5 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
6 <title>Exercício</title>
7 <script type="text/javascript">
8
9 function somar(v1, v2) {
10     // Dois valores recebidos por parâmetro.
11     var total = v1 + v2;
12     // Retornando valor calculado
13     return total;
14 }
15
16 document.write("A soma dos valores é igual a " + somar(3,5));
17
18 </script>
19 </head>

```

Figura B.11 - Exemplo função
Fonte: do autor



Com as funções, é importante compreender o uso de **variáveis locais** e **globais**.

Uma variável declarada dentro uma função pela palavra chave **var** será válida apenas dentro da própria função. Não se pode assim usá-la fora da função. Chamamos assim variável local. Se a variável é declarada contextualmente (sem utilizar a palavra var), a sua invocação será global. As variáveis declaradas logo no início do script, fora e antes de todas as funções, serão sempre globais, quer ela seja declarada com var ou de maneira contextual.

Atividades – Parte 1

1. Faça um código JavaScript que mostra uma caixa de diálogo (prompt) para o usuário informar sua idade. Se o usuário informar uma idade menor que 18 mostrar em uma caixa de alerta a mensagem “Menor de idade”. Se o usuário informar um valor maior ou igual a 18, mostrar mensagem “Maior de idade”.
2. Crie um formulário de contato e torne os campos: nome, e-mail e assunto obrigatórios, validando-os através de uma função JavaScript. Para cada campo não preenchido mostre uma mensagem de alerta.
3. No processo de seleção para cargos de uma empresa, o candidato faz 3 provas, cujos os pesos são:
 - Prova 1 = peso 3
 - Prova 2 = peso 3
 - Prova 3 = peso 4

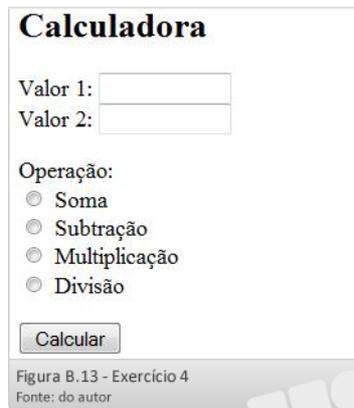
Para ser considerado aprovado o candidato deve ter média acima de 7 para o cargo de gerente e acima de 8 para o cargo de diretor. Faça uma função JavaScript para calcular a nota final do candidato e mostra em uma caixa de alerta sua situação (APROVADO/REPROVADO) considerando o formulário abaixo.

```
<form name="form1" method="post" action="cargo.php">
```

```

Nota prova 1 : <input name="p1" type="text" id="p1"> <br>
Nota prova 2 : <input name="p2" type="text" id="p2"> <br>
Nota prova 3 : <input name="p3" type="text" id="p3"> <br>
Cargo:
<select value="cargo">
  <option></option>
  <option value="7">Gerente</option>
  <option value="8">Diretor</option>
</select>
<input name="verificar" type="submit" id="Verificar" value="Verificar">
</form>
    
```

4. Crie a função *calculadora()* que considera os objetos do formulário (dois valores e a operação que pode ser +, -, * ou /). A função deve fazer o cálculo correspondente a operação e mostrar o resultado em uma caixa de alerta. Considere um formulário como da figura abaixo. Faça a chamada da função no evento *onclick* do botão



5. Dado o enunciado abaixo marque a opção correta para a estrutura de função:

Faça uma função que recebe por parâmetro um nome de objeto e dois valores. A função deve fazer a média dos dois valores recebidos e mostrar o resultado no objeto cujo nome foi recebido por parâmetro.

<p>a)</p> <pre> function fteste(x, a, b){ m = (a + b) /2; obj = document. getElementById('x'); x.value = m; if * } </pre>	<p>b)</p> <pre> function fteste(x, a, b){ m = (a + b) /2; obj = document. getElementById(x); obj.value = m; if * } </pre>
<p>c)</p> <pre> function fteste(x, a, b){ m = (a + b) /2; obj = document. getElementById(x); x.value = m; if * } </pre>	<p>d)</p> <pre> function fteste(x, a, b){ m = (a + b) /2; x.value = m; if * } </pre>

6. Dado o enunciado abaixo marque a opção correta para a estrutura de função:

Faça uma função que recebe por parâmetro dois objetos. A função deve concatenar os dois valores dos objetos e mostrar na div cujo id é 'prova'.

a)

```
function fteste(a, b){  
    v1 = document.getElementById(a).value;  
    v2 = document.getElementById(b).value;  
    document.getElementById('prova').value = v1+v2;  
}
```

b)

```
function fteste(a, b){  
    p = a + b;  
    document.getElementById('prova').innerHTML = p;  
}
```

c)

```
function fteste(a, b){  
    p = a.value + b.value;  
    document.getElementById('prova').innerHTML = p;  
}
```

d)

```
function fteste(a, b){  
    p = a.value + b.value;  
    document.getElementById('prova').value = p;  
}
```

7. Pesquise e apresente atributos não vistos neste documento para os objetos: *window*, *document* e objeto de formulário *select*.

8. Pesquise e apresente métodos não vistos neste documento para os objetos: *window*, *document* e objeto de formulário *input*.

9. Apresente um exemplo de uso de um evento em elementos de formulário (diferente de *onclick*).

10. Como uma função JavaScript pode ser chamada em um *link*? Apresente um exemplo.

Introdução a linguagem JavaScript – Parte 2

Continuando a unidade sobre linguagem do lado cliente, nesta parte 2 do material abordaremos a manipulação de arrays, strings e datas em JavaScript. Ao final desta parte são apresentadas as atividades para realização.

Arrays

Array, ou vetor, também é tratado como um objeto em JavaScript. O objeto Array é uma lista de elementos indexados nos quais pode-se guardar (escrever) dados ou as invocar (ler). O primeiro elemento do Array está no índice zero(0). Para criar um Array precisamos inicialmente definir a sua estrutura:

```
var nome_do_array = new Array (num);
```

Onde *num* é o número de elementos do array, sendo o máximo 255.

Para atribuir valores ao Array:

```
nome_do_array [i] = "valor";
```

Onde *i* é um número compreendido entre 0 e *num* menos 1.

Exemplo: uma lista de quatro frutas

```
var frutas = new Array(4);  
  
frutas[0]="maça";  
  
frutas[1]="banana";  
  
frutas[2]="abacaxi";  
  
frutas[3]="lima";
```

ou podemos definir o vetor da forma abaixo:

```
var frutas = new Array('maça','banana','abacaxi','lima');
```

Para mostrar os dados de um array é muito comum usar o uma estrutura de repetição. Vejamos um exemplo utilizando o laço *for*.

```
for(i=0; i <= frutas.length -1; i++){  
  
    document.write (<br/> + frutas[i]);  
  
}
```

Veja que utilizamos a propriedade *length* do vetor para recuperar o número de elementos do vetor. Assim, se temos um vetor de tamanho 4 (quatro elementos), nosso laço vai variar de *i* valendo 0 até *i* igual a 3.

Alguns métodos do objeto Array:

join()	Junta todos os elementos do array em uma única cadeia. Os elementos são separados por um caractere separador especificado no argumento. Por padrão, este separador é uma vírgula. <pre>var x = frutas.join(); document.write(x);</pre>
sort()	Ordena os elementos de forma crescente. <pre>frutas.sort(); for(i=0; i <= frutas.length -1; i++){ document.write ('
' + frutas[i]); }</pre>
reverse()	Inverte a ordem dos elementos no array. <pre>frutas.reverse(); for(i=0; i <= frutas.length -1; i++){ document.write ('
' + frutas[i]); }</pre>

Strings

String é uma seqüência de letras, dígitos, caracteres de pontuação e outros, que são representados pela linguagem como texto. Strings literais podem ser usadas delimitando por pares de aspas simples('...') ou aspas duplas ("..."). Em JavaScript, strings também podem ser manipuladas como objetos. A seguir veremos as principais propriedades e métodos associados a strings. Para criar explicitamente um objeto String a sintaxe é:

```
var texto = new String();
```

ou ainda:

```
var texto = new String('olá mundo!');
```

Considerando o objeto texto criado acima, vejamos a propriedade `length` que é bastante utilizada:

```
var tamanho = texto.length;
```

A propriedade `length` etorna o tamanho da string, ou seja, quantos caracteres possui.

Agora vamos ver na tabela abaixo os principais métodos do objeto String:

charAt()	Extraí o caractere em uma dada posição da string <pre>var caracter = texto.charAt(4); //retorna m</pre>
-----------------	----------------------------------------------------------------------------------------------------------------------------

<code>indexOf()</code>	<p>Pesquisa um caractere ou substring em uma string. Retorna a posição da primeira ocorrência da substring que aparece depois da posição inicial. Retorna -1 se não encontrar.</p> <pre>var retorno= texto.indexOf('!',0); // neste exemplo está procurando o caractere ! a partir da posição 0(zero) // retorna 9</pre>
<code>split()</code>	<p>Divide um string em um array de strings, quebrando em um string delimitador.</p> <pre>var texto = '10/10/2012'; var vetor = texto.split('/'); // vetor conterá 3 elementos que foram separados pelo delimitador / // retorna vetor['10','10','2012']</pre>
<code>substring()</code>	<p>Extrai um substring de um string.</p> <pre>var texto = 'Olá Mundo!'; var partetexto = texto.substring(4,9); //observe que os parâmetros são posição inicial e posição final; porém o caractere da //posição final não é incluído no retorno. // partetexto recebe 'Mundo'</pre>
<code>toLowerCase()</code>	<p>Retorna uma cópia da string, com todos os caracteres convertidos em letras minúsculas.</p> <pre>var texto = 'Olá Mundo'; var novotexto = texto.toUpperCase(); // novotexto recebe 'olá mundo'</pre>
<code>toUpperCase()</code>	<p>Retorna uma cópia da string, com todos os caracteres convertidos em letras maiúsculas.</p> <pre>var texto = 'Olá Mundo'; var novotexto = texto.toUpperCase(); // novotexto recebe 'OLÁ MUNDO'</pre>

Data e Hora

Trabalhar com datas e horas também é muitas vezes necessário para incorporar algum recurso nas páginas (X)HTML. Por exemplo, mostrar a data atual, mostrar uma saudação de acordo com o horário. Em JavaScript, data e hora são manipulados como objeto, portanto precisamos criar um objeto para conseguir manipulá-los:

```
var datahora = new Date();
```

Atenção:

A “data e hora” retornadas no exemplo acima são do computador do utilizador. Lembre-se JavaScript é interpretado pelo navegador

Se mostrar a variável datahora será mostrado no navegador algo semelhante a:

```
Tue Jun 14 2011 20:41:13 GMT-0300 (Hora oficial do Brasil)
```

Quando não passamos parâmetros na criação do objeto será retornada a data hora atual. No entanto, podemos criar um objeto data passando alguns valores. Exemplo:

```
var data = new Date(2000, 11, 30, 22, 50, 0);
```

Os valores passados por parâmetro são na sequência: ano, mês, dia, hora, minutos e segundos. Os valores para hora são opcionais, ou seja, podemos criar um objeto passando apenas os valores da data, como no exemplo abaixo:

```
var data = new Date(2000, 11, 30);
```

Agora que sabemos criar o objeto vamos ver os métodos que podemos usar na tabela abaixo:

getDate()	Retorna um inteiro correspondente ao dia do mês entre 1 e 31. <pre>var dia = datahora.getDate();</pre>
getDay()	Retorna um inteiro correspondente ao dia da semana compreendido entre 0 e 6 (0 para Domingo, 1 para Segunda-feira, ...). <pre>var dia_semana = datahora.getDay();</pre>
getMonth()	Retorna um inteiro correspondente ao mês entre 0 e 11 (0 para janeiro, 1 para fevereiro, 2 para março, ...). <pre>var mes = datahora.getMonth();</pre>
getFullYear()	Retorna um inteiro correspondente ao ano. <pre>var dia = datahora.getFullYear();</pre>
getHours()	Retorna um inteiro correspondente a hora entre 0 e 23. <pre>var dia = datahora.getHours();</pre>
getMinutes()	Retorna um inteiro correspondente aos minutos entre 0 e 59. <pre>var dia = datahora.getMinutes();</pre>
getSeconds()	Retorna um inteiro correspondente aos segundos entre 0 e 59. <pre>var dia = datahora.getSeconds();</pre>

Atividades – Parte 2

1. Considerando o formulário abaixo, faça uma função JavaScript que valida os campos para os seguintes critérios:

- todos os campos são obrigatórios
- o nome do usuário deve ter no mínimo 8 caracteres
- a senha deve conter no mínimo 6 caracteres
- os campos senha e senhaconf devem possuir o mesmo valor

Mostrar alertas para o usuário para cada inconsistência encontrada.

```
<form name="form1" method="post" action="">

Nome:<input name="nome" type="text" id="nome" size="20"> <br>

Senha:<input name="senha" type="text" id="senha" size="20"><br>

Confirma senha: <input name="senhaconf" type="text" id="senhaconf" size="20">
<br>

<input name="Enviar" type="button" id="Enviar" value="Enviar">

</form>
```

2. Dado um formulário com um campo para informar o e-mail, faça uma função JavaScript que verifica o e-mail para os seguintes critérios:

- campo é de preenchimento obrigatório
- e-mail deve possuir no mínimo 6 caracteres
- e-mail deve possuir o caractere @
- e-mail deve possuir o caractere . (ponto)

Crie um formulário para usar a função.

3. Faça uma função JavaScript que recebe por parâmetro uma data e retorna o dia da semana por extenso (Domingo, Segunda-feira, Terça-feira, ...). Mostre um exemplo de uso da função.

4. Faça uma função JavaScript que recebe por parâmetro uma data e retorna o mês por extenso. Mostre um exemplo de uso da função.

5. Faça uma função JavaScript que retorna a data atual e uma saudação no seguinte formato:

14/06/2011 - Boa noite!

6. Dado o formulário abaixo, faça uma função que calcule o gasto calórico considerando o peso, a atividade desenvolvida e o tempo que a atividade foi realizada. Mostre o gasto calórico na *div* mensagem. Considere a seguinte fórmula para o cálculo:

$$\text{peso} * \text{tempo} * (\text{valor da atividade física}) * 0.0175$$

Dicas:

Para verificar qual a atividade que foi selecionada, você pode utilizar um laço. Lembre-se que o botão de rádio considera cada opção como um índice (atividades[0], atividades[1], atividades[2],...).

Segue o código (X)HTML do formulário acima:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Gasto Calórico</title>
<script src="funcao_calorias.js" type="text/javascript">
</script>
</head>
<body>
<form id="form1" name="form1" method="post" action="">
  <table width="600" border="0" align="center" cellpadding="4" cellspacing="2">
    <tr> <td colspan="2" align="center"><h3><strong>Cálculo do gasto calórico
</strong></h3></td>
    </tr>
    <tr>
      <td width="185" align="right"><label for="peso">Peso:</label></td>
      <td width="393"><input name="peso" type="text" id="peso" size="8" /></
td>
    </tr>
    <tr>
      <td align="right">Atividades:</td>
      <td><p>  <label>
        <input type="radio" name="atividades" value="3" id="atividades_0" />
        Vôlei</label>      <br />
        <label><input type="radio" name="atividades" value="4"
id="atividades_1" />
        Remo</label>      <br />
        <label><input type="radio" name="atividades" value="4.5"
id="atividades_2" />
        Surfe</label>      <br />
        <label><input type="radio" name="atividades" value="5.5"
id="atividades_3" />
        Dança</label>      <br />

```

```

        <label><input type="radio" name="atividades" value="6"
id="atividades_4" />
        Futebol</label>        <br />
        <label><input type="radio" name="atividades" value="8"
id="atividades_5" />
        Basquete</label>        <br />
        <label><input type="radio" name="atividades" value="7"
id="atividades_6" />
        Ciclismo</label>        <br />
        <label><input type="radio" name="atividades" value="9"
id="atividades_7" />
        Tennis</label>        <br />
    </p></td>
</tr>
<tr>
    <td align="right"><label for="tempo">Tempo:</label></td>
    <td><select name="tempo" id="tempo">
        <option value="15">15</option>
        <option value="30">30</option>
        <option value="45">45</option>
        <option value="60">60</option>
        <option value="75">75</option>
        <option value="90">90</option>
    </select></td>
</tr>
<tr>
    <td align="right">&nbsp;</td>
    <td><input type="button" name="botao" id="botao" value="Calcular"
onClick="fcalorias()" /></td>
</tr>
<tr>
    <td align="right">&nbsp;</td>
    <td><div id="mensagem"></div></td>
</tr>
</table>
</form>
</body>
</html>

```

Introdução a linguagem JavaScript – Parte 3

Nesta última parte do material da linguagem JavaScript serão apresentados propriedades e métodos dos principais objetos JavaScript e um exemplo de resolução de problema desenvolvido passo a passo. Ao final desta parte, serão apresentadas as atividades para realização.

Objeto window

O objeto window representa a janela do navegador a qual o script está em execução. Este objeto possibilita o acesso a propriedades e execução de métodos. Assim, a janela do navegador pode ser manipulada de diferentes formas, por exemplo, modificando o seu tamanho, aparência ou posição, podemos abrir e fechar janelas, transferir informações entre janelas e criar janelas de diálogo.

A tabela abaixo apresenta as principais propriedades do objeto **window**:

defaultStatus	Determina o conteúdo padrão da barra de status do navegador, quando nada de importante estiver acontecendo. Ex: <code>window.defaultStatus= 'texto'</code>
status	Define uma mensagem que aparecerá na barra de status do navegador, em substituição, por exemplo, a URL de um link, quando o mouse estiver sobre o link. Ex: <code>window.status="texto"</code>
name	Contém o nome da janela.

A tabela abaixo apresenta os principais métodos associados ao objeto **window**:

open("URL") ou open("URL", "nome") ou open("URL", "nome", "características")	Abre uma nova janela contendo o documento indicado pela URL. Opcionalmente, a janela pode ter um nome que pode ser usado em HTML, ou especificar características como tamanho, layout, etc. O método retorna uma referência do tipo window para a janela criada: <code>filha = window.open("filha.htm");</code>
close()	Fecha uma janela.
focus()	Torna uma janela ativa (traz para frente das outras, se for uma janela independente)

As janelas abertas podem ter suas características alteradas no momento em que são abertas. Estas características deverão vir em uma string com uma lista de opções separadas por vírgulas, como o terceiro argumento opcional do método `open()`. As características estão na tabela abaixo:

height=h	h é a altura da janela em pixels: <code>height=150</code>
width=w	w é a largura da janela em pixels: <code>width=300</code>
resizable	Se estiver presente permite redimensionar a janela
toolbar	Se estiver presente, mostra a barra de ferramentas do browser
scrollbars	Se estiver presente, mostra as barras de rolagem do browser
menubar	Se estiver presente, mostra a barra de menus do browser

location	Se estiver presente, mostra o campo para entrada de URLs
status	Se estiver presente, mostra a barra de status

Cada característica pode ou não ter um valor. Não deverá haver espaços em qualquer lugar da lista. Por exemplo:

```
window.open("teste.html", "j2", "height=200,width=400,status");
```

O código acima abre uma janela de 200 pixels de altura por 400 de largura sem barra de ferramentas, sem barra de diretórios, sem campo de entrada de URLs, sem barra de menus, não-redimensionável e com barra de status.

Atenção:

Vale lembrar os métodos do objeto **window** que já foram abordados na parte 1 desta unidade, tais sejam: *alert()*, *prompt()* e *confirm()*.

Objeto history

O objeto **history** está associado ao objeto **window** e armazena as informações sobre os URLs que foram visitados antes e depois do atual e inclui métodos para ir para as localizações anteriores ou próximas:

go(<i>n</i>) ou go("string")	Avança ou volta <i>n</i> páginas no histórico. A segunda forma procura no histórico até encontrar a primeira página que tenha a string especificada no título do documento ou nas palavras da sua URL. Ex.: <code>window.history.go(+1)</code>
back()	Volta uma página no histórico (simula o botão "Back" ou "Voltar" do browser). Ex.: <code>window.history.back()</code>
forward()	Avança uma página no histórico (simula o botão "Forward" ou "Avançar" do browser). Ex.: <code>window.history.forward()</code>

Objeto Navigator

O objeto **navigator** (associado ao objeto **window**) representa as propriedades do navegador. Usando suas propriedades e métodos é possível identificar características do navegador e desenvolver páginas personalizadas com conteúdo específico para aproveitar ao máximo os recursos existentes. Veja abaixo uma lista de propriedades vinculadas ao navegador:

userAgent	Uma string com a informação contida no cabeçalho HTTP User-Agent. Esta propriedade é a combinação das propriedades <code>appName</code> e <code>appVersion</code> . Exemplos: <pre>Mozilla/4.0 (compatible; MSIE 4.0; Windows 95)</pre> <pre>Mozilla/4.5 [en] (Win95; I)</pre>
appName	Contém o nome interno do navegador. Exemplo: Mozilla
appVersion	Contém informações sobre a versão. Exemplos: <pre>4.0 (compatible; MSIE 4.0; Windows 95)</pre> <pre>4.5 [en] (Win95; I)</pre>
appName	Contém o nome oficial do browser. Exemplos: <pre>Microsoft Internet Explorer</pre> <pre>Netscap</pre>

Objeto document

O objeto `document` representa o documento HTML atual. `document` é uma propriedade de `window` e, portanto, pode ser usado sem fazer referência a `window`:

```
window.document // ou simplesmente document
```

As principais propriedades do tipo **document** estão listadas na tabela abaixo:

title	Recupera o título do documento
location	Contém String com URL do documento. Pode ser alterado. Ex.: <pre>document.location = "http://www.ifsul.edu.br";</pre>
URL	Mesma coisa que <code>location</code> .
lastModified	Contém a data da última modificação do arquivo. Está no formato de data do sistema. Pode ser convertida usando <code>Date.parse()</code> e transformada em objeto ou automaticamente em String.

Atenção:

Com relação aos principais métodos de **document**, já vimos na parte 1 desta unidade. Vale lembrar, por exemplo, do `document.write` utilizado para escrever algo no documento.

Solução passo a passo

O objetivo de solução de um problema passo a passo é mostrar de forma integrada a utilização dos recursos estudados e que auxiliarão no desenvolvimento das atividades propostas.

Então, mãos à obra. O enunciado do nosso problema é:

O formulário abaixo solicita que o usuário informe o peso, escolha uma atividade física e o tempo em que a atividade foi realizada. Dado este formulário, calcule, de acordo com a atividade física, o peso e o tempo de exercício, quantas calorias você queima praticando esporte. O cálculo usa a seguinte fórmula:

$$\text{Gasto Calorico} = \text{peso} * \text{tempo} * (\text{valor da atividade física}) * 0.0175$$

Calcule, de acordo com a atividade física, seu peso e tempo de exercício, quantas calorias você queima praticando esporte

Peso quilos

Atividade física

- Vôlei
- Dança de salão
- Esgrima
- Tênis
- Basquete
- Judô
- Surfe
- Remo
- Ciclismo
- Hidroginástica

Tempo minutos

****Fórmula: peso * tempo * (valor da atividade física) * 0.0175**
**** Testar se os valores foram informados**

Figura B.15 - Formulário
 Fonte: do autor

Agora vamos à solução. Primeiro vamos construir o formulário:

- a) Criar um arquivo HTML nomeado exercicio.html
- b) Incluir um título para o documento (Exercício passo a passo)
- c) Incluir um elemento form
- d) Incluir um elemento table com a seguinte formatação:
 - 8 linhas
 - 2 colunas
 - width="50%"
 - border="0"
 - align="center"
 - cellpadding="0"
 - cellspacing="10"

Dica:

Vamos utilizar uma tabela com duas colunas para organizar os elementos do formulário de forma que fiquem alinhados.

Agora vamos incluir os elementos de formulário:

- a) uma caixa de texto para peso (*id= peso*)

```
<input type="text" id="peso" name="peso" />
```

- b) botões de rádio para as atividades (*name=atividade*)

```
<input type="radio" name="atividade" value="3" /> V&ocirc;lei <br />
<input type="radio" name="atividade" value="5.5" /> Dan&ccedil;a de sal&atilde;o<br />
<input type="radio" name="atividade" value="6" /> Esgrima<br />
<input type="radio" name="atividade" value="6" /> T&ecirc;nis<br />
<input type="radio" name="atividade" value="8" /> Basquete<br />
<input type="radio" name="atividade" value="10" /> Jud&ocirc;ca<br />
<input type="radio" name="atividade" value="3" /> Surfe<br />
<input type="radio" name="atividade" value="3.5" /> Remo<br />
<input type="radio" name="atividade" value="4" /> Ciclismo<br />
<input type="radio" name="atividade" value="4" /> Hidrogin&aacute;stica </td>
```

Figura B.16 - botões de rádio para as atividades
Fonte: do autor

- c) um menu de lista para o tempo (*id=tempo*)

```
<select id="tempo" name="tempo">
  <option value="15">15</option>
  <option value="30">30</option>
  <option value="45">45</option>
  <option value="60">60</option>
  <option value="75">75</option>
  <option value="90">90</option>
  <option value="105">105</option>
  <option value="120">120</option>
  <option value="135">135</option>
  <option value="150">150</option>
  <option value="165">165</option>
  <option value="180">180</option>
  <option value="195">195</option>
  <option value="210">210</option>
  <option value="225">225</option>
  <option value="240">240</option>
  <option value="255">255</option>
  <option value="270">270</option>
  <option value="285">285</option>
  <option value="300">300</option>
</select> minutos
```

Figura B.17 - Menu de lista para o tempo
Fonte: do autor

- d) uma div para o resultado (*id=resultado*)
- e) um botão (*id=btnCalcular*)

A parte de programação JavaScript será feita em um arquivo separado (js.js). Esta é a forma recomendada de incluirmos nossas funções JavaScript no HTML, uma vez que geralmente temos várias funções que são utilizadas nas nossas páginas. O nome da nossa função será `calcula()` e será chamada no evento `onClick` do botão `Calcular`.

Antes de partirmos para a programação JavaScript vamos analisar todo o código HTML que é mostrado abaixo em duas figuras.

A figura B.18 mostra o código do arquivo `exercicio.html` da linha 1 até a linha 30.

```
1 <html>
2 <head>
3 | <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
4 <title>Exerc&iacute;cio passo a passo</title>
5 <script language="javascript" src="js.js"></script>
6 </head>
7 <body>
8 <form id="form1" name="form1" method="POST" action="">
9 <table width="50%" border="0" align="center" cellpadding="0" cellspacing="10">
10 <tr>
11 <td colspan="2" align="center"><strong>Calcule, de acordo com a atividade f&iacute;sica, seu peso e
12 tempo de exerc&iacute;cio, <br> quantas calorias voc&ecirc;o; queima praticando esporte</strong></td>
13 </tr>
14 <tr>
15 <td align="right" colspan="2"><b>Peso</b></td>
16 <td align="left" colspan="2"><input type="text" id="peso" name="peso" />
17 </tr>
18 <tr>
19 <td align="right" colspan="2"><b>Atividade f&iacute;sica</b></td>
20 <td align="left" colspan="2"><input type="radio" name="atividade" value="3" /> V&ocirc;lei <br />
21 <input type="radio" name="atividade" value="5.5" /> Dan&ccedil;a de sal&atilde;o<br />
22 <input type="radio" name="atividade" value="6" /> Esgrima<br />
23 <input type="radio" name="atividade" value="6" /> T&ecirc;nis<br />
24 <input type="radio" name="atividade" value="8" /> Basquete<br />
25 <input type="radio" name="atividade" value="10" /> Jud&ocirc;ca<br />
26 <input type="radio" name="atividade" value="3" /> Surfe<br />
27 <input type="radio" name="atividade" value="3.5" /> Remo<br />
28 <input type="radio" name="atividade" value="4" /> Ciclismo<br />
29 <input type="radio" name="atividade" value="4" /> Hidrogin&aacute;stica </td>
30 </tr>
```

Figura B.18 - Código do arquivo HTML - parte 1
Fonte: do autor

A figura B.19 apresenta a código do arquivo HTML das linhas 31 à 60.

```

31 <tr>
32 <td width="120" align="right"><b>Tempo</b></td>
33 <td>
34 <select id="tempo" name="tempo">
35 <option value="15">15</option> <option value="30">30</option>
36 <option value="45">45</option> <option value="60">60</option>
37 <option value="75">75</option> <option value="90">90</option>
38 <option value="105">105</option> <option value="120">120</option>
39 <option value="135">135</option> <option value="150">150</option>
40 <option value="165">165</option> <option value="180">180</option>
41 <option value="195">195</option> <option value="210">210</option>
42 <option value="225">225</option> <option value="240">240</option>
43 <option value="255">255</option> <option value="270">270</option>
44 <option value="285">285</option> <option value="300">300</option>
45 </select> minutos
46 </td>
47 </tr>
48 <tr>
49 <td align="right">&nbsp;&nbsp;&nbsp;</td>
50 <td align="right"><strong>Fórmula: peso * tempo * (valor da atividade física) * 0.0175<br />
51 Testar se os valores foram informados <br /> </strong></td>
52 </tr>
53 <tr>
54 <td colspan="2" align="right"><div id="resultado"></div></td> </tr>
55 <tr>
56 <td colspan="2" align="center"><input name="btnCalcular" type="button" id="btnCalcular"
57 onclick="calcula()" value="Ver resultado"></td>
58 </tr>
59 </form>
60 </body>

```

Em minutos

Chamada da função calcula()

Figura B.19 - Código do arquivo HTML - parte 2
Fonte: do autor

Tendo nosso arquivo *exercicio.html* completo, vamos à codificação JavaScript. A figura B.20 apresenta a função *calcula()* do arquivo *js.js*.

```

2 function calcula() {
3
4     peso = document.getElementById('peso').value;
5     tempo = document.getElementById('tempo').value;
6     atividade = document.getElementsByName('atividade');
7     valor_atividade = 0;
8
9     mensagem = "";
10    if (peso == "")
11        mensagem = "Informe o peso";
12
13    if (tempo == "")
14        mensagem += "<br /> Informe o tempo";
15
16    flag = false;
17    for(j=0;j<atividade.length;j++){
18        if (atividade[j].checked){
19            flag = true;
20            valor_atividade = parseFloat(atividade[j].value);
21        }
22    }
23
24    if (flag == false)
25        mensagem += "<br /> Selecione a atividade";
26
27    if (mensagem == ""){
28        calorias = parseFloat(peso) * parseInt(tempo) * valor_atividade * 0.0175;
29        mensagem = calorias.toFixed(2) + " calorias gastas ";
30    }
31
32    document.getElementById('resultado').innerHTML= mensagem;
33 }

```

Figura B.20 - função calcula() do arquivo js.js
Fonte: do autor

Agora vamos analisar todo o código da função:

Linha 2: Definição da função (não recebe parâmetros).

Linhas 4 e 5: Pega os valores dos elementos cujo *id* são respectivamente peso e tempo.

Atenção:

O método *getElementById* já foi visto na parte 1 desta unidade sobre JavaScript. Este método retorna o objeto de qualquer elemento do documento que tenha o *id* passado por parâmetro.

Linha 6: Veja que para pegar a atividade foi utilizado um método diferente (*document.getElementsByName*), isso porque existem vários elementos do tipo rádio com o mesmo name (neste caso atividade - quando um item é selecionado pelo usuário os demais itens com o mesmo nome são desmarcados). O método

`document.getElementsByName` retorna um *array*, contendo todos os elementos que possuem o nome passado por parâmetro. Nesse caso, vamos precisar de um laço para percorrer todo o *array*.

Linha 7: Inicia variável `valor_atividade` com zero. Essa variável será utilizada para pegar o valor correspondente à atividade selecionada.

Linhas 9 a 14: Inicialmente é criada uma variável `mensagem` sem valor. Nessas linhas, é verificado se foram informados valores para o peso e tempo. Caso não tenha sido informado, é atribuída mensagem para a variável `mensagem`. Observe que na linha 14 é feita uma concatenação (`+=`). Isso significa que a variável `mensagem` mantém o seu valor e concatena com o novo valor passado.

Linhas 16 a 22: Este conjunto de código tem por objetivo verificar se alguma atividade foi selecionada e neste caso pegar o valor correspondente.

- Linha 16: cria variável `flag` com valor `false`.
- Linha 17: laço `for` que percorre todas as opções de atividades (`atividade.length` retorna o número de opções, neste caso 10).
- Linha 18: verifica a propriedade `checked` da atividade. Se for `true` significa que está marcada.
- Linha 19: atribui `true` para variável `flag`. Significa que tem atividade selecionada.
- Linha 20: pega o valor correspondente à atividade. A função `parseFloat` converte o valor que é string para o tipo `float` necessário para fazer o cálculo.

Linhas 24 e 25: Verifica o valor de `flag`. Se continuar com `false` atribui uma mensagem.

Linha 27: Verifica se mensagem possui valor. Caso não possua valor procede ao cálculo.

Linha 28: Realiza o cálculo. Utiliza as funções `parseFloat` e `parseInt` para converter os valores respectivamente para `Float` e `Int` (necessário para realizar o cálculo).

Linha 29: Atribui uma mensagem com o resultado do cálculo. A função `toFixed(2)` (`calorias.toFixed(2)`) faz com que o valor apresente apenas 2 casas decimais.

Linha 32: Atribui o valor da variável `mensagem` para a `div resultado`.

Nossa solução está completa. Lembre-se de que um problema pode ter diferentes formas de solução. Aqui apresentamos uma solução que busca mostrar de forma didática cada passo realizado para a solução do problema.

Atividades – Parte 3

1. Crie duas páginas (X)HTML como mostrado na figura abaixo. A primeira página deve ter um botão “Abre Janela” que, quando clicado, deve abrir uma nova janela nas dimensões de 360 de largura x 280 de altura (pixels). Depois de aberta, a nova janela deverá estar na frente da antiga (use `focus()`). A segunda página deve conter um botão fechar que, ao ser ativado, fecha a janela.

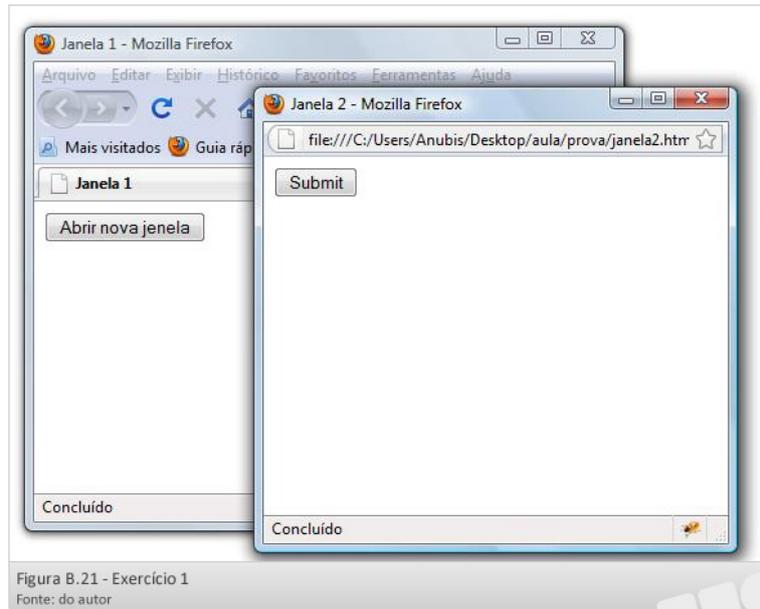


Figura B.21 - Exercício 1
Fonte: do autor

2. Considerando o código (X)HTML do formulário cujo código é disponibilizado abaixo, faça:

- Validar se todas as perguntas foram respondidas. Caso haja perguntas não respondidas mostrar na div "resultado";
- Calcular a pontuação de acordo com o valor de cada opção e mostrar na caixa "pontos".
- Mostrar na div "resultado" o diagnóstico de risco conforme pontuação calculada (conforme abaixo).

Risco baixo (inferior a 6)

Com pouca probabilidade de sofrer doenças do coração na próxima década, se for preciso, faça algumas mudanças em seu estilo de vida para manter o seu risco cardíaco baixo. Se você fuma, abandone o cigarro. Se é sedentário, engaje-se numa rotina de exercícios físicos. Tenha uma dieta equilibrada e mantenha-se no peso ideal

Risco moderado (entre 6 e 8)

Ainda que você não sinta nada, é grande a probabilidade de que suas artérias já acumulem placas moles de gordura. O risco de você vir a ser acometido por males cardíacos nos próximos dez anos pode ser minimizado com alterações em seus hábitos de vida. Talvez o seu médico recomende que você tome um comprimido de aspirina diariamente. Se o nível de LDL for igual ou superior a 130, é o caso de recorrer às estatinas, os medicamentos contra o colesterol alto

Risco alto (superior a 8)

É certo que o médico adotará medidas preventivas mais agressivas. Além da aspirina, ele poderá indicar o uso de estatinas, os remédios contra o colesterol alto, e medicamentos para baixar a pressão arterial. A adoção de hábitos saudáveis de vida é urgente

Figura B.22 - Diagnóstico de Risco
Fonte: do autor

The image shows a web form with six questions, each with radio button options:

- Perg1: 1. Qual a sua idade?**
 - até 34 anos
 - de 35 a 39 anos
 - de 40 a 44 anos
 - de 45 a 49 anos
 - de 50 a 54 anos
 - de 55 a 59 anos
 - de 60 a 64 anos
 - de 65 a 69 anos
 - Mais de 70 anos
- Perg2: 2. Qual é a sua pressão arterial?**

Sistólica	Diastólica
inferior a 12	inferior a 8
de 12 a 12,9	de 8 a 8,4
de 13 a 13,9	de 8,5 a 8,9
de 14 a 15,9	de 9 a 9,9
16 ou mais	10 ou mais
- Perg3: 3. Qual é o seu nível de colesterol?**
Nível de LDL, o colesterol "ruim" (em mg/dl)
 - inferior a 100
 - de 100 a 129
 - de 130 a 159
 - de 160 a 190
 - Superior a 190
- Perg4: 4. Nível de HDL, o colesterol "bom" (em mg/dl)**
 - inferior a 35
 - de 35 a 44
 - de 45 a 49
 - de 50 a 59
 - superior a 60
- Perg5: 5. Você fuma?**
 - Não
 - Sim
- Perg6: 6. Você tem diabetes**
 - Não
 - Sim

Figura B.23 - Visualização do formulário
 Fonte: do autor

Segue o código (X)HTML do formulário acima:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>PESQUISA</title>
<script language="javascript" src="js.js"></script>
</head>
<body>
<form id="form1" name="form1" method="POST" action="">
  <table width="50%" border="0" align="center" cellpadding="0" cellspacing="3"
  bgcolor="#FFFFFF">
    <tr>
      <td align="center"><strong>QUESTION&Aacute;RIO</strong></td>
    </tr>
    <tr>
      <td><p><strong>1.Qual a sua idade?</strong><br />
        <input name="perg1" type="radio" value="-1" />          at&eacute; 34
        anos<br />
        <input name="perg1" type="radio" value="0" />          de 35 a 39 anos
        <input name="perg1" type="radio" value="1" />          de 40 a 44
        anos<br />
        <input name="perg1" type="radio" value="2" />          de 45 a 49 anos
        <input name="perg1" type="radio" value="3" />          de 50 a 54
        anos<br />
        <input name="perg1" type="radio" value="4" />          de 55 a 59 anos
        <input name="perg1" type="radio" value="5" />          de 60 a 64
        anos<br />
        <input name="perg1" type="radio" value="6" />          de 65 a 69 anos
        <input name="perg1" type="radio" value="7" />          Mais de 70 anos
      </p>

      <p><strong>2.Qual &eacute; a sua press&atilde;o arterial
      (sist&oacute;lica / diast&oacute;lica )?
      </strong></p>
```


3. Considerando o formulário abaixo, faça:

- a) Validar se todas as perguntas foram respondidas. Caso haja perguntas não respondidas mostrar mensagem em uma .
- b) Mostrar o total de respostas Sim.
- c) Mostrar o total de respostas Não.

Sua dieta é saudável? O hábito de comer fora, sempre às pressas, faz com que seja ainda mais difícil manter uma dieta equilibrada. Veja como anda a sua alimentação

	Sim	Não
Nos últimos seis meses, seu peso vem se mantendo dentro dos padrões ideais?	<input type="radio"/>	<input type="radio"/>
Você consome frutas e legumes quase todos os dias?	<input type="radio"/>	<input type="radio"/>
Você evita carnes vermelhas?	<input type="radio"/>	<input type="radio"/>
Você evita ao máximo comida de lanchonete?	<input type="radio"/>	<input type="radio"/>
Você tem o costume de substituir pão e arroz brancos por pão e arroz integrais?	<input type="radio"/>	<input type="radio"/>
Em vez de fritar, você prefere grelhar os alimentos?	<input type="radio"/>	<input type="radio"/>
Você bebe, no mínimo, oito copos (o equivalente a 2 litros) de água por dia?	<input type="radio"/>	<input type="radio"/>
Você é adepto da linha "quanto menos sal, melhor"?	<input type="radio"/>	<input type="radio"/>
TOTAL	<input type="text"/>	<input type="text"/>

Calcular Apaga o teste

AVALIAÇÃO

Se você respondeu "sim" a todas as perguntas, sua dieta é das mais saudáveis

Se você assinalou "não" em até duas questões, preste mais atenção em seu cardápio

Se você marcou "não" em três questões ou mais, cuidado. O tipo de alimentação que você tem é uma ameaça à sua saúde

Figura B.24 - Formulário para exercício 2
Fonte: do autor

4. Considerando o formulário abaixo, faça a codificação JavaScript para:

- a) Validar se todas as perguntas foram respondidas. Caso haja perguntas não respondidas mostrar mensagem em uma *div*.
- b) Mostrar na caixa de texto correspondente o número de respostas para cada cor.

Faça agora um teste que ajuda a descobrir qual o tipo de exercício mais adequado ao seu temperamento

1) Quando penso no meu final de semana, prefiro:

- Planejar as atividades com dias de antecedência
- Deixar para definir a programação na noite de sexta-feira, pois até a última hora podem surgir idéias interessantes
- Imaginar apenas programas que me estimulem intelectualmente
- Programar viagens em grupo para lugares tranquilos, com o objetivo de conviver com pessoas e manter contato com a natureza

2) Se eu fosse um líder entre meus colegas de trabalho, procuraria:

- Estimulá-los a desenvolver o potencial individual e colocar todo o seu conhecimento a serviço do grupo
- Ler obras de auto-ajuda sobre os princípios da liderança para colocá-los em prática
- Resolver todas as crises e conflitos que surgissem
- Montar estratégias para melhorar o rendimento da equipe

3) Sempre que participo de jogos ou de atividades esportivas:

- Uso estratégias que já testei anteriormente para chegar a vitória
- Gosto de variar a estratégia a cada partida
- Acho que a diversão é mais importante do que a vitória
- Utilizo mais a emoção e a inspiração

4) Meus amigos costumam dizer que:

- Sou esperto e inteligente
- Sou capaz de me divertir em qualquer situação
- Sou seguro e independente
- Sou simpático e bem-humorado

5) Em atividades que exigem planejamento, como uma reforma em casa ou a implantação de uma nova tarefa no trabalho, procuro:

- Incentivar as pessoas a apresentar suas idéias, pois várias cabeças pensam melhor do que uma
- Analisar a situação em seu conjunto antes de tomar uma decisão
- Resolver os problemas de uma vez
- Ser metódico e observar cada detalhe

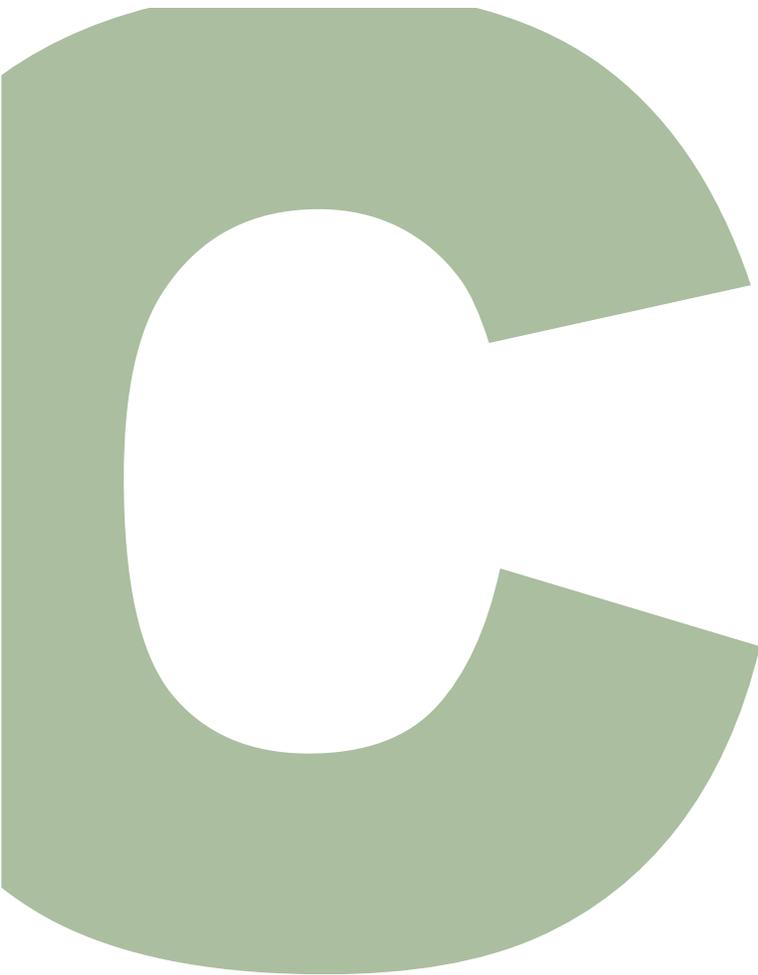
Contar

Contagem dos pontos:

Figura B.25 - Formulário para exercício 3
Fonte: do autor



TICS



Introdução a Liguagem PHP

Unidade C
Linguagem de Programação Web

INTRODUÇÃO A LINGUAGEM PHP

Linguagem PHP

Esta unidade tem por objetivo estudar uma linguagem do lado servidor, que tem seu código interpretado no servidor, sendo que o cliente recebe apenas o resultado do seu processamento. Linguagens do lado servidor são necessárias, especialmente, quando temos a necessidade de acessar um banco de dados (que fica no servidor). A maioria dos sites hoje utiliza esse tipo de linguagem, o que possibilita que as páginas do site sejam dinâmicas, ou seja, são geradas com conteúdo recuperado do banco de dados, o que facilita a manutenção destes sites.

Como já comentado na Unidade A, existem várias linguagens do lado servidor, tais como PHP, JSP, ASP e Python. No entanto, nossa disciplina estudará a linguagem PHP (*Hypertext Preprocessor*) que é atualmente uma das linguagens mais utilizadas, open-source e independente de plataforma.

Agora vamos conhecer mais sobre a linguagem PHP. A seguir abordaremos as ferramentas necessárias para iniciar a programação, a sintaxe básica da linguagem, estruturas condicionais e de repetição, manipulação de arrays, strings e datas, definição de funções e tratar sessões e cookies. Então, mãos à obra...

Características da linguagem

PHP é um acrônimo para “*Hypertext Preprocessor*” (originalmente foi chamado de Personal Home Page, mas quando começou a ganhar adeptos o seu nome foi alterado). O PHP é uma linguagem interpretada, livre, independente de plataforma e utilizada para gerar conteúdo dinâmico. Outras características importantes: pode programar estruturado e/ou orientado a objetos e é de tipagem dinâmica.

A linguagem foi criada por volta de 1994 por Rasmus Lerdorf, com o nome Personal Home Page Tools. Mais tarde, Zeev Suraski desenvolveu o analisador do PHP 3 que contava com o primeiro recurso de orientação a objetos. Pouco depois, Zeev e Andi Gutmans, escreveram o PHP 4, abandonando por completo o PHP 3, dando mais poder à linguagem e maior número de recursos de orientação a objetos. Atualmente a linguagem está na versão 5.

Com o PHP é possível fazer muitas coisas, por exemplo, coletar dados de um formulário, gerar páginas dinamicamente ou enviar e receber cookies. O PHP também tem como uma das características mais importantes o suporte a um grande número de bancos de dados, como dBase, Interbase, mSQL, MySQL, Oracle, Sybase, PostgreSQL e vários outros.

Ferramentas necessárias

Como já mencionado na Unidade A, a programação PHP (linguagem do lado servidor) é interpretada no servidor, isso significa que o PHP fica instalado no servidor e quando uma página PHP é requisitada, o PHP faz o trabalho de interpretar o código e enviar apenas o resultado para o cliente. Então, precisamos saber como vamos trabalhar enquanto estamos na fase de desenvolvimento, pois não vamos ficar

enviando nossas páginas para o servidor para testar. Por isso, vamos trabalhar com um servidor web instalado na nossa máquina e com o PHP configurado neste servidor. A indicação é usarmos a instalação do Xampp. O Xampp é um ambiente destinado a desenvolvedores e faz a instalação das ferramentas necessárias, bem como sua configuração. O ambiente é composto por:

- servidor web Apache;
- interpretadores para linguagens de script: PHP e Perl;
- base de dados MySQL.

O nome desse ambiente significa:

X = para qualquer dos diferentes sistemas operacionais;
A = Apache
M = MySQL
P = PHP
P = Perl

Atualmente XAMPP está disponível para Microsoft Windows, GNU/Linux, Solaris, e MacOS X. A sugestão é utilizar a versão 1.6.8 que é mais estável.

Dica:

O download pode ser feito pelo link:

[<http://sourceforge.net/projects/xampp/files/XAMPP%20Windows/>](http://sourceforge.net/projects/xampp/files/XAMPP%20Windows/)

Parada Obrigatória:

Assista ao vídeo “Instalação e configuração de ferramentas”, disponível em nosso Ambiente Virtual de Aprendizagem.

Estrutura do Xampp

Dentro da pasta de instalação do Xampp (provavelmente `c:\xampp`) está a pasta *htdocs* (documentos de hipertexto). Essa pasta é muito importante, pois todos os arquivos que criaremos ficarão dentro dela. Para iniciar com nossos exemplos, crie uma pasta chamada *curso* dentro de *htdocs* (`c:\xampp\htdocs\curso`). Em *htdocs* podemos ter várias pastas, por exemplo, quando trabalhamos com vários sites, cada site estará em uma pasta.

Para ter acesso ao nosso servidor local, vamos iniciar o Painel de Controle do Xampp (`c:\xampp\xampp-control.exe`). A figura C.1 mostra o Painel de Controle. Inicialmente o serviço do Apache não está rodando. Para inicializá-lo clique no botão *Start* ao lado de Apache.

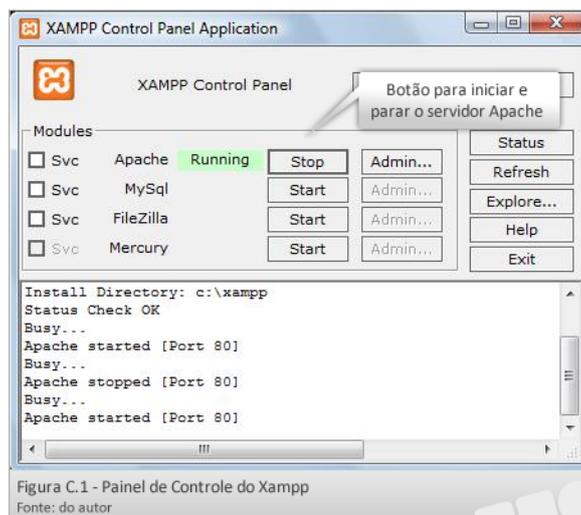


Figura C.1 - Painel de Controle do Xampp
Fonte: do autor

Agora vamos testar se o servidor está rodando. Abra o navegador e digite <http://localhost> na barra de endereço. Se você está visualizando uma página como a da figura C.2, quer dizer que o servidor está rodando. Você também pode acessar o servidor usando o IP local: <http://127.0.0.1>



Figura C.2 - Navegador acessando o servidor local
Fonte: do autor

Estrutura da linguagem

Para começar, a programação de páginas PHP pode ser feita em qualquer editor, por exemplo, o bloco de notas. Mas lembre-se de que agora a extensão dos seus arquivos vão ser *.php*.

Juntamente com uma página (X)HTML vamos escrever código na linguagem PHP. Para isso, precisamos indicar que estamos escrevendo PHP. Isto é feito usando a seguinte sintaxe:

```
<?php
```

Comandos

```
?>
```

Toda codificação PHP fica entre o abre “<?php” e fecha “?>”.

O que precisamos saber antes de escrever nosso primeiro arquivo PHP:

- Para strings pode-se usar tanto aspas simples como duplas.
- O ponto e vírgula (;) deve ser usado como fim de linha de comando.
- Para mostrar algo no navegador pode-se usar **echo** ou **print**. Abaixo são apresentadas formas válidas de usar o *echo* e o *print*:

```
echo("Bem-vindo ao PHP! ");  
  
echo "Bem-vindo ao PHP! ";  
  
echo 'Bem-vindo ao PHP! ';  
  
print'Bem-vindo ao PHP! ';  
  
print('Bem-vindo ao PHP!');
```

- Comentário de linha:

```
// comentário  
  
# comentário
```

- Comentário para mais de uma linha:

```
/* comentário */
```

Vamos criar nossa primeira página PHP. O código é mostrado na figura C.3. Salve este arquivo com o nome de *curso1.php* em *c:\xampp\htdocs\curso*.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
2 <html xmlns="http://www.w3.org/1999/xhtml">  
3 <head>  
4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />  
5 <title>Primeiro código PHP</title>  
6 </head>  
7 <body>  
8 <?php  
9  
10     echo "Olá Mundo";  
11 ?>  
12 </body>  
13 </html>
```

Figura C.3 - Primeira página PHP
Fonte: do autor

Para visualizar nossa página no navegador, acesse <http://localhost/curso/curso1.php>. A figura C.4 apresenta o resultado da página *curso1.php* no navegador.

Atenção:

Lembre-se de que o servidor web Apache deve estar rodando. No Painel de Controle do Xampp inicie o Apache.

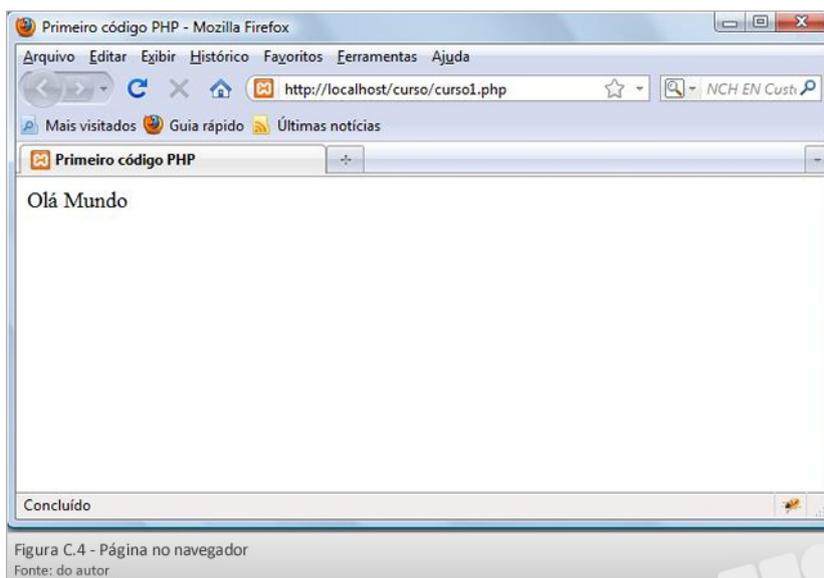


Figura C.4 - Página no navegador
Fonte: do autor

Agora vamos visualizar o código fonte da página. Utilize a tecla de atalho *ctrl + u* para abrir a janela com o código conforme a figura C.5. Observe que o código fonte não mostra nenhuma parte do código PHP. O que aconteceu? O PHP no servidor interpretou a parte de código PHP e retornou apenas o resultado, que neste caso era o texto "Olá Mundo!". É muito importante compreender como funcionam as linguagens do lado servidor. Tudo é processado no servidor; o cliente recebe apenas o resultado.

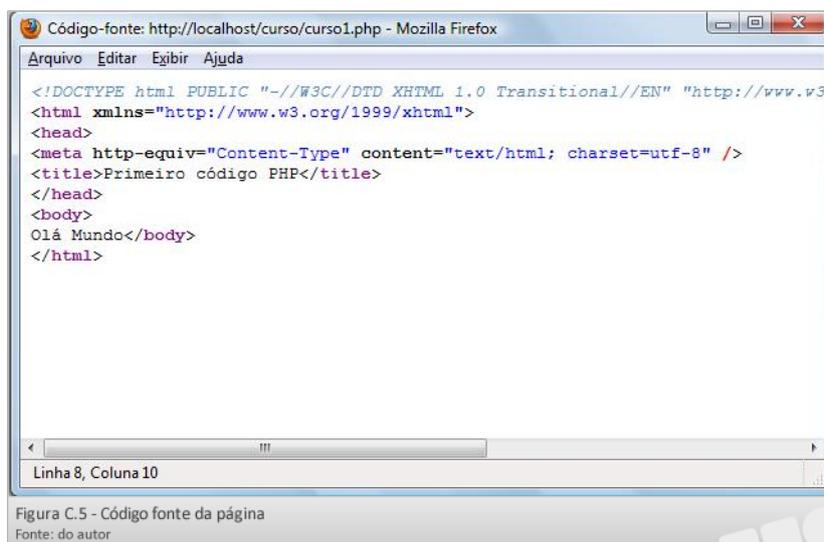


Figura C.5 - Código fonte da página
Fonte: do autor

Variáveis

Assim como na linguagem JavaScript, também no PHP não é necessário fazer uma declaração explícita de variável. Basta atribuir diretamente um valor à variável para ela ser criada e assumir o tipo de acordo com o valor recebido. As variáveis em PHP devem iniciar com o caractere \$. Após este caractere, vem o identificador da variável que não pode iniciar com um número (após a primeira letra pode-se usar números). Veja na tabela abaixo alguns exemplos de variáveis válidas e inválidas.

Variáveis válidas	Variáveis inválidas
\$valor	\$9teste
\$num1	\$800

\$nome_pessoa	\$1_valor
\$time_2	\$6time

Dica:

O PHP é case sensitive, a variável de nome **\$teste** é diferente da variável de nome **\$Teste**.

Atenção:

O PHP possui quatro tipos básicos de dados: **integer**, **float** (número de ponto flutuante, ou também **double**), **string** e **boolean**. O tipo é assumido de acordo com o valor atribuído.

Exemplos de atribuição de valores para variáveis:

```
<?php
    $ano    = 2000;
    $nome   = "Maria";
    $idade  = 10;
    $mes    = "Outubro";
?>
```

Constantes

A definição de constantes em PHP usa a seguinte sintaxe:

```
define("nome_constante", "valor_constante");
```

Exemplo de definição de constantes

```
define("PI", 3.14159265);
define("peso", 80);
```

Exemplo de uso de constante

```
echo "O valor de PI é ". PI;
```

Dica:

nome que para constante não usamos o \$. O ponto (.) no exemplo acima foi usado para concatenar a string com o valor da constante.

Atenção:

Lembre-se de que constantes não podem ter seu valor alterado ao longo do programa.

Operadores Lógicos

Operadores lógicos são normalmente utilizados em comandos condicionais, como *if*, *for* e *while*

Operador lógico	Finalidade
==	Igual
!	Negação (se <i>true</i> passa para <i>false</i> , se <i>false</i> passa para <i>true</i>)
!= ou <>	Diferente
>	Maior
<	Menor
>=	Maior ou igual
<=	Menor ou igual
&& ou and	E
ou or	Ou

Operadores Matemáticos

Operador Matemático	Finalidade
+	Adição de valores
-	Subtração de valores
*	Multiplicação de valores
/	Divisão de valores
%	Retorna o resto de uma divisão. Exemplo: $150 \% 13$ retornará 7 $7 \% 3$ retornará 1

Expressões Simples com operadores

Operador	Finalidade
=	Atribuição
+=	Adiciona ao string/valor já existente. Exemplo: $\$x += \y é o mesmo que $\$x = \$x + \$y$, da mesma forma podem ser utilizados: -= , *= , /= ou %=
++	Acrescenta 1 no valor Exemplo: $\$x = 3;$ $\$x++; // \x valerá 4

--	Decrementa 1 no valor Exemplo: <pre>\$X = 3; \$X--; // \$X valerá 2</pre>
.	Concatenação Exemplo: <pre>\$X = "linguagem "; \$X .= "PHP "; echo \$X; // linguagem PHP \$X = "linguagem "; \$Y = "PHP"; \$Z = \$X.\$Y; echo \$Z; // linguagem PHP</pre>

Estruturas de Controle

A seguir, veremos as principais instruções condicionais e de repetição. Observaremos que seguem uma sintaxe muito parecida de outras linguagens.

if

A instrução **if** é utilizada quando é necessário tomar decisões. Sua estrutura é:

```
if (<expressão>
    <Instrução>
```

A expressão é avaliada, caso seja *true* (verdadeira) a instrução é executada; caso contrário, a instrução não é executada.

Se o bloco de instrução possuir mais de uma instrução, deve-se usar início de bloco e fim de bloco, ou seja {}.

```
if (<expressão>){
    <Instrução>
}
```

Caso exista instrução a ser executada quando a expressão retornar *false* (falso), pode ser usar o **else**.

```
if (<expressão>){
    <instrução>
}
else{
    <instrução>
}
```

switch

O *switch* também pode ser utilizado para testes condicionais e é indicado para situações que exijam vários testes.

Sintaxe:

```
switch (variavel_de_controle) {  
    case opção1 :  
        comandos ;  
        break;  
    case opção2 :  
        comandos ;  
        break;  
    default :  
        comandos;  
}
```

É importante observar que a instrução *break* é utilizada ao final de cada opção. O *break* faz com que a execução pule para o fim do *switch*. Isso é importante, uma vez que quando uma opção verdadeira for encontrada, o *switch* segue executando todas as instruções até o fim, se não possuir um *break*.

for

Para situações em que precisamos realizar um bloco de instrução diversas vezes, uma opção de solução é o comando de repetição *for*.

Sintaxe:

```
for (<inicialização>;<condição>;<atualização variável>){  
  
    <instruções>  
}
```

while

O *while* é semelhante ao comando *for*, porém, às vezes, o *while* é aplicado quando não sabemos determinar a quantidade de vezes que nosso laço vai executar as instruções. Nesse laço, enquanto a condição do *while* for verdadeira as instruções serão executadas.

Sintaxe:

```
while(<condição>){  
  
    <instruções>  
}
```

Dica:

Cuidado para que seu laço não seja infinito. A condição precisa ser falsa em algum momento para sair do laço ou pode usar o comando *exit* para sair do laço.

do .. while

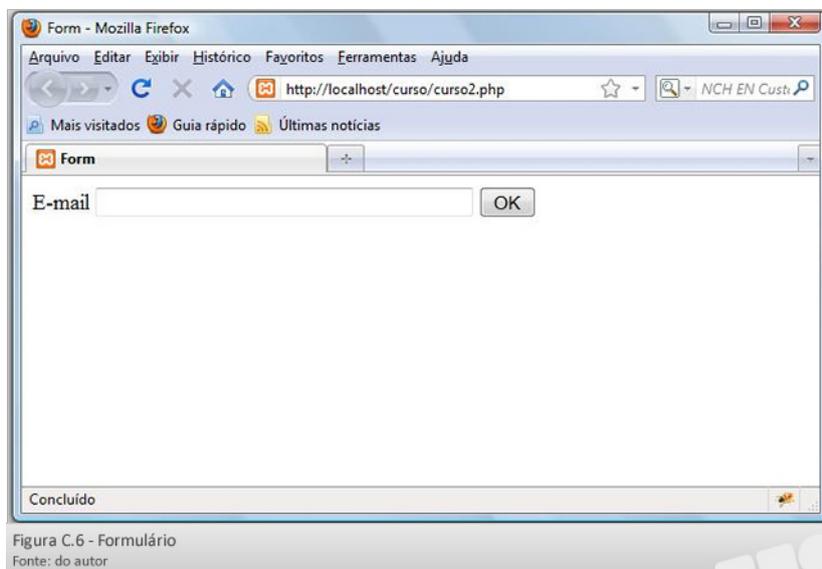
Outra opção é o *do .. while* que é semelhante ao *while*, porém, o teste condicional é feito no fim do laço. Nesse tipo de laço, os comandos são executados pelo menos uma vez.

Sintaxe:

```
do {
    comandos;
}while (<condição>)
```

Processamento de Formulários

Para tornar nossas páginas mais interativas, precisamos manipular dados vindos de formulários. Considere o formulário da figura C.6. Como podemos via PHP recuperar o e-mail informado neste formulário? É o que vamos ver agora.



A figura C.7 apresenta o código (X)HTML correspondente ao formulário acima. Algumas dessas informações importantes estão destacada em vermelho:

- método do formulário: *POST*
- ação do formulário: *tratar_dados.php*
- nome da caixa de texto do e-mail: *email*
- nome do botão: *botao*

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 <html xmlns="http://www.w3.org/1999/xhtml">
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5 <title>Form</title>
6 </head>
7 <body>
8 <form id="form1" name="form1" method="post" action="tratar_dados.php">
9 <label for="email">E-mail</label>
10 <input name="email" type="text" id="email" size="40" />
11 <input type="submit" name="botao" id="botao" value="OK" />
12 </form>
13 </body>
14 </html>

```

Figura C.7 - Código (X)HTML do formulário
Fonte: do autor

É importante observar que ao clicar no botão “OK” o navegador fará a requisição do arquivo *tratar_dados.php* passando os dados do formulário usando o método POST. A questão agora é como pegar os dados no *tratar_dados.php*?

Para conseguir pegar os valores precisamos conhecer os arrays superglobais do PHP: **\$_POST**, **\$_GET** e **\$_REQUEST**.

\$_POST: usado quando os dados são enviados pelo método POST.

\$_GET: usado quando os dados são enviados pelo método GET.

\$_REQUEST: pode ser usado tanto para dados vindos pelo método GET quanto pelo método POST.

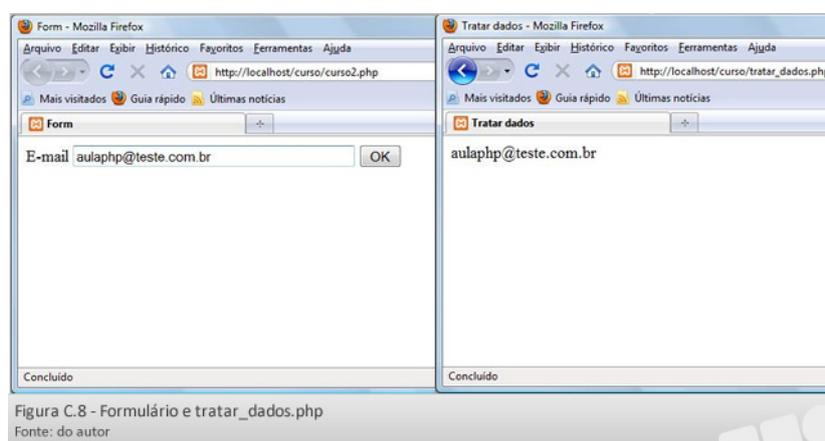
No nosso exemplo, os dados estão sendo enviados usando o método POST e o valor que queremos pegar é da caixa de texto cujo *name* é *email*. Então vamos pegar o valor no *tratar_dados.php*:

```

<?php
    $email_recebido = $_POST['email'];
    echo $email_recebido;
?>

```

O código PHP acima mostra o e-mail informado pelo usuário no formulário. Observe que 'email' foi usado um índice para o array **\$_POST**. Veja a figura C.8 que mostra os dois arquivos.



Outra forma interessante seria verificar se o *tratar_dados.php* foi requisitado pelo formulário. Veja abaixo:

```

<?php
    if ($_POST['botao'] == "OK") {
        $email_recebido = $_POST['email'];
    }

```

```

        echo $email_recebido;
    }
?>

```

Neste exemplo estamos testando se veio um valor cujo índice é botao, que é o nome do botão no formulário. Se ele possuir valor igual a "OK" então significa que veio do form e mostra o valor de email.

Dica:

- Em PHP podemos usar uma variável dentro de aspas duplas. No exemplo abaixo será mostrado o valor correspondente da variável (40).

Ex.:

```

<?php

    $idade = 40;
    echo "A idade de João é $idade";

?>

```

- O mesmo não é válido para aspas simples. No exemplo abaixo será mostrado literalmente \$idade.

Ex.:

```

<?php

    $idade = 40;
    echo 'A idade de João é $idade';

?>

```

Faça estes dois exemplos apresentados acima para testar.

- O caractere @ é usado para como um operador de controle de erro:

Ex.:

```

<?php

    $a = @$b / $c; //Suprime mensagem de erro

?>

```

- Teste o código PHP abaixo e veja o que acontece.

```

<?php

    echo ("Olá mundo!");

    echo ("nossa primeira página PHP!");

?>

```

No navegador vai mostrar tudo em uma mesma linha. Mas como fazer para mostrar cada mensagem em uma linha. Lembre-se que está visualizando no navegador, então basta usar o elemento HTML
:

```

<?php

```

```

    echo ("Olá mundo!");
    echo ("br />nossa primeira página PHP!");
?>
    
```

Síntese

Bem, agora já conhecemos a estrutura básica da linguagem e construímos algumas páginas PHP para testes. De agora em diante, podemos avançar e conhecer outros recursos da linguagem que permitirão criar páginas dinâmicas e mais interessantes.

Atividades - Parte 1

1. Faça uma pesquisa sobre as diferentes versões do PHP, apresentando suas principais características e quando (ano) elas foram disponibilizadas.
2. Crie um formulário (X)HTML como o da figura abaixo. O arquivo deve ser nomeado de *form1.html* e na *action* do formulário deve chamar *dados.php*. O *dados.php* deve mostrar todos os dados do *form1.html*.

Nome:

E-mail:

Sexo: Feminino Masculino

Estado Civil:

- Solteiro
- Casado
- União Estável
- Viúvo
- Divorciado

Figura C.9 - Exercício 2
Fonte: do autor

3. Dado o formulário abaixo do arquivo *form3.html*, crie o *calcular.php* que faz o cálculo com os dois valores informados de acordo com a operação selecionada.

Valor 1: Valor 2:

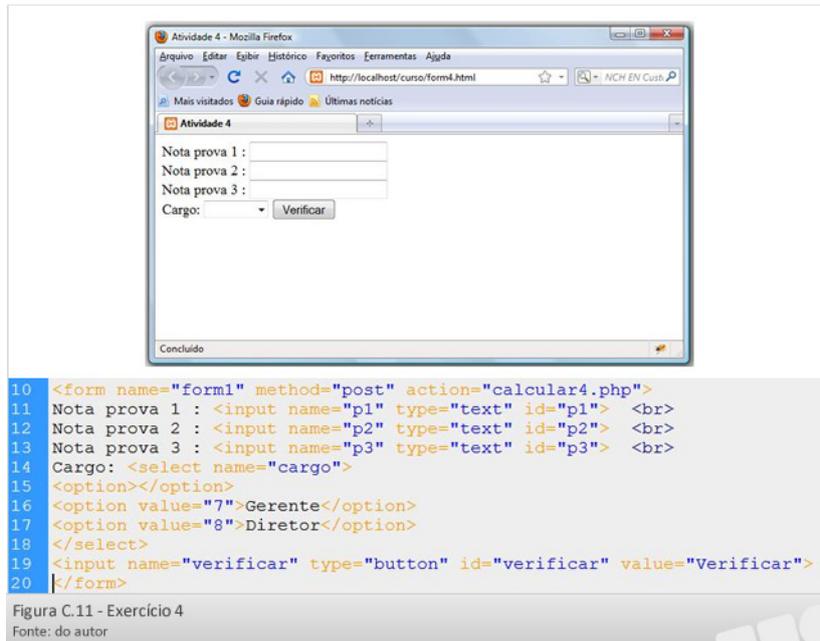
Operação: Adição Subtração Multiplicação Divisão

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml">
4 <head>
5 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
6 <title>Form Atividade 3</title>
7 </head>
8 <body>
9 <form id="form1" name="form1" method="post" action="calcular.php">
10 <label for="v1">Valor 1:</label>
11 <input name="v1" type="text" id="v1" size="10" />
12 <label for="v2">Valor 2:</label>
13 <input name="v2" type="text" id="v2" size="10" />
14 Operação:
15 <input type="radio" name="op" value="A" id="sexo_0" /> Adição
16 <input type="radio" name="op" value="S" id="sexo_1" /> Subtração
17 <input type="radio" name="op" value="M" id="sexo_2" /> Multiplicação
18 <input type="radio" name="op" value="D" id="sexo_3" /> Divisão<br />
19 <input type="submit" name="botao" id="botao" value="Calcular" /> <br />
20 </form>
21 </body>
22 </html>
    
```

Figura C.10 - Exercício 3
Fonte: do autor

4. Considere o formulário (e código (X)HTML correspondente) abaixo para solucionar o problema a seguir.



- a. No processo de seleção para cargos de uma empresa, o candidato faz 3 provas, cujos pesos são:
 - Prova 1 = peso 3
 - Prova 2 = peso 3
 - Prova 3 = peso 4
- b. Para ser considerado aprovado, o candidato deve ter média acima de 7 para o cargo de gerente e acima de 8 para o cargo de diretor.
- c. Crie o arquivo *calcular4.php* que faz o cálculo da nota final do candidato e mostra a situação (APROVADO/REPROVADO).

Linguagem PHP – Parte 2

Continuando a unidade C - Introdução à linguagem PHP, nesta parte 2 do material abordaremos a manipulação de arrays, strings e datas. Ao final desta parte, serão apresentadas a atividades para realização.

Arrays

Array, ou vetor, são estruturas que podem armazenar vários valores, sendo a referência a cada valor através de um índice. O que muda no PHP é que este índice pode ser tanto um número quanto uma string. Vejamos alguns exemplos de criação de arrays em PHP:

- Definição de um array sem valores:

```
$vet = array ();
```

- Criando de array inicializado:

```
$notas = array (5,5,3,6,8);
```

```
$nomes = array ("José", "Maria", "João", "Márcia", "Paulo");
```

Dica:

Nos exemplos acima, para acessar cada elemento de um vetor, devemos nos referir ao seu índice. Em PHP o primeiro índice de um vetor é o 0 (Zero). Sendo que o último índice é sempre igual ao número de elementos de um vetor - 1. Assim, um vetor de 5 elementos tem índices numéricos inteiros que vão de 0 a 4.

- No exemplo abaixo, quando não é especificado o índice, o valor é atribuído para o próximo índice livre, iniciando em zero:

```
$mes[] = "Janeiro"; // mesmo que $mes[0] = "Janeiro";

$mes[] = "Fevereiro"; // mesmo que $mes[1] = "Fevereiro";
```

Array Associativo

Em um vetor associativo, os índices são strings. Assim, não temos índices numéricos, mas conjuntos de caracteres que representam cada elemento do vetor. Veja abaixo duas formas diferentes de criar um array associativo:

```
$dados = array ("nome" => "Camila",
               "nascimento" => "25/11/1981",
               "cargo" => "professor",
               "rg" => "9990909090");
```

Ou

```
$dados["nome"]           = "Camila";
$dados["nascimento"]     = "25/11/1981";
$dados["cargo"]          = "professor";
$dados["rg"]             = "9990909090";
```

Laço para mostrar valores de array

Vamos utilizar um tipo especial de laço para mostrar os dados de arrays, é o `foreach`. O `foreach` fornece uma forma bastante prática de percorrermos um vetor associativo e tem duas variações, vejamos a sua estrutura:

- No `foreach` do exemplo abaixo o laço percorre todos os elementos e a cada iteração atribui o valor do elemento para a variável `$valor`:

```
$nomearray = array(1,2,3,4);
foreach($nomearray as $valor){
    echo "$valor <br />";
}
```

ou

- Já no exemplo abaixo a cada iteração é atribuído o índice do elemento para a variável `$indice` e o valor para a variável `$valor`:

```

$nomearray = array(1,2,3,4);
foreach($nomearray as $indice => $valor){
    echo "$indice - $valor <br />";
}

```

Algumas funções para arrays

A tabela abaixo apresenta funções para ordenação de arrays.

sort	ordena de forma crescente (do menor para o maior) <pre> \$frutas = array ("maça","uva","abacaxi","banana"); sort(\$frutas); //abacaxi, banana, maça, uva foreach(\$frutas as \$valor) echo "\$valor, "; //saída: abacaxi, banana, maça, uva, </pre>
rsort	ordena de forma decrescente (do maior para o menor) <pre> \$frutas = array ("maça","uva","abacaxi","banana"); rsort(\$frutas); foreach(\$frutas as \$valor) echo "\$valor, "; //saída: uva, maça, banana, abacaxi </pre>
asort	ordena um array associativo mantendo a associação entre índices e valores <pre> \$vet = array ("nome" => "Ana", "idade" => "23", "cargo" => "gerente"); asort(\$vet); foreach(\$vet as \$valor) echo "\$valor, "; //saída: 23, Ana, gerente, </pre>
arsort	ordena um array associativo em ordem decrescente mantendo a associação entre índices e valores <pre> \$vet = array ("nome" => "Ana", "idade" => "23", "cargo" => "gerente"); arsort(\$vet); foreach(\$vet as \$valor) echo "\$valor, "; //saída: gerente, Ana, 23, </pre>
ksort	ordena um array por seus índices <pre> \$vet = array ("nome" => "Ana", "idade" => "23", "cargo" => "gerente"); ksort(\$vet); foreach(\$vet as \$ind => \$valor) echo "\$ind: \$valor, "; //saída: cargo: gerente, idade: 23, nome: Ana, </pre>

Outras duas funções interessantes para tratamento de arrays são apresentadas na tabela a seguir: *implode* e *explode*.

implode	<p>Retorna uma string contendo todos os elementos do array, separados pela string passada por parâmetro.</p> <pre>\$partes = array("a", "casa número", 13, "é azul"); \$frase = implode(" ", \$partes); /* \$frase passa a conter a string: "a casa número 13 é azul" */</pre>
explode	<p>Retorna um array contendo partes da string fornecida, separadas pelo delimitador fornecido.</p> <pre>\$data = "11/14/1975"; \$vdtd = explode("/", \$data); /* O array \$vdtd vai conter os seguintes valores: \$vdtd[0] = 11 \$vdtd[1] = 14 \$vdtd[2] = 1975 */</pre>

Strings

String é uma sequência de letras, dígitos, caracteres de pontuação e outros, que são representados pela linguagem como texto. Strings literais podem ser usadas delimitando por pares de aspas simples ('..') ou aspas duplas ("... "). A seguir, veremos as principais funções do PHP associadas a strings.

Função	Descrição
strlen()	<p>Retorna o número de caracteres de uma string</p> <pre>\$str = "programação"; strlen(\$str); // retorna 11</pre>
substr	<p>Retorna uma parte de uma string. Sintaxe:</p> <pre>substr(string, posição_inicial, tamanho);</pre> <p>Obs.: lembre-se de que a string começa na posição zero</p> <pre>\$str = "lpw@teste.com.br"; substr(\$str, 3, 6); // Retorna "@teste"</pre>
ucfirst	<p>Converte para maiúsculo o primeiro caractere de uma string.</p> <pre>\$str = "programação"; ucfirst(\$str); // Retorna "Programação"</pre>
strtoupper	<p>Converte uma string para maiúsculas.</p> <pre>\$str = "programação"; strtoupper(\$str); // Retorna "PROGRAMAÇÃO"</pre>

strtolower	<p>Converte uma string para minúsculas.</p> <pre>\$str = "Programação"; strtolower(\$str); // Retorna "programação"</pre>
str_replace()	<p>Substituição de caracteres em uma string. Sintaxe: str_replace (string_pesquisar, nova_string, onde_pesquisar)</p> <pre>\$texto = "10/11/2011"; str_replace("/", "-", \$texto); // retorna "10-11-2011";</pre>
strip_tags()	<p>Retorna uma string, retirando as tags HTML e/ou PHP.</p> <pre>strip_tags('testando
'); //Retorna a string "testando"</pre>
htmlspecialchars()	<p>Converte caracteres especiais HTML para string de forma que não sejam interpretados pelo navegador.</p> <pre>\$texto = "<p>String</p>"; htmlspecialchars(\$texto); // Retorna "<p>String<p>", e não "String" em um parágrafo</pre>
urlencode()	<p>Retorna a string, convertida para o formato <i>urlencode</i>. Esta função é útil para passar valores para uma próxima página através do método GET.</p> <pre>\$frase = "Título da notícia"; urlencode(\$frase); // Retorna "Título+da+notícia";</pre>
nl2br()	<p>Converte a quebra de linha (\n) por quebra de linha em HTML (
). Obs.: esta função é interessante de ser usada quando há necessidade de mostrar um texto digitado pelo usuário, por exemplo, em um <i>textarea</i></p> <pre>\$nome = "Linguagem \nde programação para\n Web"; nl2br(\$nome); // Retorna "Linguagem
de programação para
 Web"</pre>
strrev()	<p>Retorna a string invertida</p> <pre>echo strrev("Função"); // Retorna "oãçnuF"</pre>
trim()	<p>Retira espaços e linhas em branco do início e do final da string fornecida.</p> <pre>echo trim(" teste \n \n "); // Retorna "teste"</pre>

Datas

Trabalhar com datas e horas também é muitas vezes necessário para incorporar algum recurso nos sites e tratar dados de formulário ou banco de dados. Por exemplo, recuperar a data atual, formatar datas, pegar o dia da semana. Por isso, vamos conhecer as principais funções PHP para manipular datas.

Função	Descrição
mktime()	Retorna o timestamp Unix correspondente para os argumentos dados. Este timestamp é um longo inteiro contendo o número de segundos entre a Era Unix (January 1 1970 00:00:00 GMT) e o tempo especificado. Argumentos podem ser omitidos da direita para esquerda; quaisquer argumentos assim omitidos serão definidos para o valor atual de acordo com a data e a hora local. Sintaxe: <pre>int mktime ([int \$hora [, int \$minuto [, int \$second [, int \$mes [, int \$dia [, int \$ano [, int \$is_dst]]]]]])</pre>
time()	Retorna o timestamp Unix atual. Sintaxe: <pre>time()</pre>
date()	Formata data e hora de acordo com a string format dada. Sintaxe: <pre>date("opções formatação", data)</pre>
checkdate()	Valida uma data. Sintaxe: <pre>checkdate (mes, dia, ano)</pre>

Vejamos na figura C.12 o uso das funções mktime() e time().

Na linha 11: foi criada uma data passando por parâmetro hora, minutos, segundos, mês, dia e ano.

Na linha 15: foi criada a variável \$hoje com a data e hora atual.

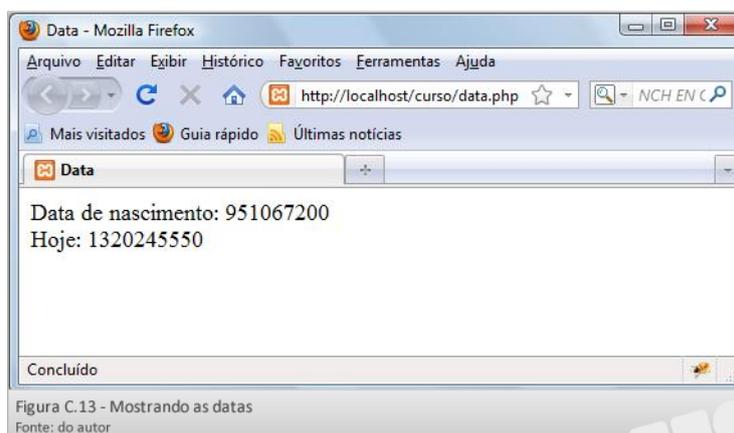
As linhas 13 e 17: mostram as variáveis criadas. Faça o teste no seu navegador! Você verá que é mostrado um valor para as variáveis e não uma data/hora, como na figura C.13.

```

9  <?php
10
11  $data_nascimento = mktime(15,20,0,2,20,2000);
12
13  echo "Data de nascimento: $data_nascimento <br/> ";
14
15  $hoje = time();
16
17  echo "Hoje: $hoje";
18  ?>

```

Figura C.12 - Funções mktime() e time()
Fonte: do autor



Como podemos ver, a data e hora mostradas não estão em um formato conhecido por nós. Precisaremos utilizar o função `date()` para formatar data e hora. A tabela abaixo apresenta as opções de formatação para a função `date()`.

Opção	Descrição
a	Antes/Depois de meio-dia em minúsculo (am ou pm)
A	Antes/Depois de meio-dia em maiúsculo (AM ou PM)
d	Dia do mês, 2 dígitos com preenchimento de zero (01 até 31)
D	Uma representação textual de um dia, três letras (Mon até Sun)
w	Representação numérica do dia da semana 0 (para domingo) até 6 (para sábado)
m	Representação numérica de um mês (01 a 12)
M	Uma representação textual curta de um mês, três letras (Jan a Dec)
t	Número de dias de um dado mês (28 até 31)
L	Se um ano é bissexto (1 se está em ano bissexto, 0 caso contrário)
Y	Uma representação de ano completa com quatro dígitos
y	Uma representação do ano com dois dígitos
g	Formato 12-horas de uma hora sem preenchimento de zero (1 até 12)
G	Formato 24-horas de uma hora sem preenchimento de zero (0 até 23)
h	Formato 12-horas de uma hora com zero preenchendo à esquerda (01 até 12)
H	Formato 24-horas de uma hora com zero preenchendo à esquerda (00 até 23)
i	Minutos com zero preenchendo à esquerda (00 até 59)
s	Segundos, com zero preenchendo à esquerda (00 até 59)

Vamos ver como fica nosso código PHP utilizando agora a função `date()`. Veja na figura C.14.

Nas linhas 12 e 16: foram criadas novas variáveis com as datas formatadas. A figura C.15 mostra o resultado do código no navegador.

```

9  <?php
10
11     $data_nascimento = mktime(15,20,0,2,20,2000);
12     $fdata_nascimento = date("d/m/Y - H:i:s",$data_nascimento);
13     echo "Data de nascimento: $fdata_nascimento <br/> ";
14
15     $hoje = time();
16     $fhoje = date("d/m/Y - H:i:s",$hoje);
17     echo "Hoje: $fhoje";
18  ?>

```

Figura C.14 - Data com formatação
Fonte: do autor

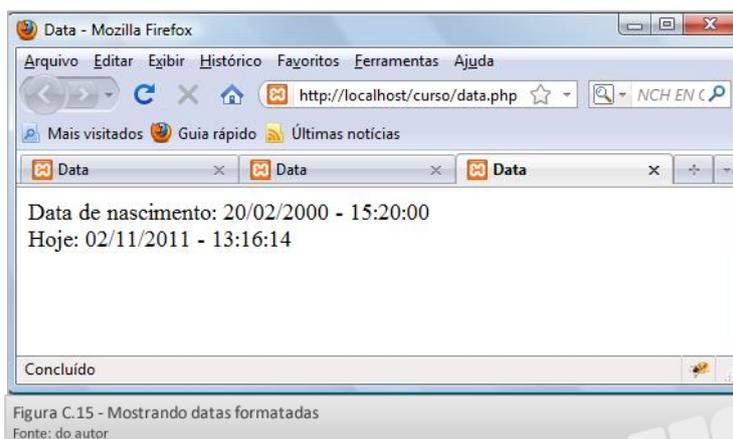


Figura C.15 - Mostrando datas formatadas
Fonte: do autor

Atenção:

A “data e hora” retornadas no exemplo acima são do servidor onde a página está hospedada. No nosso caso, como estamos trabalhando com o servidor local, será a data/hora da máquina.

Saiba Mais:

você pode ver mais opções de formatação acessando:

http://br2.php.net/manual/pt_BR/function.date.php

Outra função interessante para tratamento de datas é a *checkdate*, útil para validar uma data. Imagine que o usuário informa uma data em um formulário, antes de trabalhar com esta data é importante validá-la. Veja o código da figura C.16. No mesmo arquivo, foi colocado o *form* e o código que verifica a data (quando o *form* é submetido chama o próprio arquivo, pois não tem nada em *action*). Algumas observações sobre o código:

Na linha 14: é verificado se o form foi submetido. Neste caso, faz a verificação da data;

Como a função *checkdate* espera três parâmetros, foi necessário quebrar a data em dia, mês e ano (linha 18);

Na linha 24: é usada a função *checkdate*, passando os valores da data que estão no vetor *\$vdt*.

```

8 <form id="form1" name="form1" method="post" action="">
9   <label for="data">Data:</label>
10  <input type="text" name="data" id="data" />
11  <input type="submit" name="botao" id="botao" value="OK" />
12 </form>
13 <?php
14   if ($_POST['botao'] == 'OK'){
15
16     $data = $_POST['data'];
17     echo "Data informada: $data <br/>";
18     $vdt = explode('/', $data);
19     /*
20      $vdt[0] contém o dia
21      $vdt[1] contém o mês
22      $vdt[2] contém o ano
23     */
24     if (checkdate($vdt[1], $vdt[0], $vdt[2])) // verifica se data é válida
25       echo "Data válida";
26     else
27       echo "Data inválida";
28   }
29  ?>

```

Figura C.16 - Código PHP para validação de data
Fonte: do autor

A figura C.17 mostra o resultado do código acima no navegador, sendo uma execução com data válida e outro com data inválida.

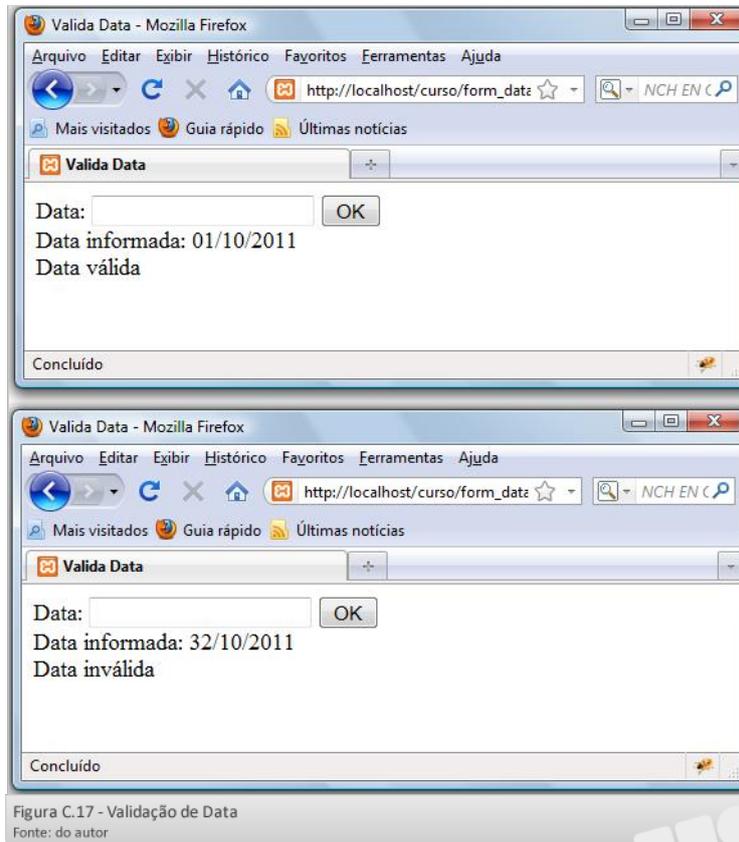


Figura C.17 - Validação de Data
Fonte: do autor

Síntese

Conhecer as funções que foram apresentadas é muito importante para o desenvolvimento de aplicações com PHP, uma vez que é muito comum trabalhar com arrays, strings e datas. Na próxima parte da Unidade C vamos trabalhar com funções definidas pelo programador e voltaremos a utilizar as funções vistas nesta parte.

Atividades - Parte 2

1. Dado o formulário da figura abaixo, crie o arquivo `exe1.php` que:

- mostra o texto todo em maiúsculas;
- mostra o texto todo em minúsculas;
- mostra o texto com primeira letra em maiúscula;
- mostra o texto convertendo todos os caracteres de nova linha (`\n`) para a marcação `
`;
- mostra o texto retirando todas as marcações HTML (tags) que tenham sido inseridas;
- mostra o tamanho do texto (número de caracteres);
- mostra os 5 primeiros caracteres do texto.

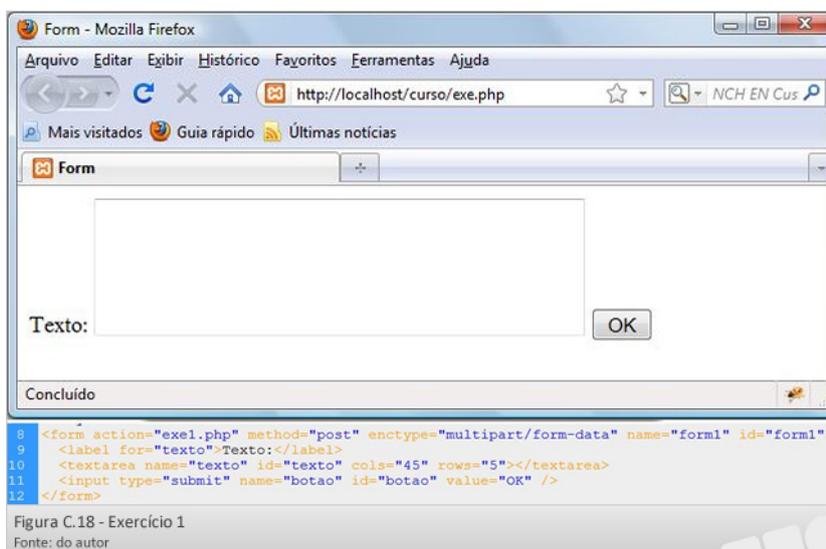


Figura C.18 - Exercício 1
Fonte: do autor

2. Crie um array associativo para armazenar as notas de alunos, onde o índice é o número de matrícula e a nota é o valor correspondente. Faça um código PHP para mostrar as notas de forma ordenada, sendo da maior para a menor nota, juntamente com o número da matrícula do aluno (conforme modelo abaixo).

Matrícula	Média
3532	8.9
2314	8.6
2342	8.5
9893	8.4

3. Dado o array abaixo, faça um código PHP que mostra o dia da semana para cada uma das datas:

```
$vetor = array("10/04/2011", "14/04/2011 ", "21/04/2011 ", "01/05/2011 ", "07/09/2011 ", "20/09/2011 ");
```

Por exemplo:

```
10/04/2011 - Domingo
14/04/2011 - Quinta-feira
```

4. Dado o array abaixo, faça um código PHP que conta e mostra quantas datas são do mês de novembro.

```
$feriados = array("07/09/2010", "20/09/2010", "12/10/2010", "02/11/2010", "15/11/2010", "08/12/2010");
```

Considere a entrada de uma informação de data de nascimento no formato dd/mm/aaaa. Faça um código PHP que mostra o signo da pessoa para aquela data. Faça um formulário para informar a data e testar o código. Para auxiliar na definição são fornecidos dois vetores:

- a) o vetor \$ultimo_dia contém o último dia do mês válido para o signo
- b) o vetor \$signos contém os signos

```
$ultimo_dia = array(20,19,20,20,20,20,21,22,22,22,21,21);
$signos = array('Capricórnio ', 'Aquário ', 'Peixes ', 'Áries ', 'Touro ',
```

```
'Gêmeos ', 'Câncer ', 'Leão ', 'Virgem ', 'Libra ', 'Escorpião ', Sagitário
');
```

Mês	Último dia	Signo
01	20	Capricórnio
02	19	Aquário
03	20	Peixes
04	20	Áries
05	20	Touro
06	20	Gêmeos
07	21	Câncer
08	22	Leão
09	22	Virgem
10	22	Libra
11	21	Escorpião
12	21	Sagitário

Exemplos:

Para a data 02/08/1975 -> observar o mês 8 -> o dia 02 é menor ou igual ao último dia, então é leão

Para a data 30/11/1991 -> observar o mês 11 -> o dia 30 é maior que o último dia, então pega o mês seguinte -> neste caso o signo é Sagitário

5. Análise os trechos de código PHP abaixo e marque o bloco que possui erros assinalando-os:

<p>a)</p> <pre>\$doc = array(); \$doc['rg'] = "00.000.000-X"; \$doc['cpf'] = "000.000.000-00"; \$doc['cartao'] = 12345; asort(\$doc); foreach(\$doc as \$i => \$dado) echo "
 \$i = \$dado ";</pre>	<p>b)</p> <pre>\$doc = array("rg" => "00.000.000-X", "cpf" => "000.000.000-00", "cartao" => 12345); arsort(\$doc); foreach(\$doc as \$i => \$dado) echo "
 \$i = \$dado ";</pre>
<p>c)</p> <pre>\$doc = array(); \$doc['rg'] = "00.000.000-X"; \$doc['cpf'] = "000.000.000-00"; \$doc['cartao'] = 12345; echo "
 RG = \$doc[rg] "; echo "
 CPF = \$doc[cpf] "; echo "
 Cartão = \$doc[cartao] ";</pre>	<p>d)</p> <pre>\$doc[rg] = "00.000.000-X"; \$doc[cpf] = "000.000.000-00"; \$doc[cartao] = 12345; sort(\$doc); foreach(\$dado as \$doc) echo "
 \$dado ";</pre>

Linguagem PHP – Parte 3

Nesta parte 3 da Unidade C vamos trabalhar funções. O uso de funções deixa o código mais organizado e modular, possibilitando a reutilização de código toda vez que precisamos realizar uma mesma tarefa. Ao longo deste material vamos apresentar situações problema em que se torne interessante o uso de funções. Ao final desta parte serão apresentadas as atividades para realização.

Funções

Uma função é um conjunto de instruções destinado a uma tarefa bem específica e que podemos utilizar várias vezes. Por exemplo, podemos precisar de funções para verificar se um CPF é válido, tratar valores recebidos de formulários, como datas e strings. A sintaxe para a criação de uma função em PHP é:

```
function nome_da_funcao(parâmetros) {

    // instruções

}
```

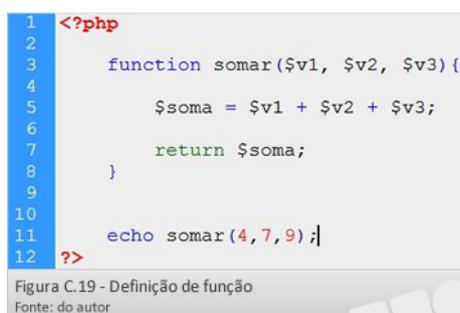
Atenção:

Lembre-se de que PHP é case sensitive para variáveis/funções definidas e que o nome de uma função deve ser único, ou seja, não podemos ter duas funções com o mesmo nome. Além disso, o nome da função não pode iniciar por número e não se podem usar caracteres como ponto, vírgula, espaço.

Os parâmetros da função são opcionais, mas os parênteses devem sempre aparecer. Uma função só é executada quando chamamos a função, ou seja, apenas a sua definição não executa as instruções pertencentes a ela. A chamada ou invocação de uma função se faz pelo nome da função com parênteses, por exemplo:

```
nome_da_funcao();
```

Uma função pode retornar um valor, neste caso, usa-se a palavra reservada **return**. Vamos a um exemplo. A figura C.19 apresenta a definição de uma função chamada somar que recebe três valores por parâmetro. A função soma os três valores e retorna o valor. A definição da função começa na linha 3 e termina na linha 8. A seguir, na linha 11 é feita a chamada da função, passando os três valores para cálculo. Como a função retorna o valor total, será mostrado no navegador o resultado da soma.



```
1 <?php
2
3     function somar($v1, $v2, $v3){
4
5         $soma = $v1 + $v2 + $v3;
6
7         return $soma;
8     }
9
10
11     echo somar(4,7,9);|
12 ?>
```

Figura C.19 - Definição de função
Fonte: do autor

Juntamente com funções, é importante compreender o escopo de variáveis em PHP:

- Para se utilizar uma variável que seja vista tanto dentro como fora da função deve-se declará-la como sendo GLOBAL, ou seja, fora da função.
- Uma variável declarada dentro de uma função será válida apenas dentro da própria função. Por exemplo, o código abaixo, não imprimirá nada em virtude de `$texto` possuir escopo local (função).

```

4     function mostrar(){
5
6         $texto = "teste de variável local";
7
8     }
9
10    echo $texto;

```

Figura C.20 - Exemplo de variável local
Fonte: do autor

A seguir vamos ver exemplos de várias funções.

Exemplo de função que recebe parâmetro e não retorna valor

A função apresentada na figura C.21, chamada `tabuada`, recebe um número por parâmetro e mostra a tabuada do número recebido. Veja que a função não retorna valor, já mostra diretamente dentro da função os valores. Na função também foi tratado o valor recebido por parâmetro. A função `is_numeric` retorna `true` se o valor passado por parâmetro é numérico e `false` em caso negativo. Se o valor for numérico é feito um teste se ele é maior ou igual a 1 e menor igual a 9, neste caso é feito um laço para mostrar a tabuada. Se o valor não estiver no intervalo então mostra uma mensagem “Informe um número entre 1 e 9”.

```

1  <?php
2
3  //exemplo de função que recebe parâmetro e não retorna valor
4
5  function tabuada($num){
6
7      if (!is_numeric($num))
8          echo "Informe um número";
9      else
10         if ((int)$num >= 1 && (int)$num <=9){
11             echo "Tabuada do $num <br />";
12             for ($i=1;$i<=10;$i++){
13
14                 $x = $num * $i;
15                 echo "$num * $i = $x <br />";
16             }
17         }else
18             echo "Informe um número entre 1 e 9";
19     }
20 }
21
22 tabuada(9);

```

Figura C.21 - Definição da função `tabuada`
Fonte: do autor

A figura C.22 mostra a chamada da função no navegador.

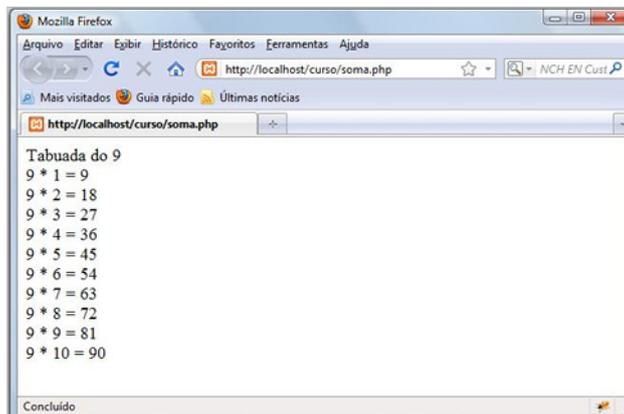


Figura C.22 - Resultado da chamada função `tabuada`
Fonte: do autor

Exemplo de função que recebe um vetor por parâmetro e retorna valor

A função apresentada na figura C.22, chamada `somar_vetor`, recebe por parâmetro um vetor e calcula a soma dos seus valores. Alguns apontamentos:

Na linha 7: Foi criada a variável `$soma`, sendo inicializada com zero.

Na linha 9: É usado um laço `foreach` para percorrer o vetor.

Na linha 10: É acumulado o valor do elemento do vetor na variável `$soma`.

Na linha 12: Após sair do laço, a função retorna o valor da variável `$soma`.

A linha 13: Fecha a função.

A linha 15: Define um array chamado `$vetor` com alguns valores.

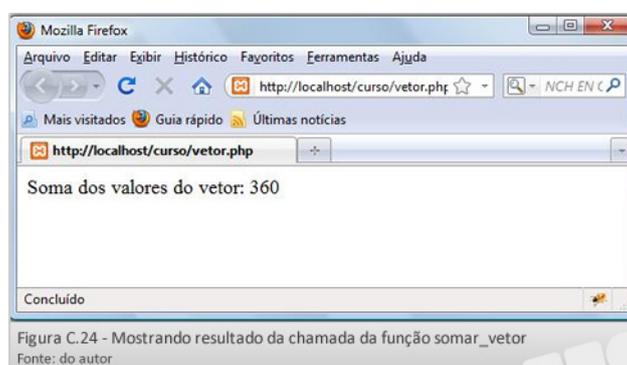
Na linha 17: Mostra uma mensagem com o valor, retornado da chamada da função `somar_vetor`, passando o `$vetor` por parâmetro.

```

1  <?php
2
3  //exemplo de função que recebe por parâmetro um vetor e retorna valor
4
5  function somar_vetor($v) {
6
7      $soma = 0;
8
9      foreach($v as $valor )
10         $soma += $valor;
11
12     return $soma;
13 }
14
15 $vetor = array(44,88,33,1,8,22,1,85,21,57);
16
17 echo "Soma dos valores do vetor: ". somar_vetor($vetor);
18
19 ?>

```

Figura C.23 - Definição da função `somar_vetor`
Fonte: do autor



Exemplo de função que recebe vários parâmetros e retorna valor

A função definida no código da figura C.25, recebe por parâmetro dois números e a operação (+, -, * ou /) que deve ser efetuada para os dois valores. A função deve retornar o resultado do cálculo. Alguns apontamentos:

- a função é definida entre as linhas 2 e 22.
- está sendo usada na função a estrutura `switch` para verificar o que foi recebido em `$op`.
- quando a operação for de divisão está sendo feito um teste para verificar se o denominador não é zero para não ocorrer erro na divisão.

Na linha 21: É retornado o valor calculado.

- para testar a função foram criadas três variáveis (linhas 23 a 25). Neste caso, os valores são 6 e 5 e a operação é multiplicação.

Na linha 27: É feita a chamada da função passando as variáveis por parâmetro.

- A figura C.26 mostra o resultado da chamada da função no navegador.

```

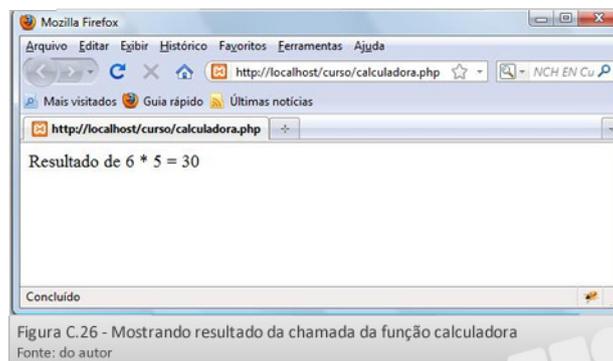
1 <?php
2 function calculadora($v1, $v2, $op) {
3     $res=0;
4     switch ($op) {
5         case "+":
6             $res = $v1 + $v2;
7             break;
8         case "-":
9             $res = $v1 - $v2;
10            break;
11           case "*":
12              $res = $v1 * $v2;
13              break;
14             case "/":
15                 if ($v2 != 0)
16                     $res = $v1 / $v2;
17                 break;
18             default :
19                 $res = "Operação inválida!";
20         }
21         return $res;
22     }
23     $num1 = 6;
24     $num2 = 5;
25     $operacao = "+";
26     echo "Resultado de $num1 $operacao $num2 = ".
27         calculadora($num1, $num2, $operacao);
28     ?>

```

Figura C.25 - Definição da função calculadora
Fonte: do autor

Atenção:

Note que as variáveis que são passadas por parâmetro para uma função não precisam ter o mesmo nome dos argumentos definidos na função.



Funções *require* e *include*

As funções *require* e *include* incluem um arquivo de forma completa no script. A principal diferença entre os dois é que, se usarmos *require()* para incluir um arquivo que não pode ser incluído (talvez o arquivo não exista), um erro fatal será gerado e a execução de código na página será imediatamente suspenso. Se usarmos *include()* e o arquivo de inclusão não puder ser localizado, teremos uma advertência, mas a execução do código na página não será interrompida.

```
<?php
```

```
include("arquivo.php");
```

```
?>
```

Ou

```
<?php

require("arquivo.php");

?>
```

Porque estamos vendo estas duas funções? Porque é muito comum e interessante que o desenvolvedor crie seus arquivos de funções e faça uso deles sempre que precisar com *include* ou *require*. Imagine que você pode criar um arquivo chamado *funcoes.php* com diversas funções comuns que você costuma usar, como na figura C.27. Sempre que necessitar usar essas funções, pode fazer a inclusão no arquivo que precisar. A figura C.28 mostra o código do arquivo *exemplo.php* que na linha 10 faz a inclusão do *funcoes.php*. A partir da inclusão, qualquer uma das funções pode ser usada. A linha 14 faz uso da função *valida_data* do *funcoes.php*. As funções *include* e *require* são bastante utilizadas pelos desenvolvedores e é recomendado que você organize seus arquivos para usá-las.

```
1 <?php
2
3 function valida_cpf($cpf){
4     //código da função
5 }
6
7 function valida_data($data){
8     //código da função
9 }
10
11 function formata_data($data){
12     //código da função
13 }
14
15 function formata_numero($num){
16     //código da função
17 }
18
19 ?>
```

Figura C.27 - Arquivo funcoes.php
Fonte: do autor

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml">
4 <head>
5 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
6 <title>Exemplo</title>
7 </head>
8 <body>
9 <?php
10     include('funcoes.php');
11
12     $dt = "30/11/2000";
13
14     if (valida_data($dt)){
15
16         // código....
17
18     }
19
20 ?>
21 </body>
22 </html>
```

Figura C.28 - Arquivo exemplo.php incluindo o funcoes.php
Fonte: do autor

Síntese

Trabalhar com funções é fundamental para melhorar a organização do código e possibilitar a sua reutilização. Agora você precisa praticar, por isso, faça as atividades propostas, retomando todo o material da Unidade C.

Atividades - Parte 3

1. Faça uma função PHP que recebe três parâmetros: um vetor de strings e duas strings (\$string1 e \$string2). Para cada elemento do vetor, a função deve substituir todas as ocorrências da string1 pela string2. A função deve retornar o vetor ordenado de forma crescente.

2. Considerando o vetor abaixo:

```
$vetor = array("10/04/2011", "14/04/2011", "21/04/2011", "01/05/2011",  
"07/09/2011", "20/09/2011");
```

Faça uma função PHP que recebe por parâmetro:

- um vetor (conforme estrutura acima) e
- um número correspondente a um mês.

A função deve mostrar todas as datas do mês recebido, sendo no formato abaixo:

```
07 de setembro - Quarta-feira
```

```
20 de setembro - Sexta-feira
```

3. Faça uma função PHP que recebe por parâmetro um vetor de notas e gera e retorna um vetor com apenas as notas abaixo de 6.0, ordenando pela chave (índice) do vetor. Considere um vetor associativo.

4. Faça uma função que recebe por parâmetro o \$ano_de_nascimento e retorna a idade da pessoa. Faça um formulário para testar a função.

5. Faça uma função que recebe por parâmetro \$litros e \$km e retorna quantos Km por litro um carro faz. Faça um formulário para testar a função.

6. Faça uma função PHP que recebe por parâmetro um vetor de datas de feriados.

A função deve verificar se a data de hoje é um feriado. Caso a data seja um feriado, a função deve retornar "SIM" em caso negativo retornar "NÃO". Considere o vetor abaixo como exemplo e mostre a chamada da função.

```
$feriados = array("07/09/2010", "20/09/2010", "12/10/2010", "02/11/2010",  
"15/11/2010", "08/12/2010");
```

Linguagem PHP – Parte 4

Nesta parte 4 da Unidade C, vamos introduzir o conceito de sessões e cookies e mostrar como eles podem ser usados em PHP. Estes dois recursos são muito importantes no desenvolvimento de aplicações para o ambiente Web e é imprescindível que o desenvolvedor tenha conhecimento para manipulá-los. Ao final desta parte são apresentadas as atividades para realização.

Sessões

É muito comum em sites a necessidade de manter informações do usuário enquanto este estiver navegando (manter informações entre as diversas páginas dentro do site). O exemplo mais comum é quando você informa login e senha em algum site para acessar uma área de dados restrita ao seu usuário, por exemplo, para e-mails. Ou ainda, quando você compra pela internet e coloca os produtos escolhidos no carrinho de compras, os dados do carrinho de compras são exclusivamente seus.

Mas então, como estas informações são mantidas entre as páginas? Bem, a forma mais usual é a SESSÃO. Primeiro precisamos entender o que é a sessão, para depois vermos como trabalhar com ela em PHP.

A Sessão, de modo geral, é uma área do servidor onde podemos guardar valores para recuperar depois. Cada usuário que se conecta ao site recebe uma sessão (um identificador). Os dados colocados na sessão só são removidos quando a sessão é encerrada (navegador é fechado) ou quando a sessão expira por um período de inatividade configurado pelo programador.

Agora vamos começar a trabalhar na prática para que você entenda melhor.

A sessão precisa ser iniciada em cada página que você for usar ou definir um valor. Para abrir a sessão é só usar a função abaixo:

```
session_start(); // Inicia a sessão
```

Atenção:

É necessário iniciar a sessão antes de qualquer echo ou de inserir qualquer HTML fora de blocos php. Geralmente o início da sessão está no começo da página.

Depois de iniciada a sessão você pode definir valores usando o array superglobal do PHP:

```
$_SESSION["login"] = "maria";
```

Vamos visualizar isto em uma sessão. Crie o arquivo *sessao1.php* conforme código da figura C29.



```

1 <?php
2
3     session_start();
4
5     $_SESSION["login"] = "maria";
6
7     echo "Criando variável na sessão...";
8
9 ?>

```

Figura C.29 - Código do arquivo sessao1.php
Fonte: do autor

Ao visualizar o arquivo *sessao1.php* no navegador, apenas a mensagem será mostrada. Mas o que nos interessa é verificar onde a variável ficou armazenada no servidor. Como estamos trabalhando de forma

local, podemos acessar estes dados. Abra a pasta `c:\xampp\tmp`. Você verá que um arquivo foi gerado para a sessão criada a partir do código do arquivo `sessao1.php`. A figura C.30 mostra a pasta com o nome do arquivo gerado. O nome do arquivo inicia com `sess_` e depois vem o identificador da sessão.

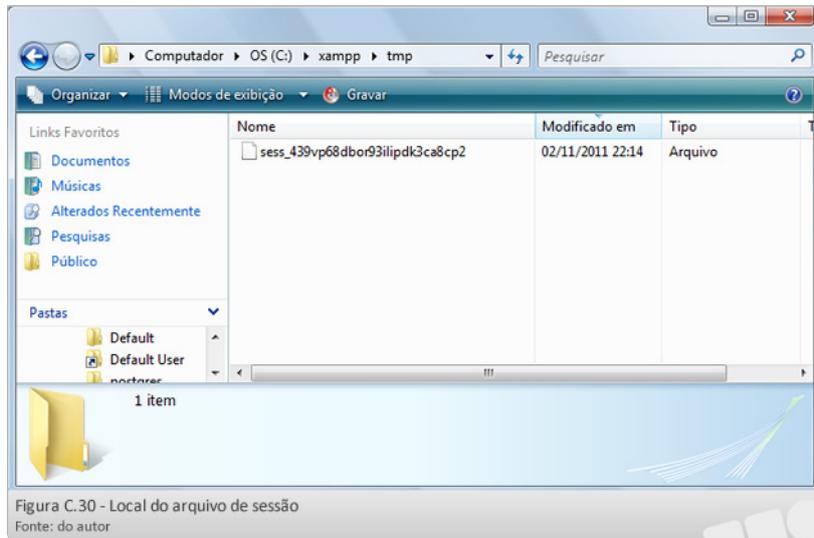


Figura C.30 - Local do arquivo de sessão
Fonte: do autor

Ao abrir o arquivo, é possível verificar os dados da sessão. A figura C.31 mostra o arquivo aberto no bloco de notas. Primeiro vem o nome da variável colocada na sessão, depois do pipeline vem a letra `s` indicando que a variável é do tipo `string`, e depois dos dois pontos vem o valor correspondente à variável (`maria`).

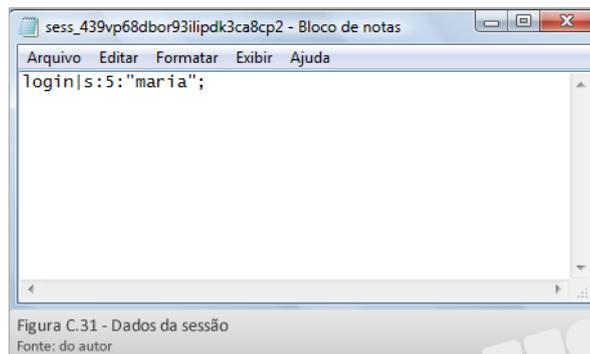


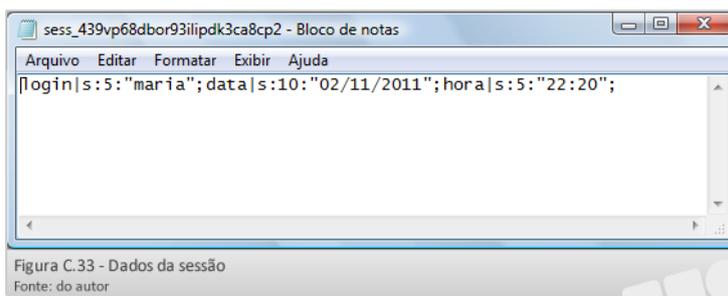
Figura C.31 - Dados da sessão
Fonte: do autor

Contudo, uma sessão pode conter muitas variáveis. Vamos alterar um pouco nosso código para acrescentar mais variáveis. Veja como ficou na figura C.32. Também usamos agora a função `session_id()` que retorna o identificador da sessão, aquele que também é usado no nome do arquivo. Assim, você poderá ver no navegador o identificador da sessão. A figura C.33 mostra o arquivo agora com as novas variáveis registradas na sessão.

```

1 <?php
2
3     session_start();
4
5     $_SESSION["login"] = "maria";
6     $_SESSION["data"]  = "02/11/2011";
7     $_SESSION["hora"]  = "22:20";
8
9     $id_sessao = session_id(); //identificador da sessão
10
11     echo "Criando variáveis na sessão $id_sessao ";
12
13 ?>
```

Figura C.32 - Código do arquivo `sessao1.php` com mais variáveis
Fonte: do autor



Como recuperar estes dados da sessão? O código da figura C.34 mostra como verificar se existe um dado na sessão e como podemos recuperá-lo. Usamos a função *isset* para verifica se a variável está definida (retorna *true* se estiver e *false* se não existir).

```

1  <?php
2      session_start();
3
4      if (isset($_SESSION['login'])) {
5
6          echo $_SESSION['login'];
7
8      }else {
9
10         echo "usuário não logado";
11     }
12
13  ?>

```

Figura C.34 - recuperar valor da sessão
Fonte: do autor

Outras funções importantes para trabalhar com sessões são apresentadas na tabela abaixo:

Função	Descrição
<code>unset()</code>	Exclui a uma variável específica da sessão. Ex.: <code>unset(\$_SESSION["login"]);</code>
<code>session_destroy()</code>	Destrói toda a sessão, eliminando todas as variáveis salvas nela. Antes de chamar a função <code>session_destroy()</code> , deve-se primeiro abrir a sessão com <code>session_start()</code> .Essa função é normalmente utilizada quando um usuário requisita sua saída da aplicação (<i>logout</i>).
<code>session_cache_expire()</code>	Retorna o tempo de expiração da sessão ou define novo tempo de expiração em minutos. Para redefinir o tempo de expiração, deve ser executada antes de <code>session_start()</code> . O tempo padrão de duração da sessão é de 180 minutos. Para alterar o tempo padrão, deve-se chamar <code>session_cache_expire()</code> a cada página que faça uso da sessão.

Cookies

Outra forma de manter dados para posterior acesso em outras páginas do site é através do uso de cookies. Só que diferentemente da sessão, o **cookie** é um arquivo texto armazenado no computador do usuário e pode ser recuperado posteriormente pelo servidor.

O uso de cookies, por exemplo, permite que o usuário após ter se autenticado em um site, desligue o computador, acesse o site um tempo depois e não precise se autenticar novamente. Ex: GMail, Hotmail, Yahoo, etc. Para utilizar esse recurso, geralmente o usuário precisa aceitar algum tipo de opção como

“salvar as minhas informações neste computador”. Além disso, é preciso que o navegador do usuário esteja com a opção de armazenar cookies habilitada.

Vamos ver como isso pode ser configurado no Firefox:

- clique no menu Ferramentas ou *Tools* e depois em Opções ou *Options*;
- clique na guia Privacidade ou *Privacy* e, na seção Cookies, ative a opção “Sites podem definir Cookies” ou “*Allow sites to set cookies*”. Veja na figura C.35.

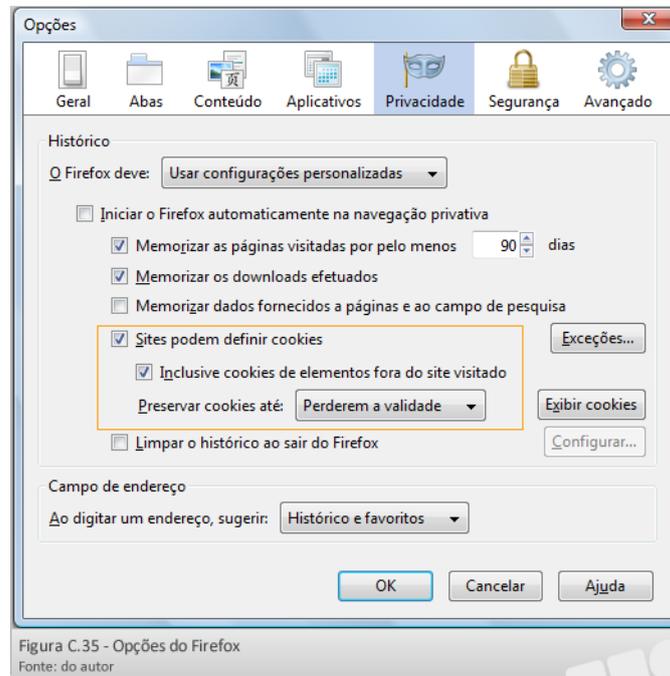


Figura C.35 - Opções do Firefox
Fonte: do autor

Vamos ver agora como definir um cookie em PHP:

```
<?php
    setcookie('nome_cookie', 'valor do cookie');
?>
```

Na definição de um cookie os dois primeiros parâmetros são obrigatórios: *nome do cookie* e *valor do cookie*. No entanto, outros parâmetros podem ser definidos. Veja abaixo:

```
setcookie(nome, valor, expira, caminho, domínio, seguro, http)
```

Nome	nome do cookie.
Valor	valor do cookie.
Expira	tempo para o cookie expirar (número de segundos desde 01/01/1970).
Caminho	caminho no servidor aonde o cookie estará disponível.
Domínio	domínio para qual o domínio estará disponível.
Seguro	(<i>true/false</i>) indica que o cookie só poderá ser transmitido sob uma conexão segura HTTPS do cliente.
Http	(<i>true/false</i>) quando for TRUE, o cookie será acessível somente sob o protocolo HTTP. Isso significa que o cookie não será acessível por linguagens de script, como JavaScript.

Vamos a um exemplo. Veja o código da figura C.36 e faça o teste na sua máquina.

```

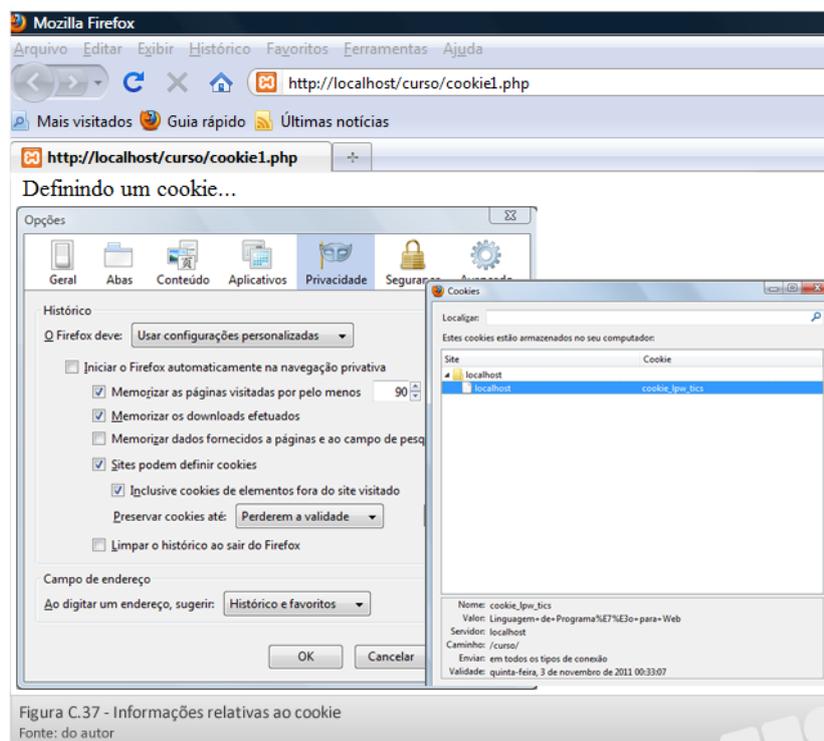
1 <?php
2
3     setcookie( 'cookie_lpw_tics' , 'Linguagem de Programação para Web' , time()+ 3600 );
4
5     echo "Definindo um cookie...";
6 ?>
    
```

Figura C.36 - Código
Fonte: do autor

Você deve ter observado que na função *setcookie* foram passados três parâmetros. O terceiro parâmetro é referente ao tempo de validade do cookie. Nesse caso, foi definido que o cookie tem validade de uma hora (tempo atual + 3600 segundos). Vamos ver onde este cookie foi criado na sua máquina. Se o seu navegador for Firefox, acesse o menu Ferramentas e depois Opções. Clique em “Exibir Cookies”. A figura C.37 mostra a visualização da janela que é apresentada. Na figura é possível observar as informações relativas ao cookie.

Dica:

Se você possuir muitos cookies, uma opção é excluir todos eles e depois executar o arquivo para geração do cookie. Assim, você vai encontrar facilmente o cookie gerado.



Para recuperar os dados do cookie usamos o array superglobal do PHP `$_COOKIE[]`, passando o nome do cookie como parâmetro. Veja na figura C.38 um código de exemplo.

```

1 <?php
2
3     $dados = $_COOKIE['cookie_lpw_tics'];
4
5     echo "Dados recuperados do cookie: $dados";
6
7 ?>
    
```

Figura C.38 - Recuperar dados do cookie
Fonte: do autor

Se for necessário remover um cookie, basta usar a mesma função `setcookie` com o nome do cookie e com a string vazia (um exemplo pode ser visto na figura C.39).

```

1 <?php
2
3     setcookie('cookie_lpw_tics', '');
4
5     echo "excluindo cookie...";
6 ?>

```

Figura C.39 - Excluindo o cookie
Fonte: do autor

Síntese

O uso de sessões e cookies é muito comum em web sites, por exemplo, sessões é sempre usado para autenticação de usuários e cookies para armazenar dados do perfil do usuário para uma posterior visita, podendo mostrar produtos que são de seu interesse. Bem, agora é com você, faça as atividades propostas para ganhar mais experiência com essas novas funções.

Leitura:

Para saber mais e ver outros exemplos sobre sessões e cookies, leia o capítulo 13 da seguinte bibliografia: NIEDERAUER, J. Desenvolvendo websites com PHP: Aprenda a criar websites dinâmicos e interativos. São Paulo: Novatec, 2004.

Atividades - Parte 4

1. Crie uma página PHP que coloca na sessão um contador iniciando em 1. Cada vez que a página for executada dentro da sessão (existindo já a variável) o contador é incrementado em 1. Mostre sempre o valor do contador na página e o identificador da sessão.
2. Dado o código PHP da figura abaixo, faça o arquivo `dados_sessao.php` para mostrar todos os dados da sessão.

```

1 <?php
2
3     session_start();
4
5     $_SESSION["data"] = date("d/m/Y");
6     $_SESSION["hora"] = date("H:i:s");
7     $_SESSION["dia"] = date("w");
8
9     $id_sessao = session_id(); //identificador da sessão
10
11     echo "Criando variáveis na sessão $id_sessao ";
12 ?>
13 <br /><br />
14 <a href="dados_sessao.php"> Ver dados da sessão</a>

```

Figura C.40 - Exercício 2
Fonte: do autor

3. Utilizando cookies, crie uma página com um mecanismo que conte o número de acessos de um mesmo usuário a essa página. Mantenha esta contagem válida por 1 minuto (desde o último acesso).
4. Crie um formulário de login para autenticação e mantenha os dados de login do usuário por meia hora, mesmo depois de o usuário ter fechado o navegador.



TICS



Linguagem PHP com acesso a Banco de Dados

Unidade D
Linguagem de Programação Web

LINGUAGEM PHP COM ACESSO A BANCO DE DADOS

PHP com Banco de Dados

Nesta unidade vamos abordar em detalhes o acesso a banco de dados com a linguagem PHP. Hoje é fundamental criarmos sites dinâmicos que facilitem a atualização de conteúdo. Isso só é possível com o armazenamento dos dados em banco de dados. Além disso, a tendência é que as aplicações, como por exemplo, sistemas de gestão empresarial, gestão de pessoas, entre outros, migrem para o ambiente web, ou seja, sejam desenvolvidos para acesso no navegador.

Como já comentamos anteriormente, o PHP possibilita acesso a diversos bancos de dados. Nesta unidade vamos apresentar a integração com o banco de dados MySQL, mas de uma forma que fica fácil a utilização de outros bancos de dados.

Inicialmente veremos quais as funções principais para acesso ao banco de dados por meio da linguagem PHP, para depois criarmos um exemplo completo de manutenção de dados e posterior apresentação no site. Quando nos referimos a uma manutenção completa quer dizer ter as funções de incluir, alterar e excluir dados de uma tabela do banco de dados. Então, mãos a obra...

Sites Dinâmicos

Antes de iniciarmos, vamos ver uma situação de uso de banco de dados para que você compreenda como funcionam os sites dinâmicos. Imagine um que site precisa mostrar quais as cidades que o IFSUL tem campus. Sabemos que os Institutos Federais estão em expansão e por isso novos campi podem surgir. Por isso, é interessante termos essa informação em uma base de dados e fornecer uma forma de o administrador do site manter estes dados. Um site dinâmico (com banco de dados) possui o que chamamos de **Área Administrativa**, onde apenas pessoas com permissão podem ter acesso e manter as informações do site. Para acessar esta área o usuário precisa se autenticar informando login e senha. As figuras D.1 e D.2 apresentam um exemplo de área administrativa para manter os dados de cidades (cidades que possuem campi do IFSUL).

A figura D.1 apresenta uma página que lista as cidades (armazenadas em uma tabela do banco de dados). Observe que possui links para incluir uma nova cidade, para alterar os dados de cada cidade e para excluir cada cidade. Quando os links incluir ou alterar forem clicados será aberta uma outra página com um formulário para entrada de dados. A figura D.2 apresenta a página do formulário para incluir dados de uma nova cidade. Este é um exemplo de Área Administrativa para manter dados do banco de dados.

Por outro lado, o site público aos usuários busca os dados deste banco para apresentar as informações aos internautas. A figura D.3 apresenta o que chamamos de **Área Pública** do site, onde qualquer usuário pode visualizar as informações.

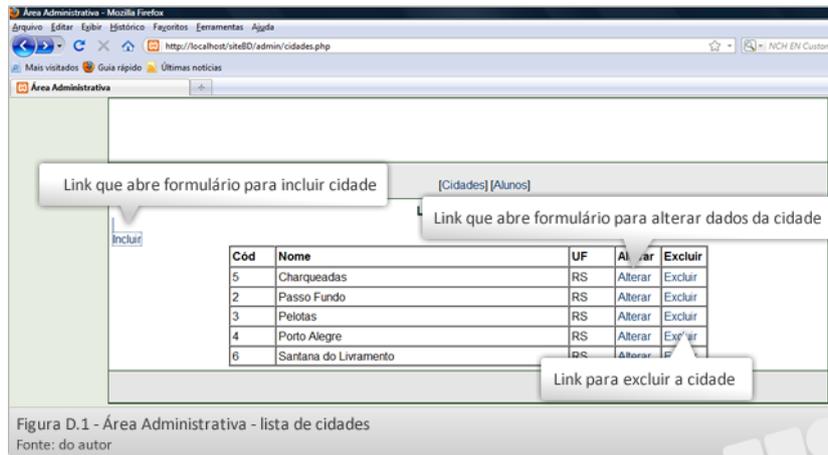


Figura D.1 - Área Administrativa - lista de cidades
Fonte: do autor

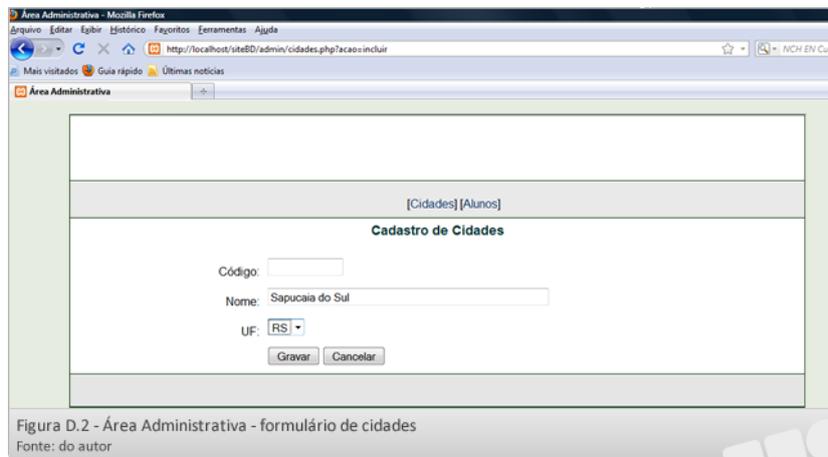


Figura D.2 - Área Administrativa - formulário de cidades
Fonte: do autor

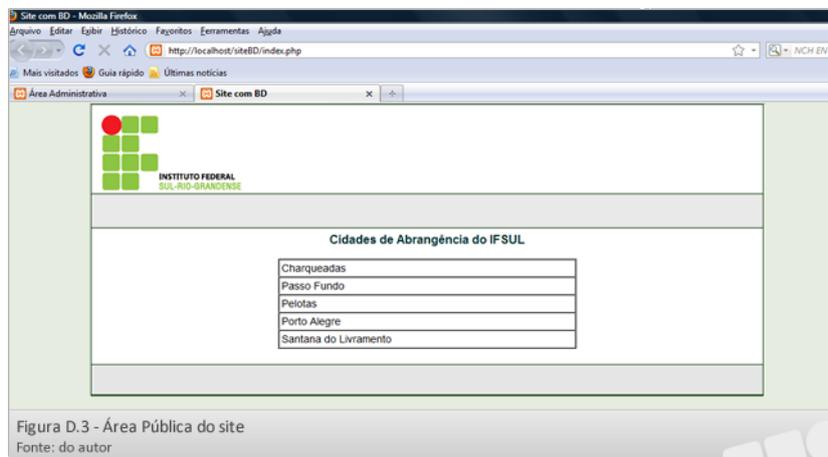


Figura D.3 - Área Pública do site
Fonte: do autor

Configurações iniciais

Antes de começarmos a trabalhar, vamos ver o que vamos precisar para realizar nossos testes.

Biblioteca ADODB para acesso a banco de dados com PHP

Para acesso ao banco de dados MySQL vamos utilizar uma biblioteca chamada ADODB (*Active Data Objects Data Base*). ADODB é conjunto de bibliotecas/classes que padronizam as funções de banco de dados do PHP. O objetivo é ocultar as diferenças existentes entre os diferentes bancos de dados e prover um método simples de fazer consultas e manipulações nas bases de dados com o mínimo de alteração de código. Desta forma, podemos com uma simples alteração mudar o banco de dados que estamos utilizando para outro (por exemplo de MySQL para PostgreSQL, se o esquema do banco de dados for o mesmo).

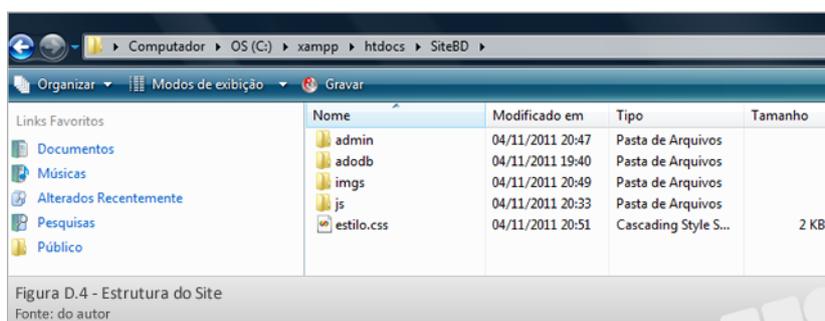
Você pode fazer download da biblioteca em

[<http://sourceforge.net/projects/adodb/files/>](http://sourceforge.net/projects/adodb/files/)

Com a ADODB podemos acessar os bancos de dados: MySQL, Oracle, SQL Server, Sybase, Informix, PostgreSQL, FrontBase, SQLite, Interbase/Firebird, FoxPro, Access, ADO, DB2, SAP DB, ODBC.

Para dar sequência vamos organizar uma estrutura de site para realizarmos nossos testes. Você poderá fazer o download do material disponibilizado que se chama **siteBD.zip**.

O arquivo deve ser descompactado em C:\xampp\htdocs. A estrutura ficará como da figura D.4.



Alguns apontamentos sobre a estrutura:

- pasta admin conterá todos os arquivos da Área Administrativa
- pasta adodb é da biblioteca ADODB que vamos usar
- pasta imgs para imagens do site
- pasta js para arquivos de JavaScript do site
- arquivo estilo.css contém as folhas de estilo usadas no site

Atenção:

Esta disciplina não aborda CSS (folha de estilo). Porém você já deve ter visto em outra disciplina que CSS é uma linguagem para estilos que define o layout de documentos (X)HTML. Por exemplo, CSS controla fontes, cores, margens, linhas, alturas, larguras, imagens de fundo, posicionamentos e muito mais.

Banco de Dados MySQL

Agora vamos criar nosso banco de dados para que possamos depois através do PHP acessar o MySQL. Com o servidor web Apache e o banco de dados MySQL rodando, acesse no navegador o endereço:

[<http://localhost/phpmyadmin/>](http://localhost/phpmyadmin/)

Você estará acessando uma ferramenta para administrar o MySQL, é o phpMyAdmin. Esta ferramenta já foi instalada junto com o Xampp e facilita a criação do banco de dados.

Agora crie o banco de dados chamado “curso”.

Neste banco vamos criar, inicialmente, duas tabelas: cidades e alunos, conforme script abaixo:

```
CREATE TABLE IF NOT EXISTS cidades (
  id BIGINT NOT NULL auto_increment ,
  nome VARCHAR( 50 ) NOT NULL ,
  uf CHAR( 2 ) NOT NULL ,
```

```
PRIMARY KEY ( id )
);

CREATE TABLE IF NOT EXISTS alunos (
    id bigint NOT NULL auto_increment,
    nome varchar(100) NOT NULL,
    email varchar(100) NOT NULL,
    fone varchar(15) NOT NULL,
    nascimento date NOT NULL,
    cidade int(11) NOT NULL,
    PRIMARY KEY (id)
);
```

Parada Obrigatória:

Antes de continuar assista o vídeo **Instalação e configuração de ferramentas**, disponível no Ambiente Virtual de Aprendizagem em < <http://www.youtube.com/embed/7zqhZtD2mo0>>.

Conexão com o banco de dados

Para conectar com o banco de dados vamos usar um método da biblioteca ADODB. Crie um arquivo na pasta admin nomeado de `conexao.php`. Vejamos o código do `conexao.php` da figura D.5 e vamos analisá-lo:

Linha 3 – Define uma variável para especificar o caminho da biblioteca ADODB (lembre-se que o `conexao.php` está na pasta `admin`, por isso estamos saindo desta pasta para então entrar em `adodb`).

Linha 4 – Faz include do arquivo `adodb.inc.php` da pasta da biblioteca ADODB. Por convenção os arquivos que são `inc.php` indicam que serão arquivos para include.

Linha 6 – Cria a instância `$con` da conexão com o tipo de banco especificado, neste caso MySQL. Por exemplo, se o banco para acesso fosse o PostgreSQL, seria passado por parâmetro ‘`postgres`’.

Linha 8 – Desabilita a opção de debug. Esta opção é interessante para o desenvolvedor identificar erros durando a programação. Quando precisar verificar como os comandos estão sendo executados no banco de dados, esta opção pode ficar ligada, atribuindo `true`. Por hora vamos deixar desabilitada (`false`).

Linha 10 - Executa o método `Connect` com os parâmetros de conexão (servidor, usuário, senha, nome_banco). Neste caso estamos usando o usuário padrão “`root`”, que inicialmente não possui senha. É interessante criar um usuário com senha específico para cada banco de dados. Após a conexão vem “`or die(...)`”, esta função mostra uma mensagem caso ocorra um erro na execução do comando e interrompe o script.

```

1 <?php
2
3 $caminho_adodb = "../adodb";
4 include("$caminho_adodb/adodb.inc.php");
5
6 $con = ADONewConnection('mysql');
7
8 $con->debug = false;/// Debug do SQL
9
10 $con->Connect("localhost", "root", "", "curso") or die
11 ("Erro ao conectar o banco de dados");
12
13 echo "conectou com o banco de dados curso";
14
15 ?>

```

Figura D.5 - Código para conexão com o banco de dados
Fonte: do autor

Dica:

A função **die()** aborta imediatamente a execução da aplicação. Essa é uma função simples para o tratamento de erros, pois ela encerra a execução do script.

Crie o arquivo *conexao.php* e faça o teste no navegador. Este arquivo de conexão será usado toda vez que precisarmos interagir com o banco de dados, por isso, após fazer os testes e verificar que está conectando, apague a linha 13 do arquivo.

Executar SQL

Toda e qualquer manipulação de dados do banco de dados será feita usando a linguagem SQL. Vamos ver como é o script SQL para incluir um registro na tabela **'cidades'** do nosso banco **'curso'**:

```
insert into cidades (nome, uf) values ('Pelotas','RS')
```

Você observou que o campo **id** não aparece no SQL? Isso porque ele foi definido como *auto_increment* quando a tabela foi criada. Com isso, não devemos passar um valor para campo **id**, ele terá seu valor gerado automaticamente. Este comando SQL cria um novo registro na tabela. Agora vamos ver como fica a execução no PHP com a biblioteca ADODB. A figura D.6 mostra o código.

```

1 <?php
2
3 include('conexao.php');
4
5 $sql = "insert into cidades (nome, uf) values ('Pelotas','RS')";
6
7 $rs = $con->Execute($sql);
8
9 if (!$rs) {
10     echo $con->ErrorMsg();
11 }else
12     echo "Dados incluídos";
13
14 ?>

```

Figura D.6 - Inserir dados na tabela
Fonte: do autor

Para este teste crie o arquivo *inserir.php* na pasta *c:\xampp\htdocs\siteBD\admin*, como o código acima. Vejamos alguns apontamentos:

Linha 3 – Faz include do arquivo *conexao.php* que possui a conexão com o banco de dados. Veja como é interessante ter este arquivo, pois se precisar, por exemplo, trocar a senha, será necessário alterar apenas em um único arquivo;

Linha 5 – Cria a variável *\$sql* com o comando SQL;

Linha 7 – Executa o SQL (que está na variável `$sql`). O resultado é atribuído para variável `$rs` (recebe `false` se ocorrer erro ou o retorno do SQL se realizado com sucesso). Veja que `$con` é a variável da conexão que foi criada no `conexao.php`. Mai uma coisinha: `$rs` vem de *result set* (conjunto de resultados).

Linhas 9 e 10 – Testa se ocorreu erro, se `$rs` for `false` (com `!` vai se tornar `true`), ou seja, tem erro, mostra a mensagem retornada do banco de dados.

Atenção

Se o código não possuir erros, a execução do arquivo gerará um novo registro na tabela `idades`, porém se executares o código mais de uma vez, será criada mais uma cidade com o mesmo nome. Por isso, para testar várias vezes, troque o nome da cidade para incluir cidades diferentes. Visualize as cidades usando o phpMyAdmin <<http://localhost/phpmyadmin>>.

Bem, agora precisamos alterar um registro de cidade, como faremos? A lógica é a mesma, o que vai mudar é o comando SQL. O comando abaixo altera o nome da cidade cujo id (identificador) da cidade é 2:

```
update cidades set nome = 'PELOTAS' where id = 2
```

Observe que o nome da cidade ficou entre aspas simples. Isso porque o campo nome no banco é um `varchar`. O mesmo vale para os tipos `char` e `date`. Já campos do tipo numérico, como `int`, `bigint`, `decimal`, não deve-se usar aspas. Outra dica aqui é sempre atribuir o SQL dentro de aspas duplas:

```
$sql = "update cidades set nome = 'PELOTAS' where id = 2 ";
```

Para o processo de alteração dos dados crie o arquivo `alterar.php` (também na pasta `admin`). A figura D.7 mostra o código do `alterar.php`. Veja que a estrutura é a mesma, o que muda é o SQL.

```
1 <?php
2
3     include('conexao.php');
4
5     $sql = "update cidades set nome = 'PELOTAS' where id = 2 ";
6
7     $rs = $con->Execute($sql);
8
9     if (!$rs) {
10         echo $con->ErrorMsg();
11     }else
12         echo "Dados alterados";
13
14 ?>
```

Figura D.7 - alterar um registro da tabela
Fonte: do autor

Para a exclusão de uma cidade seguimos a mesma lógica. Crie o arquivo `excluir.php` (na pasta `admin`) com o código apresentado na figura D.8.

```
1 <?php
2
3     include('conexao.php');
4
5     $sql = "delete from cidades where id = 1";
6
7     $rs = $con->Execute($sql);
8
9     if (!$rs) {
10         echo $con->ErrorMsg();
11     }else
12         echo "Cidade excluída";
13
14 ?>
```

Figura D.8 - excluir um registro da tabela
Fonte: do autor

Listar dados

Já vimos como incluir, alterar e excluir dados. Só falta a listagem de dados. Da mesma forma vamos executar um SQL (select), mas precisaremos tratar os dados retornados, que geralmente são vários registros. Se precisarmos listar todos registros da tabela cidades (todas as cidades), o comando SQL será:

```
select * from cidades order by nome
```

Vamos dar uma olhada no código da figura D.9 (arquivo *listar.php*).

```

1  <?php
2
3      include('conexao.php');
4
5      $sql = "select * from cidades order by nome";
6
7      $rs = $con->Execute($sql);
8
9      if (!$rs) {
10
11          print $con->ErrorMsg();
12
13      }else
14
15      while(!$rs->EOF){
16
17          $nome = $rs->fields['nome'];
18          $uf   = $rs->fields['uf'];
19
20          echo "$nome - $uf <br/> ";
21
22          $rs->MoveNext();
23      }
24  ?>

```

Figura D.9 - código para listar dados
Fonte: do autor

Para este teste crie o arquivo *listar.php* na pasta *c:\xampp\htdocs\siteBD\admin*, com o código acima. Vejamos alguns apontamentos:

Linha 3 – Faz include do arquivo *conexao.php* que possui a conexão com o banco de dados;

Linha 5 – Cria a variável *\$sql* com o comando SQL;

Linha 7 – Executa o SQL (que está na variável *\$sql*). O resultado é atribuído para variável *\$rs* (recebe *false* se ocorrer erro ou o retorno do SQL se realizado com sucesso). *\$rs* faz referência a conjunto de resultados, que são vários registros de cidades;

Linhas 9 e 11 – Testa se ocorreu erro, se *\$rs* for *false* (com ! vai se tornar *true*), ou seja, tem erro, mostra a mensagem retornada do banco de dados;

Linha 13 – Se não ocorreu erro então vai executar o código para mostrar os dados;

Linha 15 – Para listar os dados vamos precisar de um laço, cada vez que passar pelo laço lista uma cidade. Este laço *while* verifica se possui resultado para listar: *EOF* sigla de *END OF FILE* que significa “fim de arquivo”. *\$rs->EOF* retorna *true* se for fim do arquivo e *false* em caso contrário. O laço pára a execução quando não possuir mais resultados (chega ao fim dos resultados);

Linhas 17 e 18 – Recupera os dados que deseja mostrar (atribui para as variáveis *\$nome* e *\$uf*). Para pegar os campos usa *\$rs->fields['nome_campo']* onde *nome_campo* é exatamente o nome do campo no banco de dados. No nosso caso queremos mostrar o nome da cidade (*nome*) e a unidade federativa (*uf*).

Linha 20 – Mostra os dados da cidade e quebra linha;

Linha 22 – Para passar ao próximo registro executa *\$rs->MoveNext()*;

A figura D.10 mostra o resultado do código listar.php no navegador, listando uma cidade por linha.

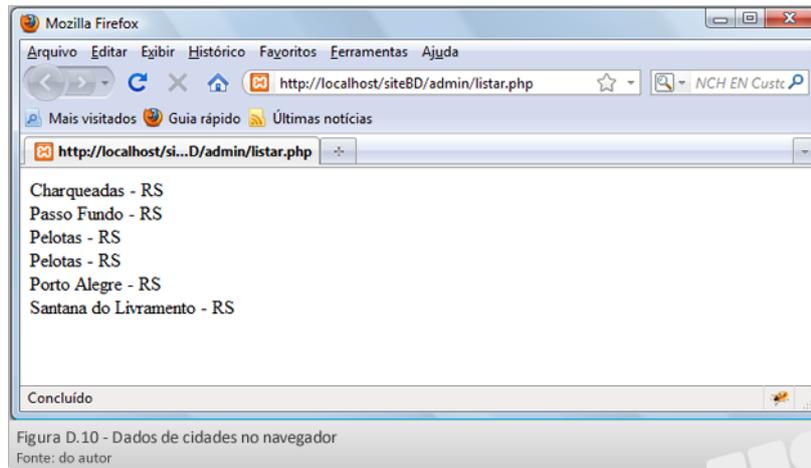


Figura D.10 - Dados de cidades no navegador
Fonte: do autor

Síntese

Bem, para acessar banco de dados com a linguagem PHP utilizamos a biblioteca ADODB. Seu uso é interessante por abstrair aspectos de acesso ao banco, tornando mais fácil a migração de um banco para outro. Vimos como fazer a conexão com o banco e usamos linguagem SQL para manipular os dados com o banco de dados. Principalmente, vimos como tratar dados retornados do banco de dados. Agora precisamos integrar melhor isso com o (X)HTML, assim como mostramos no início desta Unidade nas figura D.1, D.2 e D.3. Antes de seguirmos, é importante que esta parte tenha sido bem assimilada, por isso faça a atividade proposta a seguir.

Atividades - Parte 1

1. Agora é com você. Dado o script abaixo de criação da tabela cursos crie os arquivos:

incluir_cursos.php – para incluir um registro em cursos
 alterar_cursos.php – para alterar um registro de cursos
 excluir_cursos.php – para excluir um registro de cursos
 listar_cursos.php – para listar todos os cursos

```
CREATE TABLE IF NOT EXISTS cursos (
    id BIGINT NOT NULL auto_increment ,
    nome VARCHAR( 50 ) NOT NULL ,
    horas integer NOT NULL ,
    PRIMARY KEY ( id )
);
```

Obs.: crie a tabela cursos no mesmo banco de dados já criado (curso)

Após a realização das atividades participe do chat em horário marcado pelo professor formador para discutir questões relativas aos exercícios propostos.

Área Administrativa

Nesta segunda parte da unidade D vamos desenvolver passo a passo uma manutenção completa, incluindo listagem de dados, inclusão, alteração e exclusão (como foi mostrado nas figuras D.1, D.2 e D.3 da primeira parte da Unidade D). Utilizaremos a estrutura já passada do site *siteBD* e o banco de dados *curso*. Este material foi preparado de forma que você acompanhe fazendo cada passo do processo de construção da solução.

Estrutura de arquivos da área administrativa

A pasta *admin* do projeto já possui alguns arquivos. Veja na tabela abaixo uma breve descrição sobre cada arquivo.

Arquivo	Descrição
<code>acessonegado.php</code>	Arquivo que mostra mensagem de acesso negado à área administrativa
<code>conexao.php</code>	Arquivo para conexão com o banco de dados
<code>funcoes.php</code>	Arquivo de funções
<code>index.php</code>	Arquivo inicial da área administrativa. Mostra apenas o menu
<code>login.php</code>	Arquivo com formulário para login
<code>logininvalido.php</code>	Arquivo que mostra mensagem de login inválido para o usuário que está tentando se logar na área administrativa
<code>menusup.php</code>	Arquivo com as opções de menu

Atenção:

Para o usuário utilizar a área administrativa é necessário a autenticação informando login e senha. Veremos este processo na última parte desta unidade.

O arquivo *index.php* é o primeiro a ser apresentado (por default) para o usuário na área administrativa. Vamos ver a sua estrutura:

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
2 <html xmlns="http://www.w3.org/1999/xhtml">
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5 <title>Área Administrativa</title>
6 <link href="../estilo.css" rel="stylesheet" type="text/css" />
7 </head>
8 <body>
9 <div id="geral">
10 <div id="topo"></div>
11 <div id="menu">
12 <<?php include('menusup.php');?>
13 </div>
14 <div id="conteudo">
15 </div>
16 <div id="rodape"></div>
17 </div>
18 </body>
19 </html>

```

Figura D.11 - Estrutura do arquivo index.php
Fonte: do autor

No corpo do documento existe uma estrutura de *divs* para organizar o layout e que são formatadas por folha de estilo (arquivo *estilo.css*). Na *div menu* faz o include do arquivo *menusup.php* com as opções de

menu (figura D.12). As opções de menu por enquanto são duas, cidades e alunos. Caso seja necessário incluir mais opções basta acrescentar neste arquivo.

```

1 <ul>
2   <li>[<a href="cidades.php">Cidades</a>]</li>
3   <li>[<a href="alunos.php">Alunos</a>]</li>
4 </ul>

```

Figura D.12 - Código do arquivo menusup.php
Fonte: do autor

A figura D.13 mostra a visualização do *index.php* no navegador.



Manutenção completa

Nosso objetivo é mostrar a manutenção completa da tabela de cidades (listar, incluir, alterar e excluir) na área administrativa do site (*siteBD*). Para tanto, trabalharemos com uma estrutura de manutenção com os seguintes arquivos:

Arquivo	Descrição
cidades_list.php	É o arquivo para listar os dados de cidades. Contém código (X)HTML e PHP
cidades_form.php	É o arquivo que contém o formulário para incluir e alterar dados. Contém código (X)HTML e PHP.
cidades.php	É o gerenciador da manutenção. Este arquivo contém apenas código PHP e é responsável por receber as requisições do usuário e proceder a execução do código correspondente.

Os arquivos *cidades_list.php* e *cidades_form.php* inicialmente possuem a mesma estrutura do *index.php*. A partir desta estrutura básica, primeiro vamos incluir a parte (X)HTML para depois fazer a programação PHP.

- Para começar, vamos ver como é a estrutura básica do arquivo *cidades_list.php* (figura D.14). Observe que a figura está mostrando apenas a parte de código que está na *div conteúdo*, pois já vimos na figura D.11 a estrutura básica. Por hora o código não possui programação PHP, apenas a estrutura (X)HTML. Na figura D.15 você pode conferir como fica a apresentação do *cidades_list.php* no navegador. Alguns apontamentos sobre o código:
- Veja que existem alguns links, na linha 17, 30 e 33. Todos são direcionados para *cidades.php*, este arquivo é o gerenciador da manutenção, por isso, sempre ele será requisitado. Veja que é uma requisição GET passando por parâmetro um valor para *acao*. Este parâmetro *acao* (ação) indica qual a intenção do usuário, neste arquivo pode ser: incluir, alterar e excluir.
- Entre as linhas 19 e 25 temos a primeira linha da tabela que é o cabeçalho;
- Entre as linhas 26 e 35 temos a segunda linha da tabela. Esta linha posteriormente será repetida para cada registro da tabela *cidades* mostrando seus dados. Na penúltima coluna tem o link para alterar e na última coluna há um link para excluir que faz uma chamada da JavaScript para confirmar a exclusão.

```

15 <div id="conteudo">
16 <div id="titulo">Lista de Cidades</div>
17 <a href="cidades.php?acao=incluir"><br />Incluir</a><br />
18 <table width="600" border="1" align="center" cellpadding="3" cellspacing="0">
19 <tr>
20 <th width="50" scope="col">Cód</th>
21 <th scope="col">Nome</th>
22 <th width="50" scope="col">UF</th>
23 <th width="50" scope="col">Alterar</th>
24 <th width="50" scope="col">Excluir</th>
25 </tr>
26 <tr>
27 <td id</td>
28 <td nome</td>
29 <td uf</td>
30 <td><a href="cidades.php?acao=alterar">Alterar</a></td>
31 <td><a href="javascript:
32 if (confirm('Confirma exclusão?')){
33 location='cidades.php?acao=excluir'
34 }">Excluir</a></td>
35 </tr>
36 </table>
37 </div>
    
```

Figura D.14 - Código do cidades_list.php
Fonte: do autor

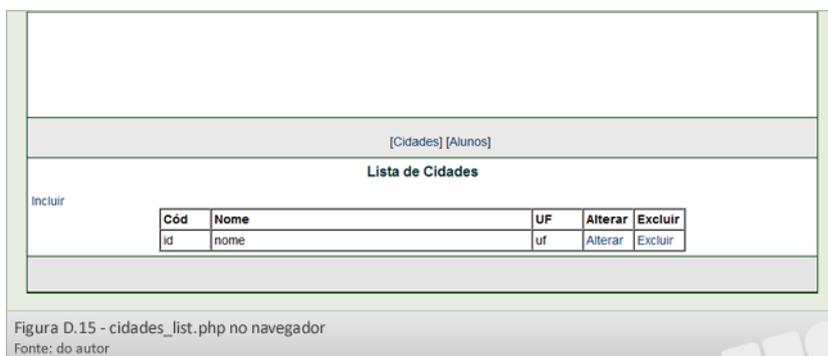


Figura D.15 - cidades_list.php no navegador
Fonte: do autor

Na figura D.16 pode ser analisado o código (X)HTML do formulário (cidades_form.php). Alguns apontamentos sobre o (X)HTML:

Linha 15 – O *action* do *form* faz referência ao *cidades.php*, ou seja, quando submeter o form (clicar em gravar) chamará o *cidades.php*;

Linha 20 – *Readonly* indica que a caixa de texto é apenas leitura. O usuário não pode editar, isso porque o campo *id* de cidades é gerado automaticamente;

Linha 29 – Possui um elemento *select* chamado *uf* sem *options*. Depois acrescentaremos as opções.

Linha 34 e 35- Quando clicar no botão cancelar altera a *location* para *cidades.php* (chama o *cidades.php*) ;

Linha 36 – Tem um campo oculto chamado *acao*. Logo veremos porque este campo é necessário.

É muito importante saber o nome dos objetos criados no *form*, principalmente: *id*, *nome*, *uf*, *acao*. Vamos precisar depois.

```

14 <div id="conteudo">
15 <form id="form1" name="form1" method="post" action="cidades.php">
16 <div id="titulo">Cadastro de Cidades</div> <br/>
17 <table width="600" border="0" align="center" cellpadding="5" cellspacing="0">
18 <tr>
19 <td align="right"><label for="id">Código:</label></td>
20 <td><input name="id" type="text" id="id" value="" size="10" readonly="readonly" />
21 </td>
22 </tr>
23 <tr>
24 <td align="right"><label for="nome">Nome:</label></td>
25 <td><input name="nome" type="text" id="nome" value="" size="50" maxlength="50" /></td>
26 </tr>
27 <tr>
28 <td align="right"><label for="uf">UF:</label></td>
29 <td><select name="uf" id="uf"> </select></td>
30 </tr>
31 <tr>
32 <td align="right">&nbsp;</td>
33 <td><input type="submit" name="botao1" id="botao1" value="Gravar" />
34 <input type="button" name="botao2" id="botao2" value="Cancelar"
35 onclick="location='cidades.php' " />
36 <input name="acao" type="hidden" id="acao" value="" /></td>
37 </tr>
38 </table>
39 </form>
40 </div>
    
```

Figura D.16 - Código do cidades_form.php
Fonte: do autor



Agora vamos para a programação PHP que vai integrar os arquivos da listagem e formulário. O código referente ao gerenciador da manutenção, arquivo *cidades.php* é mostrados na figura D.18.

```

1 <?php
2 include('conexao.php');
3 include('funcoes.php');
4
5 $acao = $_REQUEST['acao'];
6 $redireciona = false;
7
8 if ($acao == ""){
9     $sql = "select * from cidades order by nome";
10    $rs = $con->Execute($sql);
11    include('cidades_list.php');
12 }
13
14 if ($acao == "incluir"){
15     include('cidades_form.php');
16 }
17
18 if ($acao == "alterar"){
19     $id = $_REQUEST['id'];
20     $sql = "select * from cidades where id = $id";
21     $rs = $con->Execute($sql);
22     include('cidades_form.php');
23 }
24 if ($acao == 'excluir'){
25     $id = $_REQUEST['id'];
26     $sql = "delete from cidades where id = $id";
27     $rs = $con->Execute($sql);
28     $redireciona = true;
29 }
30 if ($acao == "gravar_incluir"){
31     $nome = $_POST['nome'];
32     $uf = $_POST['uf'];
33     $sql = "insert into cidades (nome, uf) values ('$nome', '$uf') ";
34     $rs = $con->Execute($sql);
35     if (!$rs) // se ocorreu erro no sql...
36         $mensagem = "Erro ao incluir dados";
37     $redireciona = true;
38 }
39 if ($acao == "gravar_alterar"){
40     $id = $_POST['id'];
41     $nome = $_POST['nome'];
42     $uf = $_POST['uf'];
43     $sql = "update cidades set nome = '$nome', uf = '$uf' where id = $id ";
44     $rs = $con->Execute($sql);
45     if (!$rs)
46         $mensagem = "Erro ao alterar dados";
47     $redireciona = true;
48 }
49
50 if ($mensagem > ''){
51     echo "<script> alert('$mensagem');</script>";
52 }
53
54 if ($redireciona == true){
55     echo "<script> location='cidades.php' </script> ";
56 }
57
58 ?>

```

Figura D.18 - arquivo cidades.php
Fonte: do autor

Vamos analisar todo código:

Linha 5 – O `$_REQUEST[]` é um *array* super global do PHP que pode ser usado para pegar dados que sejam recebidos tanto pelo método GET quanto pelo POST. *\$acao* será o parâmetro utilizado para verificar a ação que o usuário selecionou, que pode ser incluir, alterar, excluir, entre outras. O valor para *acao* pode ser passado tanto por GET como POST, por isso estamos usando o `$_REQUEST`;

Linha 6 – A variável *\$redireciona* quando possuir valor *true* vai redirecionar para a listagem das cidades. Inicialmente possui valor *false*;

Linha 8 – Quando a *\$acao* não possuir um valor, vamos sempre mostrar a listagem de cidades. Isso vai ocorrer quando for chamado diretamente *idades.php* sem passar nenhum parâmetro. Neste caso, executa o SQL para selecionar os registros de cidades em ordem de nome e faz o include do *idades_list.php* que vai listar os dados;

Linhas 14 e 15 – Quando a *\$acao* for incluir faz o include do *idades_form.php* que vem em branco.

Linhas 18 até 23 – Quando a *\$acao* for alterar, vai fazer a alteração de uma cidade cujo *id* (identificador) será recebido por parâmetro. A linha 19 pega o valor de *id*, para então na linha 20 montar o SQL que busca o registro da cidade e executa na linha 21. Após faz o include de *idades_form.php* que mostrará o formulário com os dados da cidade.

Linhas 24 até 29 - Quando a *\$acao* for *excluir*, vai fazer a exclusão de uma cidade cujo *id* (identificador) será recebido por parâmetro. A linha 25 pega o valor de *id*, para então na linha 26 montar o SQL que busca o registro da cidade e executa na linha 27. Na linha 28 altera o valor da variável *\$redireciona* para *true* (isto vai fazer voltar para a listagem atualizada).

Linhas 30 até 38 - Quando a *\$acao* for *gravar_incluir*, vai fazer a inclusão de uma cidade cujos dados vieram do formulário. Primeiro uma questão, de onde vem o valor *gravar_incluir*? Deverá vir do formulário quando for uma inclusão. Veremos como fazer isso quando incluirmos programação PHP no *idades_form.php*. Lembra que destacamos a importância de saber o nome dos objetos do formulário? Agora vamos precisar deles. Nas linhas 31 e 32 estamos recuperando a informação do nome e da UF da cidade que foram informados no formulário (usa método POST). Após montamos um SQL de inserção usando estes valores. Outra forma seria fazer diretamente o uso do *\$_POST* no SQL, desta forma:

```
$sql = "insert into cidades (nome, uf) values
      ('$_POST[nome]', '$_POST[uf]') ";
```

Fique a vontade para usar a forma que achar melhor. Na linha 34 o SQL é executado atribuindo o resultado para *\$rs*. Na linha 35 testa se ocorreu erro e na linha 36 atribui uma mensagem de erro para variável *\$mensagem* (se for o caso). Na linha 37 altera o valor da variável *\$redireciona* para *true* (isto vai fazer voltar para a listagem atualizada).

Linhas 39 até 48 - Quando a *\$acao* for *gravar_alterar*, vai fazer a alteração de uma cidade cujos dados vieram do formulário. De onde vem o valor *gravar_alterar*? Deverá vir do formulário quando for uma alteração. Veremos como fazer isso quando incluirmos programação PHP no *idades_form.php*. Nas linhas 40, 41 e 42 estamos recuperando a informação *id*, *nome* e *UF* da cidade do formulário (usa método POST). Após montamos um SQL de alteração usando estes valores. Na linha 44 o SQL é executado atribuindo o resultado para *\$rs*. Na linha 45 testa se ocorreu erro e na linha 46 atribui uma mensagem de erro para variável *\$mensagem* (se for o caso). Na linha 47 altera o valor da variável *\$redireciona* para *true* (isto vai fazer voltar para a listagem atualizada).

Linhas 50 até 52 – Se a variável *\$mensagem* possuir algum valor, faz mostrar a mensagem usando JavaScript.

Linhas 54 até 56 – Se a variável *\$redireciona* possuir valor *true*, usa JavaScript para redirecionar para *idades.php*(vai mostrar a listagem).

Nosso *idades.php* está pronto. Agora precisamos retornar ao *idades_list.php* e ao *idades_form.php* para incluir a programação PHP. Com esta estrutura que trabalhamos, teremos pouca codificação PHP neste dois arquivos. A maior parte da codificação é no *idades.php*. Isso é interessante, pois facilita a manutenção posterior do código. Vamos ver agora o código completo dos arquivos *idades_list.php* e *idades_form.php* e alguns comentários.

idades_list.php

A figura D.19 apresenta o *idades_list.php* com a inclusão da programação PHP. Vamos comentar agora o que programamos neste arquivo:

Para mostrar todas as cidades retornadas a partir do SQL executado na linha 10 do *idades.php*, precisamos de um laço (*while*). Entre as linhas 26 de 31 foi incluído um código PHP com o início de laço *while* para percorrer todos resultado. Nas linhas 28, 29 e 30 recuperamos os dados que vamos mostrar, atribuindo para variáveis. É importante observar que a segunda linha da tabela fica dentro do laço, ou seja, será repetida para cada cidade. Na linha 41 tem o código para avançar para o próximo registro e o laço é fechado na linha 42;

Para mostrar os dados nas colunas da tabela, foram mostradas as variáveis nas linhas 33, 34 e 35;

Para o processo de alteração e exclusão precisamos saber o *id* (identificador/chave) do registro. Por isso, nos links de alterar e excluir precisamos passar esta informação. A adição de código nas linhas 36 e 38 tem este objetivo. Acrescentamos no link mais um parâmetro (& para indicar que vem outro parâmetro) cujo nome é *id* e usando PHP mostramos o valor da variável *\$id*.

```

15 <div id="conteudo">
16 <div id="titulo">Lista de Cidades</div>
17 <a href="idades.php?acao=incluir">Incluir</a><br />
18 <table width="600" border="1" align="center" cellpadding="3" cellspacing="0">
19 <tr>
20 <th width="50" scope="col">Cód</th>
21 <th scope="col">Nome</th>
22 <th width="50" scope="col">UF</th>
23 <th width="50" scope="col">Alterar</th>
24 <th width="50" scope="col">Excluir</th>
25 </tr>
26 <?php
27 while (!$rs->EOF) { //enquanto não for o fim do resultado fica no laço
28     $id = $rs->fields['id'];
29     $nome = $rs->fields['nome'];
30     $uf = $rs->fields['uf'];
31     ?>
32 <tr>
33 <td><?php echo $id;?></td>
34 <td><?php echo $nome;?></td>
35 <td><?php echo $uf;?></td>
36 <td><a href="idades.php?acao=alterar&id=<?php echo $id;?>">Alterar</a></td>
37 <td><a href="javascript: if (confirm('Confirma exclusão?')){
38     location='idades.php?acao=excluir&id=<?php echo $id;?>'}">Excluir</a></td>
39 </tr>
40 <?php
41     $rs->MoveNext(); //mover para o próxima cidade
42 } //fim do laço
43 ?>
44 </table>
45 </div>

```

Figura D.19 - código do *idades_list.php* com PHP
Fonte: do autor

idades_form.php

A figura D.20 apresenta o *idades_form.php* com a inclusão da programação PHP. O que basicamente foi alterado:

Inserção de código PHP no atributo *value* para mostrar os dados que vem do banco de dados (vai mostrar quando o formulário é chamado para uma alteração de cidade). Nas linhas 21 e 27 utiliza a variável *\$rs* do resultado do SQL (criado na linha 21 do *idades.php*) e pega o valor do campo usando *fields*;

Para mostrar a lista de estados no menu de lista, estamos incluindo na linha 33 a chamada de uma função denominada *lista_uf()* que está definida no arquivo *funcoes.php* (foi incluído no *idades.php*). Esta

função recebe por parâmetro a UF da cidade. Este parâmetro serve para trazer selecionada a UF da cidade quando for um processo de alteração. A seguir vamos ver em detalhes a função *lista_uf()*;

Lembra que comentamos sobre os valores *gravar_incluir* e *gravar_alterar* do *\$acao*? Quando o formulário for submetido enviará um valor para o campo oculto *acao* que está na linha 41. Atribuímos para o *valeu dele* o seguinte:

```
gravar_<?php echo $acao;?>
```

Isto significa que quando o formulário for chamado a partir de um valor de *\$acao* igual a *incluir*, o campo oculto ficará valendo *gravar_incluir*. Se for chamado a partir de um valor de *\$acao* igual a *alterar*, o campo oculto ficará valendo *gravar_alterar*. Com isso, podemos saber no *idades.php* se é necessário fazer um processo de inclusão ou alteração.

```

15 <div id="conteudo">
16 <form id="form1" name="form1" method="post" action="idades.php">
17 <div id="titulo">Cadastro de Cidades</div> <br/>
18 <table width="600" border="0" align="center" cellpadding="5" cellspacing="0">
19 <tr>
20 <td align="right"><label for="id">Código:</label></td>
21 <td><input name="id" type="text" id="id" value="<?php echo $rs->fields['id'];?>"
22 size="10" readonly="readonly" />
23 </td>
24 </tr>
25 <tr>
26 <td align="right"><label for="nome">Nome:</label></td>
27 <td><input name="nome" type="text" id="nome" value="<?php echo $rs->fields['nome'];?>"
28 size="50" maxlength="50" /></td>
29 </tr>
30 <tr>
31 <td align="right"><label for="uf">UF:</label></td>
32 <td><select name="uf" id="uf">
33 <?php echo lista_uf($rs->fields['uf']); ?>
34 </select></td>
35 </tr>
36 <tr>
37 <td align="right">&nbsp;</td>
38 <td><input type="submit" name="botaol" id="botaol" value="Gravar" />
39 <input type="button" name="botaos" id="botaos"
40 value="Cancelar" onclick="location='idades.php' " />
41 <input name="acao" type="hidden" id="acao" value="gravar_<?php echo $acao;?>" /></td>
42 </tr>
43 </table>
44 </form>
45 </div> |

```

Figura D.20 - código do *idades_form.php* com PHP
Fonte: do autor

Função *lista_uf()*

Como visto acima, usamos a função *lista_uf()* para mostrar a lista de unidades federativas no *idades_form.php*. Agora vamos dar uma olhada com mais detalhes nesta função que está definida no *funcoes.php*. A figura D.21 mostra o código da função. Uma primeira observação importante a fazer é sobre como é montada uma lista de opções para um menu de lista. O código de exemplo abaixo apresenta cinco opções, sendo que a primeira não possui valor, fica em branco. Cada opção está em um elemento *option* que por sua vez possui um atributo *value* para o valor correspondente ao que está sendo mostrado, no caso da UF tem o mesmo valor. Mas observe que uma *option* possui atributo *selected*. Isto significa que o elemento que possuir este atributo virá com este valor selecionado para o menu de lista. Neste exemplo o menu de lista viria com a opção AM selecionada. No caso do nosso exemplo este atributo será importante para mostrar a UF correspondente à cidade que está sendo alterada.

```

<option></option>
<option value='AC'>AC</option>
<option value='AL'>AL</option>
<option value='AM' selected>AM</option>
<option value='AP'>AP</option>

```

Alguns apontamentos:

Linha 4 – Define um *array* com todas as unidades federativas;

linha 7 – Ordena o vetor em ordem alfabética;

linha 8 – Inicializa a variável *\$lista* com a primeira opção em branco;

linha 9 - Faz um *foreach* para percorrer todo o *array*. *\$sigla* recebe cada vez um dos valores do vetor;

linha 10 – Cria a variável *\$s* sendo uma string sem valor;

linhas 11 e 12 – Verifica se o valor do vetor (*\$sigla*) é igual a *\$uf* recebida por parâmetro (UF da cidade a ser alterada). Se for igual atribui “*selected*” para variável *\$s*. Isso fará com que esta UF seja a selecionada.

Linha 13 – Monta a opção e concatena na variável *\$lista*;

Linha 14- Fim do *foreach*;

Linha 15- Retorna as opções montadas (variável *\$lista*).

```

3 function lista_uf($uf){
4     $ufs = array('RS','SC','PR','SP','RJ',
5                 'AM','AC','AL','AP','BA','CE','DF','ES','GO','MA',
6                 'MG','MS','MT','PA','PB','PE','PI','RN','RO','RR','SE','TO');
7     sort($ufs); //ordena o array
8     $lista = "<option></option>";
9     foreach($ufs as $sigla){
10        $s = "";
11        if($sigla == $uf)
12            $s = "selected";
13        $lista .= "<option value='$sigla' $s >$sigla</option>";
14    }
15    return $lista;
16 }

```

Figura D.21 - código da função lista_uf()
Fonte: do autor

Mostrar dados na Área Pública

Só falta vermos como mostrar os dados de cidades na Área Pública do site. Vamos criar um arquivo chamado *idades.php* na pasta *c:\xampp\htdocs\siteBD*. A figura D.22 apresenta o código do arquivo *idades.php*. Não temos nada de novo aqui, estamos usando tudo o que já vimos anteriormente.

Dica:

Na linha 21 do *idades.php* faz o include do *conexao.php*. Observe que o caminho muda, pois está buscando na pasta *admin*.

```

13 <div id="conteudo">
14 <div id="titulo">Lista de Cidades</div> <br />
15 <table width="600" border="1" align="center" cellpadding="3" cellspacing="0">
16 <tr>
17 <th scope="col">Nome</th>
18 <th width="50" scope="col">UF</th>
19 </tr>
20 <?php
21 include("admin/conexao.php");
22 $sql = "select * from cidades order by nome";
23 $rs = $con->Execute($sql);
24 while(!$rs->EOF){ //enquanto não for o fim do resultado fica no laço
25     $nome = $rs->fields['nome'];
26     $uf = $rs->fields['uf'];
27 }
28 <tr>
29 <td><?php echo $nome;?></td>
30 <td><?php echo $uf;?></td>
31 </tr>
32 <?php
33 $rs->MoveNext(); //mover para o próxima cidade
34 } //fim do laço
35 }
36 </table>
37 </div> |

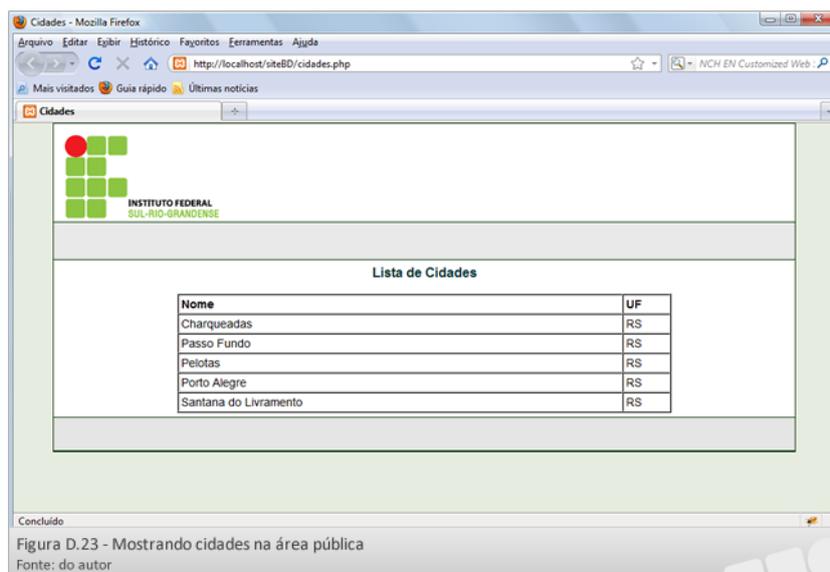
```

Figura D.22 - Código do arquivo cidades.php da área pública
Fonte: do autor

Atenção:

A estrutura do arquivo, na parte (X)HTML, é a mesma do *index.php* da área administrativa, apenas retiramos o menu. Neste momento não nos preocupamos tanto com o aspecto visual, haja vista que nosso propósito é a programação PHP.

A figura D.23 mostra a visualização do arquivo *idades.php* da Área Pública no navegador.

**Tratamento chave estrangeira**

Outro recurso importante é o tratamento de chave estrangeira. Para explicitar vamos ver a estrutura da tabela *alunos* que criamos no início da Unidade D. Veja abaixo que a tabela possui um campo chamado *cidade* que é do tipo *int(11)*. O valor do campo *cidade* faz referência a chave primária da tabela *idades* (campo *id*).

```
CREATE TABLE IF NOT EXISTS alunos (
    id bigint NOT NULL auto_increment,
    nome varchar(100) NOT NULL,
    email varchar(100) NOT NULL,
    fone varchar(15) NOT NULL,
    nascimento date NOT NULL,
    cidade int(11) NOT NULL,
    PRIMARY KEY (id)
);
```

Pensando no formulário de alunos, podemos fazer a seleção da cidade conforme mostra a figura D.24, buscando os dados da tabela *idades* e mostrando em um menu de lista.

O menu de lista seria montado com as opções da forma apresentada abaixo, onde mostra o nome da cidade, mas o valor correspondente ao nome da cidade é o identificador (*id*):

```
<select name='cidade'>
<option></option>
<option value='5'>Charqueadas</option>
<option value='2' selected >Passo Fundo</option>
<option value='3' >Pelotas</option>
<option value='4' >Porto Alegre</option>
<option value='6'>Santana do Livramento </option>
</select>
```

Para montar este menu de lista, a biblioteca ADODB possui um método chamado *GetMenu*. Por isso, no arquivo *funcoes.php* criamos a função *lista_cidades()* como exemplo para tratar chave estrangeira com o *GetMenu*. Vamos dar uma olhada no código conforme figura D.25:

Linha 19 – Definição da função *lista_cidades()* que recebe dois parâmetros: a instância da conexão e o nome da cidade que deve ser mostrado no menu de lista por padrão;

Linha 21 – Monta o SQL para buscar os dados da cidade. Importante a ordem dos campos no *select*, deve sempre primeiro vir a descrição e depois o campo que é chave primária;

Linha 22 – Executa o SQL e atribui o resultado para *\$rs*;

Linha 23 – Testa se chegou um valor no parâmetro *\$nome*. Se chegou valor significa que é uma alteração, senão uma inclusão. O menu de lista é montado de forma diferente para cada situação.

Linha 24 – Retorna o menu montado. O menu se chamará cidade, virá selecionado por padrão a cidade cujo nome é igual ao valor de *\$nome*, e não possui um primeiro valor do menu em branco;

Linha 26 – Retorna o menu montado. O menu se chamará cidade, virá selecionado com a primeira opção em branco;

```

19 function lista_cidades($con, $nome){
20     //no sql deve sempre vir primeiro o nome e depois o código (id)
21     $sql = "select nome, id from cidades order by nome";
22     $rs = $con->Execute ($sql);
23     if ($nome > "")
24         return $rs->GetMenu ('cidade', $nome, false); //alterar
25     else
26         return $rs->GetMenu('cidade', null, true); //incluir
27 }
28
29 GetMenu:
30 1° parametro: nome do menu de lista
31 2° parametro: valor para mostrar na caixa
32 3° parametro: true para mostrar a primeira opção vazia e false
33 para mostrar um valor (do 2° parametro)
34 */
35 }

```

Figura D.25 - Código da função lista_cidades()
Fonte: do autor

Atenção:

Note que o método *GetMenu* monta toda a estrutura do menu de lista, o *select* e as *option*

Abaixo segue um exemplo de chamada da função no formulário dentro da coluna que se deseja mostrar o menu de lista:

```
<td><?php echo lista_cidades($con, $rs->fields['cidade_nome']); ?></td>
```

Tratamento de datas e valores

Para campos do tipo *float*, *decimal* e *double*, o banco de dados Mysql vai retornar o ponto (.) como separador decimal. Para formatar um valor podemos utilizar a função *number_format* do PHP. Por exemplo, considere que a variável *\$valor* possui um valor retornado do banco de dados:

```
$fvalor = number_format($valor, 2, ',', '.');
```

Os parâmetros na sequência são:

- valor a ser formatado
- número de casas decimais
- caractere para separador decimal
- caractere para separador de milhar

Também precisamos nos preocupar no formato para gravar um valor no banco. Considere que usuário tenha informado o valor 4.560,80, para gravar no banco o formato correto é 4560.80. Para fazer esta conversão podemos usar a função abaixo que pode ser definida no *funcoes.php* e ser chamada sempre que necessário:

```

function fvalor_banco($val){
    $val = str_replace(".", "", $val);
    $val = str_replace(",", ".", $val);
    return $val;
}

```

Para chamar a função:

```
$valor = fvalor_banco($valor);
```

Atenção:

O primeiro *str_replace* da função retira o ponto. O segundo *str_replace* troca vírgula por ponto.

O MySQL trabalha com datas no formato ano/mês/dia. Já o nosso formato é dia/mês/ano. Por isso, sempre que pegarmos uma data do banco precisamos converter para nosso formato e quando enviarmos uma data para o banco devemos tratar para ser no formato aceito pelo MySQL.

A função abaixo recebe uma data no formato a/m/d e transforma para o formato d/m/a.

```
function fdata($dt){
    $data = explode('-', $dt);
    return "$data[2]/$data[1]/$data[0]";
}
```

A função abaixo recebe uma data no formato d/m/a e transforma para o formato a/m/d.

```
function fdata_banco($dt){
    $data = explode('/', $dt);
    return "$data[2]/$data[1]/$data[0]";
}
```

Dica:

Inclua estas duas funções no arquivo *funcoes.php*.

Síntese

Bem, esta parte da Unidade D mostrou em detalhes a construção de uma manutenção completa usando a linguagem PHP com acesso a banco de dados. Agora é importante que você pratique, por isso faça a atividade proposta e anote todas as suas dúvidas para posteriormente discutir tais questões no fórum. Bom trabalho!

Atividades - Parte 2

1. Considerando a tabela *alunos* criado no banco de dados *curso* faça a manutenção completa para a tabela (listar, incluir, alterar e excluir).

- utilize a estrutura do site *siteBD*
- deve usar o padrão de manutenção apresentado na disciplina
- observar que há um campo de chave estrangeira (usar o modelo da função *lista_cidades()*)
- fazer a validação do formulário (campos obrigatórios e validação de data) usando JavaScript

Após a realização das atividades participe do fórum de discussão proposto pelo professor formador.

Autenticação de usuário para área administrativa

Nesta segunda parte da unidade D vamos mostrar como construir um mecanismo de autenticação de usuário para acesso a área administrativa.

Controle de acesso

O processo de autenticação de usuário para acesso a área administrativa é extremamente importante, uma vez que não podemos deixar esta área do site vulnerável para acesso de usuários sem permissão. Imagine qualquer usuário tendo acesso aos recursos de incluir, alterar e excluir dados da base de dados do site. Por isso, é importante que somente usuários registrados tenham acesso. Inicialmente o usuário deve informar seu *login* e senha em uma tela com mostra a figura D.26.

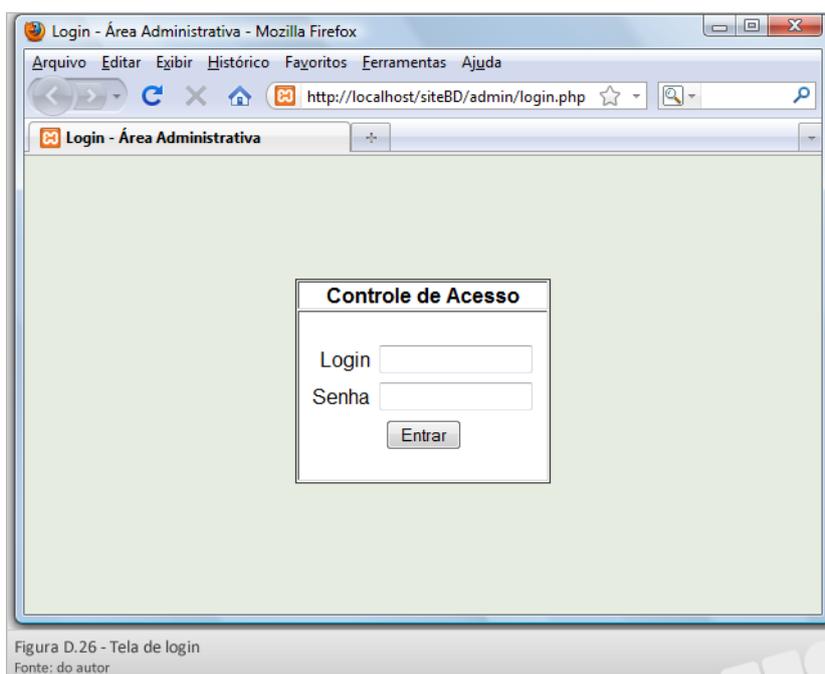


Figura D.26 - Tela de login
Fonte: do autor

Após informar *login* e senha válidos, o usuário tem acesso a área administrativa (como a figura D.27). Em cada acesso de página da área administrativa é verificado se o usuário se autenticou. Veja na figura que foi mostrada uma saudação ao usuário: “Olá Daniel!”. Isso porque o *login* do usuário que está logado fica na sessão.

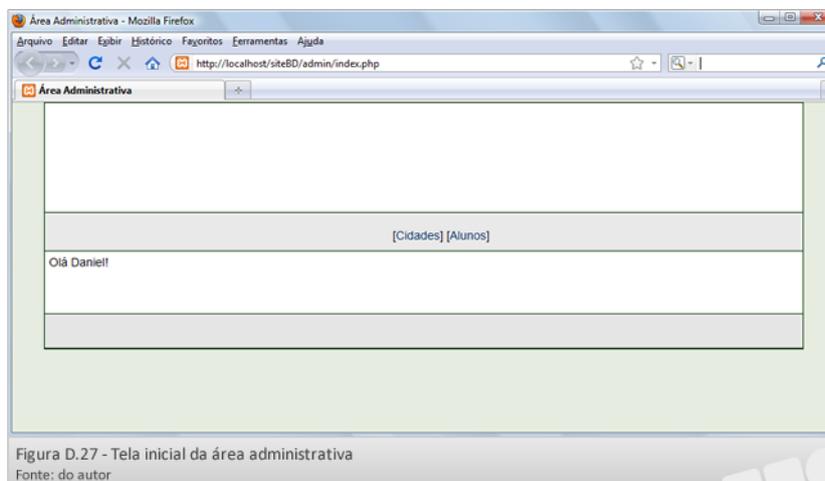


Figura D.27 - Tela inicial da área administrativa
Fonte: do autor

Agora vamos ver passo a passo o que precisamos para fazer a autenticação do usuário e controlar a sessão para a área administrativa.

Tabela usuários

Primeiro precisamos ter no nosso banco de dados (*curso*) uma tabela para registrar os usuários que tem acesso a área administrativa. O script abaixo deve ser usado para criar a tabela *usuarios*.

```
CREATE TABLE usuarios (
    login VARCHAR( 20 ) NOT NULL ,
    nome VARCHAR( 50 ) NOT NULL ,
    senha VARCHAR( 35 ) NOT NULL ,
    PRIMARY KEY ( login ) );
```

A tabela contém apenas três campos, sendo o *login* a chave primária. Além disso, temos o nome do usuário e a senha. Note que o campo senha foi criado com tamanho 35, isso porque a senha será criptografada.

No exemplo que vamos usar, todos os usuário vão possuir o mesmo tipo de acesso. Mas existem sistemas que podem necessitar de níveis diferentes de acesso. Neste caso deverá ser projetada uma estrutura diferente, de forma a identificar qual o nível de acesso do usuário.

Não vamos criar a manutenção de usuários neste material, pois nosso objetivo agora é fazer a autenticação do usuário. Por isso, para incluir dados na tabela crie um arquivo nomeado de *inserir_usuario.php*. A figura D.28 mostra o código do *inserir_usuario.php*.

Só precisamos chamar a atenção de uma função usada neste código. É a função **md5** da linha 5. A função **md5** faz a criptografia do valor passado por parâmetro usando um algoritmo unidirecional, isto significa que não existe o processo de descriptografar. Veremos depois como vamos testar a senha. A função gera uma string de 32 caracteres.

Ao executar o *inserir_usuario.php* será criado um usuário com os seguintes dados:

- Login = root
- Nome = Administrador
- Senha = e8d95a51f3af4a3b134bf6bb680a213a

Atenção:

Lembre-se que e8d95a51f3af4a3b134bf6bb680a213a é a criptografia de “senha”.

```
1 <?php
2
3     include('conexao.php');
4
5     $senha = md5('senha'); //e8d95a51f3af4a3b134bf6bb680a213a
6
7     $sql = "insert into usuarios (login, nome, senha) values
8         ('root','Administrador', '$senha)";
9
10    $rs = $con->Execute($sql);
11
12    if (!$rs) {
13        echo $con->ErrorMsg();
14    }else
15        echo "Dados inseridos";
16
17    ?>
```

Figura D.28 - Código do *inserir_usuario.php*
Fonte: do autor

Dica:

Lembre-se dos dados do usuário para posteriormente fazer o *login*. Você também pode visualizar estes dados pelo *phpMyAdmin* <<http://localhost/phpMyAdmin>>.

Tela Login

Para tela de *login* crie o arquivo *login.php* na pasta *admin* (c:\xampp\htdocs\siteBD\admin). O código da tela de *login* é mostrado na figura D.29. A parte de código apresentada está no elemento *body*. Destaque para:

- No *action* do *form* (linha 18) chama *index.php*, que a página inicial da área administrativa. Como o botão Entrar é do tipo *submit*, quando for acionado chamará o *index.php*;
- Possui os campos *login* e senha;
- O campo senha é do tipo *password*.

```

11 <table width="190" border="1" align="center" cellpadding="0" cellspacing="1"
12 bordercolor="#333333" bgcolor="#FFFFFF" >
13 <tr>
14 <td height="19" align="center"><strong>Controle de Acesso</strong></td>
15 </tr>
16 <tr>
17 <td height="109">
18 <form action="index.php" method="POST" name="frmlogin" id="frmlogin"> <br>
19 <table width="192" border="0" align="center" cellpadding="0" cellspacing="7">
20 <tr>
21 <td width="28%" height="21" align="right"> Login </td>
22 <td width="72%"> <input name="login" type="text" id="login" size="15" maxlength="30"> </td>
23 </tr>
24 <tr>
25 <td align="right"> Senha</td>
26 <td> <input name="senha" type="password" id="senha" size="15" maxlength="30"> </td>
27 </tr>
28 <tr align="center" >
29 <td colspan="2"> <input name="btnEntrar" type="submit" id="btnEntrar" value="Entrar" > </td>
30 </tr>
31 </table>
32 </form>
33 </td>
34 </tr>
35 </table>

```

Figura D.29 - código do login.php

Fonte: do autor

Controle de acesso

Agora que já temos uma tabela *usuarios* e a tela de *login*, vamos fazer o controle de acesso a área administrativa. Este controle será feito pelo arquivo *controle_acesso.php*. Importante: para controlar se o usuário está logado vamos criar uma variável na sessão chamada *ses_usu_login* que conterà o *login* do usuário. A figura D.30 mostra o código completo do *controle_acesso.php*. Vamos analisá-lo:

Linha 2 – Inicia a sessão;

Linha 3 – Verifica se tem a variável *ses_usu_login* na sessão, se não possuir entra neste if ;

Linha 5 – Para verificar se veio da tela de *login*, testa se existe e se há um valor para *login* no array *\$_REQUEST[]*, se existir entra neste IF;

Linha 6 – Atribui a senha que recebeu da tela de *login* para a variável *\$senha*, já criptografando. Por que a criptografia? Porque a senha do banco está criptografada e não há mecanismo para reverter. Por isso, vamos testar as duas criptografadas;

Linhas 7 e 8 – Monta o SQL para consultar no banco se existe usuário com o login e senha informados;

Linha 9 – Executa o SQL e atribui o resultado para *\$rs*;

Linhas 11 e 12 – Verifica novamente (pode não ter retornado registro) se o registro retornado possui o login igual ao informado. Se *true* cria a variável *ses_usu_login* na sessão atribuindo o *login*.

Linhas 14 até 16 – Se não encontrar usuário no banco, faz include do arquivo que mostra mensagem de *login* inválido (*logininvalido.php*) e executa a função *exit* (para a execução);

Linhas 19 até 21 – Se não veio da tela de *login*, faz include do arquivo que mostra mensagem de acesso negado (*acessonegado.php*) e executa a função *exit* (para a execução);

Linha 24 – Entra no *else* se possuir na sessão a variável *ses_usu_login*

Linhas 25 até 29 – Se a variável da sessão não possuir valor, ou seja não tem um *login*, retira a variável da sessão, faz include do arquivo que mostra mensagem de acesso negado (*acessonegado.php*) e executa a função *exit* (para a execução);

```

1 <?php
2 session_start();
3 if(session_is_registered("ses_usu_login") == false ){
4
5     if (isset($_REQUEST["login"]) and $_REQUEST['login'] != ""){
6         $senha = md5($_REQUEST['senha']); //md5
7         $sql = "select * from usuarios where
8             login = '$_REQUEST[login]' and senha = '$senha'";
9         $rs = $con->Execute($sql);
10
11         if ($rs->fields['login'] == $_REQUEST['login']) {
12             $_SESSION['ses_usu_login'] = $_REQUEST['login'];
13         }
14         else{
15             include("logininvalido.php");
16             exit;
17         }
18     }
19     else{
20         include("acessonegado.php");
21         exit;
22     }
23 }
24 else {
25     if ($_SESSION['ses_usu_login'] <= ""){
26         session_unregister("ses_usu_login");
27         include("acessonegado.php");
28         exit;
29     }
30 }
31 ?>

```

Figura D.30 - código do controle_acesso.php
Fonte: do autor

Agora resta saber como vamos usar o *controle_acesso.php*. Ele servirá para controlar o acesso a todas as páginas da área administrativa. Por isso, no início de cada arquivo, que é chamado diretamente, faremos um include do *controle_acesso.php* (*index.php*, *idades.php*, *alunos.php*). O include do *controle_acesso.php* será sempre após o include do *conexao.php*, pois ele precisa da conexão com o banco de dados.

Veja como fica no *index.php* (figura D.31). Na linha 19 mostra uma saudação ao usuário logado.

Dica:

Lembre-se que o *session_start* deve vir antes de qualquer escrita do documento (X(HTML)). Por isso o include fica no início do arquivo.>

```

1 <?php
2     include ("conexao.php");
3     include ("controle_acesso.php");
4 >?>
5 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
6 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
7 <html xmlns="http://www.w3.org/1999/xhtml">
8 <head>
9 <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
10 <title>Área Administrativa</title>
11 <link href="../estilo.css" rel="stylesheet" type="text/css" />
12 </head>
13 <body>
14 <div id="geral">
15     <div id="topo"> </div>
16     <div id="menu">
17         <?php include ('menusup.php');?>
18     </div>
19     <div id="conteudo">
20     <div>Olá <?php echo $_SESSION['ses_usu_login']. "!";?></div>
21     <br/>
22     <br/>
23     </div>
24 <div id="rodape"></div>
25 </div>
26 </body>
27 </html>

```

Figura D.31 - index.php com o uso do controle_acesso.php
Fonte: do autor

A figura D.32 mostra como fica o arquivo *idades.php* com o include do *controle_acesso.php*. O include foi feito na linha 3.

```

1 <?php
2     include ('conexao.php'); // incluir arquivo de conexão
3     include ('controle_acesso.php');
4     include ('funcoes.php'); // incluir arquivo de funções
5
6     $acao = $_REQUEST['acao'];
7     $redireciona = false;

```

Figura D.32 - início do *idades.php* com o include do *controle_acesso.php*
Fonte: do autor

Após a inclusão do *controle_acesso.php* nos arquivos, vamos fazer um teste. Antes de tentar fazer o *login*, tente acessar diretamente o *index.php*:

```
http://localhost/admin/index.php
```

Se tudo estiver certo, deverá mostrar a mensagem "Acesso Negado!". Isso ocorre porque o *controle_acesso.php* verificou que não tem usuário na logado na sessão. Faça outros testes agora e veja o que acontece:

Acesse também diretamente <http://localhost/admin/cidades.php>

Na tela de *login* informe usuário e senha inválidos

Faça *login* e agora tente acessar <http://localhost/admin/cidades.php>

Logout

Para sair da área administrativa pode ser criado o arquivo *logout.php* com o código para destruir a sessão. O código do *logout.php* é mostrado na figura D.33.

```

1  <?php
2
3      session_start();
4      session_destroy();
5
6  ?>
7  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
8  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
9  <html xmlns="http://www.w3.org/1999/xhtml">
10 <head>
11     <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
12     <title>Área Administrativa</title>
13     <link href="../estilo.css" rel="stylesheet" type="text/css" />
14 </head>
15 <body>
16     <div align="center">Sessão finalizada<br />
17     <a href="login.php">Voltar para login </a><br />
18 </div>
19 </body>
20 </html>

```

Figura D.33 - Código do *logout.php*
Fonte: do autor

Síntese

Esta parte da Unidade D mostrou como fazer a autenticação de usuário para acesso à área administrativa e como usar sessão para este controle. Agora é com você! Faça a atividade proposta e anote todas as suas dúvidas para posteriormente discutir tais questões no fórum. Bom trabalho!

Atividades - Parte 3

1. Considerando a tabela *usuarios* do *script* abaixo, faça a manutenção completa para a tabela (listar, incluir, alterar e excluir).

- utilize a estrutura do site *siteBD*
- deve usar o padrão de manutenção apresentado na disciplina
- usar mecanismo de criptografia da senha
- no formulário de entrada de dados insira uma caixa de texto para confirmação da senha (terá os campos senha e confirma senha)
- fazer a validação do formulário (campos obrigatórios e confirmação de senha) usando JavaScript
- use o *controle_acesso.php* para verificar se o usuário está logado.

```

CREATE TABLE usuarios (
    login VARCHAR( 20 ) NOT NULL ,
    nome VARCHAR( 50 ) NOT NULL ,
    senha VARCHAR( 35 ) NOT NULL ,
    PRIMARY KEY ( login) );

```

Após a realização das atividades participe do fórum de discussão proposto pelo professor formador.