

Aplicativos para Web II

William Geraldo Sallum

Curso Técnico em Planejamento e
Gestão em Tecnologia da Informação



Aplicativos para Web II

William Geraldo Sallum



CEFET-MG

Belo Horizonte-MG
2012

Presidência da República Federativa do Brasil
Ministério da Educação
Secretaria de Educação Profissional e Tecnológica

© Centro Federal de Educação Tecnológica de Minas Gerais
Este Caderno foi elaborado em parceria entre o Centro Federal de Educação Tecnológica de Minas Gerais e a Universidade Federal de Santa Catarina para a Rede e-Tec Brasil.

Equipe de Elaboração

Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG

Coordenação Institucional

Adelson de Paula Silva/CEFET-MG

Professor-autor

William Geraldo Sallum/CEFET-MG

Comissão de Acompanhamento e Validação

Universidade Federal de Santa Catarina – UFSC

Coordenação Institucional

Araci Hack Catapan/UFSC

Coordenação do Projeto

Silvia Modesto Nassar/UFSC

Coordenação de Design Instrucional

Beatriz Helena Dal Molin/UNIOESTE e UFSC

Coordenação de Design Gráfico

Juliana Tonietto/UFSC

Design Instrucional

Gustavo Pereira Mateus/UFSC

Web Master

Rafaela Lunardi Comarella/UFSC

Web Design

Beatriz Wilges/UFSC

Mônica Nassar Machuca/UFSC

Diagramação

Bárbara Zardo/UFSC

Breno Takamine/UFSC

Liana Domeneghini Chiaradia/UFSC

Luiz Fernando Tomé/UFSC

Marília Ceriulli Hermoso/UFSC

Roberto Gava Colombo/UFSC

Revisão

Júlio César Ramos/UFSC

Projeto Gráfico

e-Tec/MEC

Catálogo na fonte elaborada pela DECTI da Biblioteca Central da Universidade Federal de Santa Catarina.

S169a Sallum, William Geraldo
Aplicativos para a WEB II / William Geraldo Sallum. -
Belo Horizonte : CEFET- MG, 2012.
138 p.. : il., tabs.

Inclui bibliografia.

ISBN: 978-85-99872-27-7

Elaborado para o Curso Técnico em Planejamento e Gestão em Tecnologia da Informação da Rede e-Tec Brasil.

1.Tecnologia da informação. 2. Linguagem de programação (Computadores). 3. SQL (Linguagem de programação de computador). I. Título.

CDU : 681.31.066

Apresentação e-Tec Brasil

Prezado estudante,

Bem-vindo ao e-Tec Brasil!

Você faz parte de uma rede nacional pública de ensino, a Escola Técnica Aberta do Brasil, instituída pelo Decreto nº 6.301, de 12 de dezembro 2007, com o objetivo de democratizar o acesso ao ensino técnico público, na modalidade a distância. O programa é resultado de uma parceria entre o Ministério da Educação, por meio das Secretarias de Educação a Distância (SEED) e de Educação Profissional e Tecnológica (SETEC), as universidades e escolas técnicas estaduais e federais.

A educação a distância no nosso país, de dimensões continentais e grande diversidade regional e cultural, longe de distanciar, aproxima as pessoas ao garantir acesso à educação de qualidade, e promover o fortalecimento da formação de jovens moradores de regiões distantes, geograficamente ou economicamente, dos grandes centros.

O e-Tec Brasil leva os cursos técnicos a locais distantes das instituições de ensino e para a periferia das grandes cidades, incentivando os jovens a concluir o ensino médio. Os cursos são ofertados pelas instituições públicas de ensino e o atendimento ao estudante é realizado em escolas-polo integrantes das redes públicas municipais e estaduais.

O Ministério da Educação, as instituições públicas de ensino técnico, seus servidores técnicos e professores acreditam que uma educação profissional qualificada – integradora do ensino médio e educação técnica, – é capaz de promover o cidadão com capacidades para produzir, mas também com autonomia diante das diferentes dimensões da realidade: cultural, social, familiar, esportiva, política e ética.

Nós acreditamos em você!

Desejamos sucesso na sua formação profissional!

Ministério da Educação
Janeiro de 2010

Nosso contato
etecbrasil@mec.gov.br

Indicação de ícones

Os ícones são elementos gráficos utilizados para ampliar as formas de linguagem e facilitar a organização e a leitura hipertextual.



Atenção: indica pontos de maior relevância no texto.



Saiba mais: oferece novas informações que enriquecem o assunto ou “curiosidades” e notícias recentes relacionadas ao tema estudado.



Glossário: indica a definição de um termo, palavra ou expressão utilizada no texto.



Mídias integradas: sempre que se desejar que os estudantes desenvolvam atividades empregando diferentes mídias: vídeos, filmes, jornais, ambiente AVEA e outras.



Atividades de aprendizagem: apresenta atividades em diferentes níveis de aprendizagem para que o estudante possa realizá-las e conferir o seu domínio do tema estudado.

Sumário

Palavra do professor-autor	9
Apresentação da disciplina	11
Projeto instrucional	13
Aula 1 – Servidor web: preparando para utilizar o PHP	15
1.1 Introdução	15
1.2 Instalando o servidor web e serviços	16
Aula 2 - PHP: conceitos, evolução estrutura básica e operadores	29
2.1 Introduzindo a linguagem PHP	29
2.2 História e evolução da linguagem PHP	30
2.3 Como fazer rodar o PHP	32
2.4 Introduzindo a linguagem PHP	32
2.5 Declarando PHP	32
2.6 Primeiro <i>script</i> PHP	33
2.7 Teste de <i>script</i> PHP	34
2.8 Comentários em PHP	35
2.9 Variáveis	35
2.10 PHP é uma linguagem fracamente tipada	36
2.11 Regras de denominação para as variáveis	36
2.12 Operadores do PHP	37
Aula 3 – PHP: estruturas condicionais e funções de tempo	43
3.1 Estrutura condicional <i>if ... else</i>	43
3.2 Instrução <i>switch</i>	49
3.3 Funções de data e hora	50
Aula 4 – PHP: estruturas de repetição e funções externas	59
4.1 A estrutura de repetição	59
4.2 Funções externas	63

Aula 5 – PHP: funções matemáticas e matrizes	69
5.1 Funções matemáticas	69
5.2 Matrizes	75
5.3 O laço <i>foreach</i>	85
Aula 6 - PHP: funções <i>string</i> e funções de inserção de arquivos externos	89
6.1 Variáveis do tipo <i>string</i>	89
6.2 Incluindo arquivos – <i>Server Side Includes (SSI)</i>	99
Aula 7 – PHP: manipulação de formulários	105
7.1 Manipulação de formulários	105
7.2 <i>Upload</i> de arquivos	109
Aula 8 – PHP: manipulando o banco de dados MySQL	117
8.1 Conexão de banco de dados com PHP	117
Referências	137
Currículo do professor-autor	138

Palavra do professor-autor

Prezado estudante!

Praticamente todos os sistemas informacionais desenvolvidos hoje em dia tratam de dados e/ou informações utilizando algum tipo de tecnologia ligada à internet. As informações produzidas ou tratadas podem ser transmitidas de um lugar a outro ou simplesmente manipuladas localmente. Mesmo para as informações manipuladas localmente, o uso de recursos tais como linguagens de programação e aplicativos de desenvolvimento está ligado direta ou indiretamente à web. Isto tem indicado o grau de relevância que a área de desenvolvimento para web vem acumulando dia a dia.

A nossa disciplina – Aplicativos para Web II, ou AW2, possui a característica de ser bem-aceita e bem assimilada, pois trabalha diretamente com tecnologia de vanguarda na comunicação, utilizada em todo o mundo: a internet. Esta disciplina, assim como as demais deste curso, visa fornecer informações para que você possa desenvolver e manter programas ou aplicativos para a internet capazes de interagir com o usuário em qualquer parte do mundo. Assim, espero que aproveite todas as aulas e lembre-se: acompanhar integralmente as aulas e praticar efetivamente os exercícios é imprescindível para sedimentar as informações obtidas no seu banco de conhecimentos.

Um grande abraço!
William G. Sallum

Apresentação da disciplina

Aplicativos para Web II é uma disciplina do curso de PGTI que se apresenta com um conteúdo bastante focado no desenvolvimento de interações entre clientes e servidores. Esta disciplina contempla a parte de interação entre páginas locais, banco de dados e servidor remoto, proporcionando conhecimentos acerca de sua criação, subsidiados pela utilização dos seguintes conhecimentos técnicos:

- Ambiente *Web-server* – Conceitos, funcionalidades, elementos, infraestrutura, instalações e configurações do ambiente dos servidores *web*.
- Linguagem PHP – Linguagem que opera do lado dos servidores, com sintaxe semelhante à linguagem C e C++ e bastante rica em funções de manutenção, armazenamento e transmissão de dados.

O objetivo principal consiste em instrumentalizar o aluno com conceitos e habilidades técnicas, metodologias para o desenvolvimento de aplicativos e construção de páginas avançadas para a web.

Especificamente, são objetivos desta disciplina: transmitir conhecimento acerca de desenvolvimento de páginas a partir de conceitos sobre a estrutura e a operacionalização dos servidores *web*; desenvolvimento de páginas utilizando a linguagem PHP com objetivos de transmissão de dados, manipulação e armazenamento de informação. Além desses, ao final da disciplina, o aluno deverá: dominar tecnologias para desenvolvimento de páginas web dinâmicas; planejar uma estrutura de servidor *web*; estruturar e desenvolver um *site*, conciliando informação e *design*; executar estudos de casos usando esses conceitos; integração PHP com MySQL.

Os conteúdos programáticos serão abordados sobre servidores *web*. Também serão abordados os conceitos da linguagem PHP, sua sintaxe, variáveis, operadores, estruturas de decisão e repetição, construção de funções, transferência de dados entre páginas, bem como conexão com BDs e manipulações de informações em BDs.

Projeto instrucional

Disciplina: Aplicativos para Web II (carga horária: 60h).

Ementa: Linguagem de programação para web-PHP. Banco de dados MySQL.

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
1. Servidor web: preparando para utilizar o PHP	Conhecer conceitos de servidores <i>web</i> e suas funcionalidades. Identificar a escolha e instalação do servidor <i>web</i> bem como agregados.	Instalador de servidor internet WampServer; Instalador portátil de internet WOS.	7
2. PHP: conceitos, evolução, estrutura básica e operadores	Conhecer a Linguagem PHP. Aprender a instalação e configuração da "máquina" PHP. Compreender a estrutura básica da linguagem PHP e sua sintaxe. Saber sobre a evolução dessa linguagem. Compreender a utilização de variáveis e seus operadores.	Acesso ao link: http://www.php.net/manual/en/history.php ; Editoração e visualização de código-fonte através do aplicativo MAX's HTML.	7
3. PHP: estruturas condicionais e funções de tempo	Compreender as estruturas condicionais <i>if...else</i> e <i>switch...case</i> . Conhecer as funções que manipulam data e hora.	Editoração e visualização de código-fonte através do aplicativo MAX's HTML.	7
4. PHP: estruturas de repetição e funções externas	Entender as estruturas de repetição <i>while</i> e <i>for</i> . Conhecer sobre a criação de funções externas.	Editoração e visualização de código-fonte; ambiente virtual de ensino-aprendizagem: fórum de discussão.	7
5. PHP: funções matemáticas e matrizes	Conhecer sobre as funções matemáticas e matrizes.	Editoração e visualização de código-fonte pelo aplicativo MAX's HTML; fórum de discussão.	7
6. PHP: funções <i>string</i> e funções de inserção de arquivos externos	Conhecer os operadores da linguagem e as funções <i>string</i> . Saber como inserir arquivos externos ao <i>script</i> .	Editor de códigos-fontes MAX's HTML; ambiente virtual de ensino-aprendizagem: fórum de discussão.	7
7. PHP: manipulação de formulário	Utilizar formulários e os tipos de transferências de dados entre páginas, utilizando os operadores <code>\$_GET</code> e <code>\$_POST</code> .	Editor de códigos-fontes MAX's HTML; fórum de discussão; e <i>chats</i> .	7
Continua			

AULA	OBJETIVOS DE APRENDIZAGEM	MATERIAIS	CARGA HORÁRIA (horas)
8. PHP: manipulando o banco de dados MySQL	Compreender a conexão com o banco de dados MySQL. Entender a manipulação e armazenamento de dados. Rever os conceitos trabalhados ao longo da disciplina e integrar todas as tecnologias e técnicas para elaboração de páginas <i>web</i> completas.	Acesso ao <i>link</i>: http://www.dca.fee.unicamp.br/cursos/PooJava/javadb/sql.html; Gerenciador phpmyadim; Editor de códigos-fontes MAX's HTML	11
Conclusão			

Aula 1 – Servidor web: preparando para utilizar o PHP

Objetivos

Conhecer conceitos de servidores *web* e suas funcionalidades.

Identificar a escolha e instalação do servidor *web* bem como agregados.

1.1 Introdução

A computação ou simplesmente a utilização do computador pode ser restrita ao local residencial, ao local de trabalho ou à escola. Mas a troca de informações, seja buscando e/ou publicando, é o processo mais inusitado desde o último século; e é também o processo mais trivial, quando visto sob a ótica do usuário atual.

Buscar ou publicar informações implica diretamente no entendimento e na operação de estruturas do tipo cliente-servidor. Isto significa que, se de um lado tem-se um computador ou um sistema que necessita de serviços (como, por exemplo, transferência de um arquivo, ou de *e-mail*, ou simplesmente de uma página da internet), denominado de cliente, do outro deve existir um computador ou sistema que possa prover ou atender às necessidades de tais solicitações; e aí, o sistema situado desta outra ponta é denominado de servidor. Em suma: quem necessita de algum tipo de serviço ou produto digital e o solicita é denominado **cliente**; aquele que fornece tais serviços ou produtos é denominado **servidor** (MELO; NASCIMENTO, 2007, p. 27-44).

Entretanto, sob o ponto de vista técnico, o funcionamento de um sistema cliente-servidor exige a compreensão em análise, instalação, operação e manutenção de todo um ambiente de prestação de serviços virtuais através da internet, de forma mais acurada e sistemática. O cerne de um ambiente como este – também denominado de gestor ou máquina *web* – é o servidor *web*.

Os servidores *web* são os principais fornecedores de conteúdo da internet e os dois mais famosos servidores *web* executados pelo mundo afora são:

o IIS – Internet Information Server (Servidor de Informações da Internet da Microsoft®) e o Apache, inicialmente criado para plataformas Unix e depois estendido para Windows®. Esta responsabilidade se divide, principalmente, entre dois serviços: o Httpd do grupo Apache, também responsável por aproximadamente 65% das máquinas servidoras em todo o mundo, e o IIS da Microsoft, com cerca de 25% do mercado – segundo *sites* especializados no assunto. O restante do mercado (10%) é distribuído a outros servidores.

O IIS é um servidor proprietário da Microsoft (GILMORE, 2008, p. 16; RAMOS et al., 2007, p. 12) e o Apache é um *software* livre regido pelas normas GPL, com suporte a uma grande variedade de sistemas operacionais, incluindo todas as versões do Windows, Linux, Unix e Mac OS X, além de uma variedade de sistemas operacionais *nonmainstream*, como o BeOS e VMS. Nesse sentido, nos ateremos, no curso, à instalação e configuração do servidor Apache para ambiente Windows.

A-Z

WampServer

É uma versão mais atualizada do pacote WAMP5.



Os *links* ao lado fornecem duas opções para baixar o WAMP: WampServer 2.1e (versão 32 *bits*) e WampServer 2.1d (versão 64 *bits*). Você deve escolher a versão que atenda à sua configuração de plataforma. A versão 32 *bits* do pacote WampServer possui 19,6 Mb de tamanho e a versão 64 *bits*, 24,9 Mb. O tempo de *download* depende diretamente do tipo e velocidade de sua conexão com a internet.



1.2 Instalando o servidor web e serviços

Basicamente precisaremos de três componentes para a construção de um servidor **web**: o próprio servidor *web*, que nesse caso o escolhido aqui é o Apache; uma linguagem de interface cliente-servidor que trabalhe do lado do servidor, que no caso é o PHP, e um banco de dados: o MySQL (RAMOS et al., 2007, p.15-16).

Como a instalação será em um ambiente Windows, opcionalmente podemos adotar duas posturas: ou instalar os sistemas Apache, PHP e MySQL separadamente (HERRINGTON, 2008, p. 22-32; NIEDERAUER, 2007), ou todos em um, como é o caso do pacote **WampServer** (que possui os aplicativos: Windows, Apache, MySQL e PHP), obtido gratuitamente pela internet (WAMP, 2012).

1.2.1 Onde e como conseguir o pacote WampServer

Para instalar o seu servidor, utilize os *links*: <http://www.wampserver.com/en/download.php> ou <http://www.wampserver.com/en/#download-wrapper>.

Tais *links* fornecem opção de instalação para a plataforma Linux.

1.2.2 Instalação do pacote WampServer

Como dito anteriormente, o pacote WampServer é entregue com as disposições das últimas versões do Apache, MySQL e PHP. Depois do WampServer ter sido instalado, você pode adicionar outros lançamentos por *download*. Eles irão aparecer no menu WampServer e você será capaz de mudar de versões com um simples clique.

Dê um duplo clique no arquivo baixado e siga as instruções da instalação. Tudo é quase automático. Assim, após clicar no arquivo baixado, surgirá uma janela semelhante à da Figura 1.1.

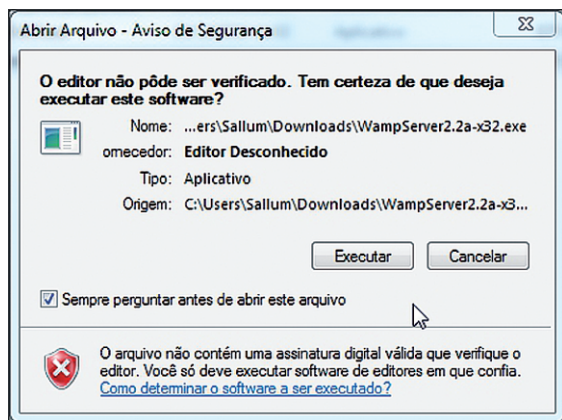


Figura 1.1: Janela de instalação do WampServer solicitando autorização para a autoinstalação

Fonte: WampServer, 2011

Selecione executar o arquivo clicando no botão "Run" ou "Executar".

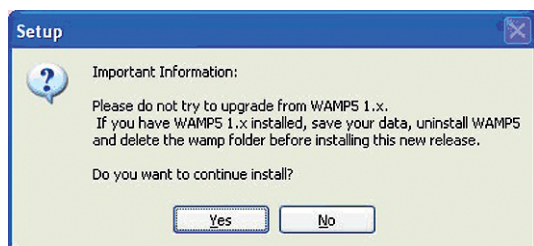


Figura 1.2: Janela de instalação do WampServer alertando para o caso de haver alguma versão anterior do sistema

Fonte: WampServer, 2011

Você será questionado, conforme mostra a Figura 1.2, sobre se deseja ou não instalar essa versão em cima da WAMP5 1.x., caso ela exista. Clique em **Sim (yes)** para continuar com a instalação.

A partir daí, aparecerá a primeira janela de instalação de boas-vindas (Figura 1.3). Clique em **Avançar (Next)** para continuar.



Figura 1.3: Janela de instalação do WampServer dando as boas-vindas à instalação
Fonte: WampServer, 2011

A janela seguinte (Figura 1.4) contém a *General Public License* (GNU), que lhe permitirá usar esse *software*. Após ler toda a licença, marque o botão “*I accept the agreement*” caso aceite os termos apresentados no texto lido. Em seguida clique no botão **Avançar (Next)** para continuar.

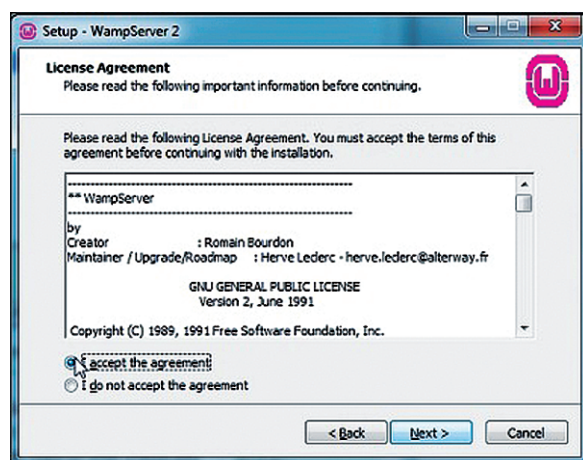


Figura 1.4: Janela de instalação do WampServer solicitando autorização de instalação conforme licença
Fonte: WampServer, 2011

A caixa da Figura 1.5 permite que você escolha onde gostaria de instalar todos os arquivos. Você pode mudar esta opção se desejar, mas, se o fizer, tenha em mente que você deve colocá-los em uma pasta cujo nome não contenha quaisquer espaços, porque alguns navegadores/servidores têm problemas de manipulação de espaços em nomes de arquivo. O local selecionado é também onde os arquivos da *web* serão armazenados. Se não tiver certeza, aceite a sugestão impressa no campo da janela abaixo.

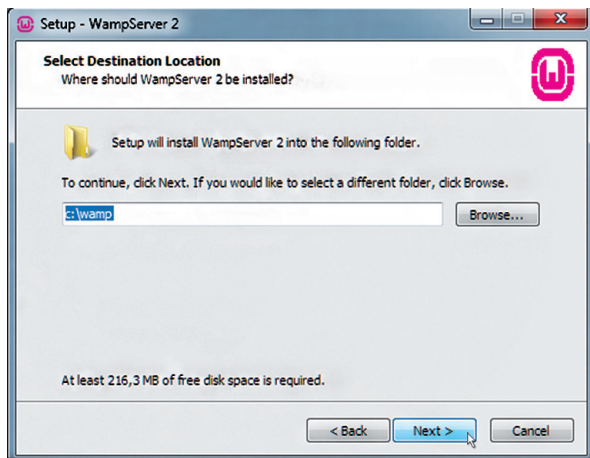


Figura 1.5: Janela de instalação do WampServer instruindo sobre o local físico da instalação

Fonte: WampServer, 2011

A janela da Figura 1.6 permite criar o ícone no *desktop* e na barra de inicialização rápida.

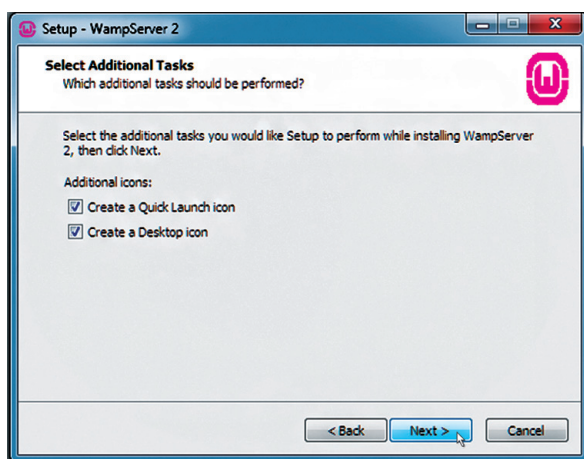


Figura 1.6: Janela de instalação do WampServer sobre a criação dos ícones do sistema

Fonte: WampServer, 2011

A próxima janela (Figura 1.7) apenas informa sobre todas as opções de instalação selecionadas. Verifique se as configurações estão corretas e clique em **Avançar (Next)** para iniciar a instalação do aplicativo.

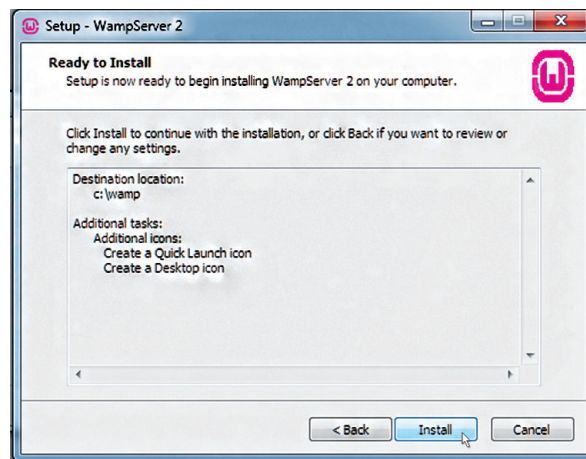


Figura 1.7: Janela de instalação do WampServer aguardando autorização para iniciar a autoinstalação

Fonte: WampServer, 2011

A janela a seguir (Figura 1.8) mostra a barra de progressão de instalação do *software*.

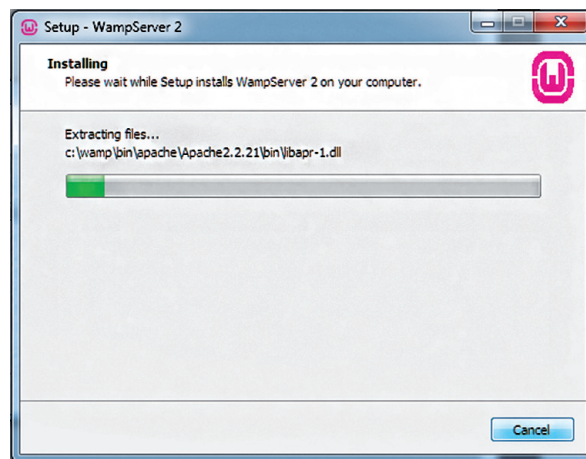


Figura 1.8: Janela de instalação do WampServer com a barra de instalação do sistema

Fonte: WampServer, 2011

A próxima janela (Figura 1.9) sobrepõe a janela anterior (Figura 1.8) e solicita que o usuário informe qual o programa do *browser* deverá ser inicialmente utilizado pelo WampServer como padrão de gerenciamento do servidor Apache.

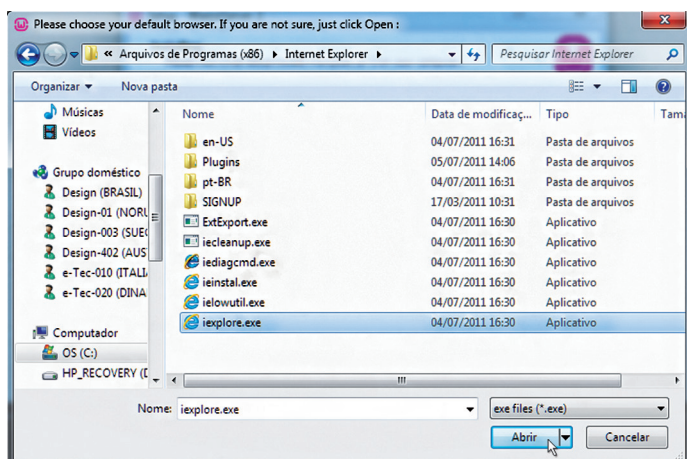


Figura 1.9: Janela de instalação do WampServer instruindo sobre a preferência de *browser*

Fonte: WampServer, 2011

A próxima janela (Figura 1.10) permite que você configure o seu servidor para encaminhar qualquer *e-mail* que o PHP produza para uma conta de correio eletrônico. Se você não tem um *e-mail* ou não deseja receber qualquer mensagem do sistema, basta deixá-los como se apresentam e clicar em **Avançar (Next)** e prosseguir para a próxima janela, conforme Figura 1.11.

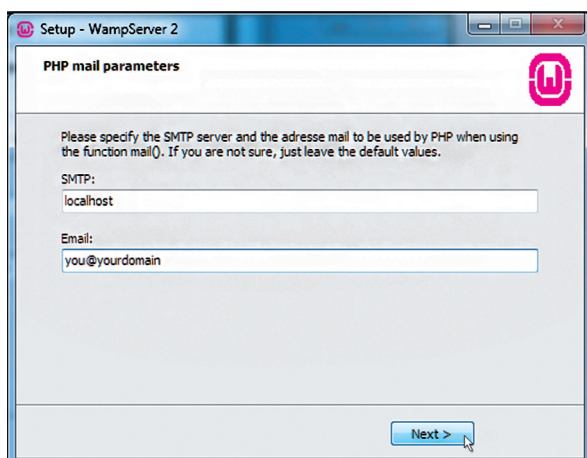


Figura 1.10: Janela de instalação do WampServer instruindo sobre a conta de e-mail

Fonte: WampServer, 2011

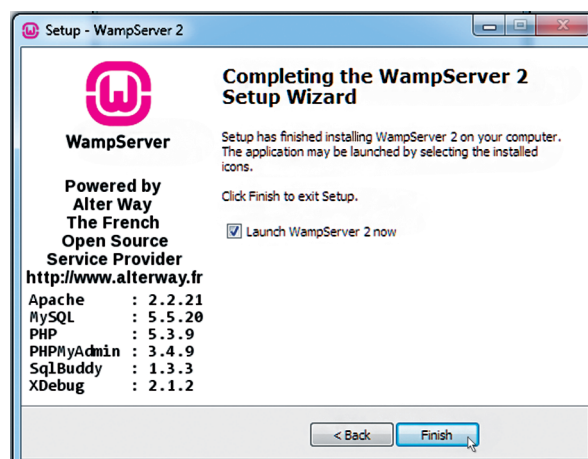


Figura 1.11: Janela de instalação do WampServer instruindo sobre o final da instalação

Fonte: WampServer, 2011

A Figura 1.11 mostra a janela de conclusão da instalação do aplicativo. A partir daí, você já pode iniciar o servidor automaticamente, depois de fechar esta janela. Clique em **Concluir (Finish)** para finalizar a instalação.

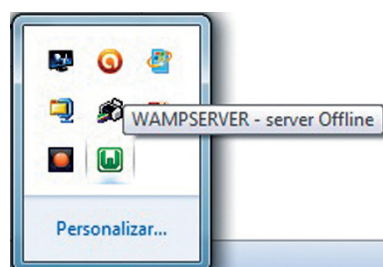


Figura 1.12: Detalhe da barra de tarefas do Windows indicando operacionalidade do WampServer

Fonte: WampServer do Windows, 2011

Depois que o servidor estiver rodando, você verá o ícone do servidor Wamp, conforme Figura 1.12, na barra de tarefas (para todos os Windows, exceto Windows 7).

O menu acima deverá ser apresentado assim que você clicar no ícone do aplicativo que aparecerá na barra de tarefas, ao lado do relógio. As opções desse menu são apresentadas na Figura 1.12 a seguir.

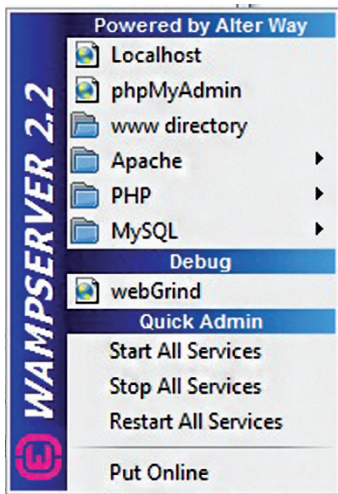


Figura 1.13: Menu *pop-up* de aplicativos WampServer e respectivas configurações

Fonte: WampServer, 2011

- **Localhost** – esta opção simplesmente abre a página que está armazenada no diretório *home* (que é “C:\wamp\www\index.php” por padrão ou aquela que você determinou durante a instalação). Você também pode obter esta página, simplesmente digitando “http://localhost” ou http://127.0.0.1 na barra de endereços de seu navegador *web*.
- **phpMyAdmin** – esta opção irá apresentar o aplicativo phpmyadmin (que também pode ser executado, indo para http://localhost/phpmyadmin/ no seu navegador *web*). O phpMyAdmin é uma ferramenta desenvolvida para gerenciar, consultar, selecionar e visualizar suas bases de dados MySQL. Este aplicativo foi desenvolvido em PHP e é relativamente simples de usar.
- **www Directory** – esta opção irá abrir a pasta onde os *sites* estão armazenados. Este é o diretório onde os arquivos serão carregados quando você evoca *localhost*.
- **Apache** – esta opção disponibiliza as definições de configuração para o Apache e permite que você faça a configuração de que você necessita.
- **PHP** – esta opção possui as definições de configuração do PHP e irá permitir que você faça a configuração de que você necessita.
- **MySQL** – esta opção possui as configurações do MySQL e permitirá que você faça a configuração de que você necessita.

A partir daí, você detém as opções para controlar os serviços do servidor web bem como torná-lo *on* ou *off-line*.

Por padrão, o servidor é configurado com um arquivo `index.php` que mostra os serviços que são iniciados pelo servidor. Esse arquivo mostra também os programas que começaram em seu servidor com base nas pastas que estão dentro do diretório `www`. Você não tem que manter esse arquivo se você não quiser. Você pode apagá-lo e fazer sua própria `index.php` que será exibida sempre que você acessar *localhost*. Existem várias configurações que podem ser feitas para a presente configuração do servidor.

1.2.3 Instalando um servidor num drive removível (via porta USB)

Imagine a seguinte situação: Você está precisando apresentar ou testar uma aplicação desenvolvida em PHP e a máquina pela qual a apresentação se dará não possui instalado um servidor Apache e muito menos uma máquina PHP rodando. E o pior, essa máquina não permite a instalação de qualquer aplicativo. Este é, realmente, um grande problema!

Mas existe uma solução bastante original para este e outros casos semelhantes: o pacote WOS (*Web of Server* ou *Servidor Web Portátil*). Este pacote é um aplicativo destinado a ser utilizado a partir de um *pendrive* USB. WOS é um *software* gratuito que permite a rápida criação de um sistema WAMP (Windows, Apache, MySQL e PHP) em qualquer sistema Windows.



Vamos clicar no *link* da página de *download* do WOS <http://wos-portable.en.softonic.com/download> e baixar o arquivo `SoftonicDownloader_for_wos_portable.exe`. Terminado o *download* do arquivo, vamos transferi-lo para o nosso *pendrive* e clicar duas vezes no arquivo para executá-lo e, assim, iniciarmos a instalação. Escreva em um documento digital quais as dificuldades que você teve ao fazer o que foi solicitado. Poste esse relatório no ambiente virtual, na atividade tarefa criada para esta finalidade.

Primeiramente devemos fazer o *download* desse pacote para nosso computador.

1.2.4 Testando as funcionalidades do pacote WampServer

Após a conclusão da instalação do pacote, é necessário verificar se o Apache, que é a “máquina” que faz funcionar a linguagem PHP, bem como as funcionalidades do MySQL, está rodando.

A primeira providência a ser executada é testar se o Apache está funcionando. Para isso vamos abrir o browser e, na sua barra de endereços, digitar o URL: <http://localhost/>

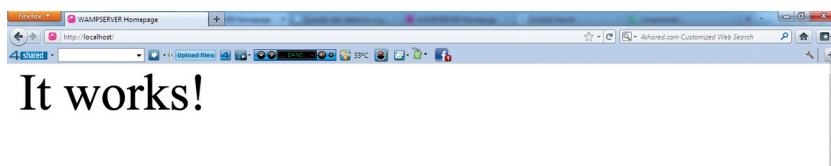


Figura 1.14: Janela de retorno da página do Apache confirmando a sua funcionalidade-
Fonte: Firefox versão 9.0.1

Se na janela do *browser* aparecer a mensagem “It works!”, tal como ilustrado na Figura 1.14, parabéns!!! O Apache está instalado corretamente e está funcionando.

Agora é necessário verificar se o PHP está rodando. Para isto vamos criar um arquivo PHP e salvá-lo no local de referência do localhost, que poder ser: a pasta www (se instalado o WAMP5 em C:/ ou htdocs se outro pacote – por exemplo: XAMPP). Desse modo, vamos criar um arquivo com um *script* bem simples, conforme Figura 1.15 a seguir.

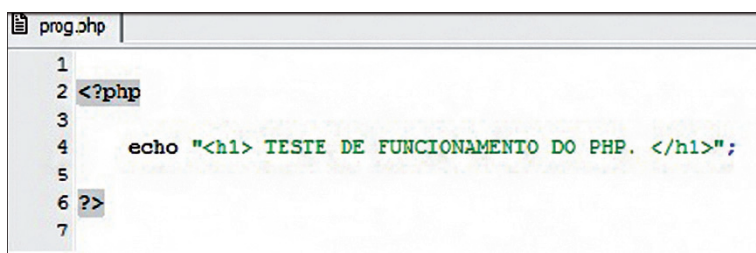


Figura 1.15: Script de teste de funcionalidade do PHP
Fonte: Elaborada pelo autor

Mais adiante, iremos explicar como esse *script* funciona. Por hora, vamos apenas digitá-lo no editor de texto de sua preferência (aqui nós utilizamos o MAX's HTML *Beauty*) e salvá-lo com o nome prog.php, dentro da pasta especificada anteriormente.

Agora, vamos abrir o *browser* e digitar na barra de endereço: **http://localhost/prog.php**, evocando o arquivo gerado: prog.php. Caso o PHP esteja em correto funcionamento, a tela da Figura 1.16 deverá ser apresentada.



Figura 1.16: Janela de retorno de execução do script desenvolvido na figura anterior
Fonte: Elaborada pelo autor e visualizada pelo Firefox versão 9.0.1

Agora é a vez de verificar se o nosso banco de dados (BD) também está funcionando (RAMOS et al., 2007, p. 15). Para isto, vamos digitar no *browser* o endereço: <http://localhost/phpmyadmin/>, conforme mostrado na Figura 1.17 a seguir.

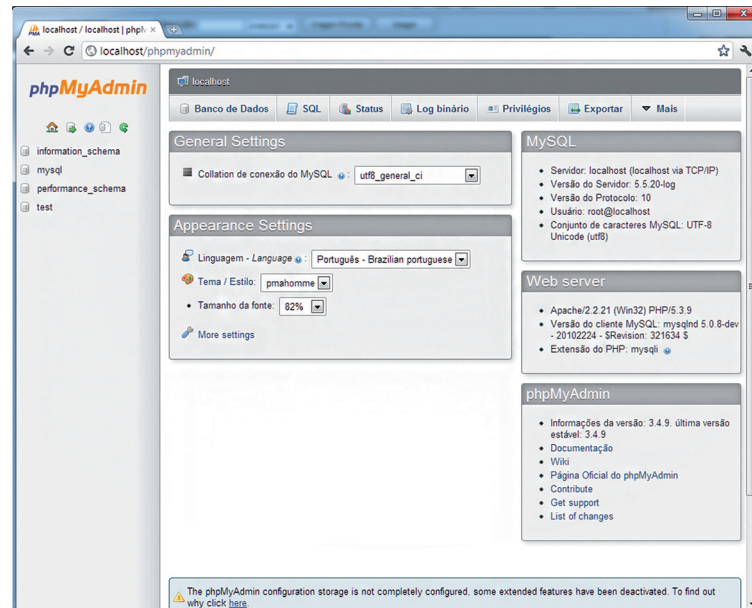


Figura 1.17: Janela da página do gerenciador phpMyAdmin confirmando sua funcionalidade
Fonte: Browser Firefox versão 9.0.1.

Se a tela anterior (Figura 1.17) foi apresentada, então nosso BD também está funcionando corretamente. Mais adiante iremos mostrar como criar banco de dados e tabelas utilizando o aplicativo phpMyAdmin.

Resumo

Nesta aula, aprendemos os conceitos de sistema Cliente-Servidor, conhecemos a sua estrutura e quais são os seus objetivos. Vimos que é necessário um aplicativo sendo executado nos bastidores do sistema de modo a promover e gerenciar a comunicação de dentro para fora e de fora para dentro do computador. Entendemos que esse aplicativo é um servidor de acesso e de comunicação e que existem instalações pagas e gratuitas (ISS e Apache, respectivamente). Soubemos onde obter e como instalar o servidor Apache passo a passo. Aprendemos, também, que existem versões compactas (WOS) para instalação em memórias removíveis (do tipo USB), ideais para testes e manutenções.

Atividades de aprendizagem

1. Você deve instalar o pacote Wamp5 ou WampServer ou Xampp (mais indicado para o Windows 7) em um computador, conforme orientações descritas nesta aula. Deve, ainda, deixar o sistema funcionando corretamente. Ao final, escreva um pequeno relatório sobre essa atividade e poste no fórum criado para esta finalidade no ambiente virtual de ensino-aprendizagem.

Opcionalmente, para esta atividade, você pode instalar no *pendrive* o WOS (ou pacote semelhante) em substituição ao Wamp5 ou WampServer ou Xampp.



Aula 2 - PHP: conceitos, evolução, estrutura básica e operadores

Objetivos

Conhecer a linguagem PHP com seus conceitos e evolução histórica.

Aprender a instalação e configuração da “máquina” PHP.

Compreender a estrutura básica da linguagem PHP e sua sintaxe.

Compreender a utilização de variáveis e seus operadores.

2.1 Introduzindo a linguagem PHP

Até recentemente, a criação de *scripts* na internet era algo que poucas pessoas dominavam. Recentemente, porém, mais e mais pessoas vêm construindo seus próprios *websites* e linguagens de *script*. Devido a isso, as linguagens de *script* estão ficando mais fáceis de aprender e a linguagem PHP é uma das mais poderosas.

PHP é uma linguagem *server-side*, isto é, o *script* é executado no seu servidor *web*, e não no navegador do usuário. Uma grande vantagem é o fato de não precisar se preocupar com problemas de compatibilidade (MELO; NASCIMENTO, 2007, p. 43-44). PHP é relativamente nova (em comparação com outras linguagens como Perl-CGI e Java), mas está rapidamente se tornando uma das mais populares linguagens de *script* na internet.

Você pode estar se perguntando por que devemos escolher PHP em relação às outras linguagens, tal como Perl, ou mesmo por que você deve aprender uma linguagem de *script*. Do ponto de vista de alguém que está iniciando, aprender uma linguagem de *script* pode abrir novas e enormes possibilidades para o seu *site*. Embora você possa fazer *download* de *scripts* pré-fabricados pela internet, estes geralmente ou contêm publicidade para o autor ou não vão fazer exatamente o que você precisa que façam. Como estudioso e/ou técnico em informática e especialista em uma linguagem de *script*, você pode facilmente editar quaisquer *scripts* dessa linguagem para fazer o que desejar, além de poder criar seus próprios *scripts*, páginas, portais, etc.

A utilização de *scripts* nos *sites* permite que se adicionem novos recursos “interativos”, tais como: formulários, *guestbooks*, fóruns, contadores e até recursos mais avançados, como sistemas de portais, gestão de conteúdos, publicidade, etc. Desenvolver páginas utilizando programação em linguagens de *script* demonstra, do ponto de vista técnico, imagem mais profissional, além de ampliar os horizontes no mercado ávido por profissionais dessa categoria.

2.2 História e evolução da linguagem PHP

A história de surgimento e evolução da linguagem PHP é narrada no endereço <http://www.php.net/manual/en/history.php.php>. Veja o resumo da tradução desta história:

O PHP sucede a um produto mais antigo, chamado PHP/FI. PHP/FI. Foi criado por Rasmus Lerdorf em 1995, inicialmente como um simples conjunto de *scripts* Perl para monitorar os acessos ao seu currículo *on-line*. Ele chamou esse conjunto de *scripts* de “*Personal Home Page Tools*”. Como mais funcionalidades foram requeridas, Rasmus escreveu uma implementação C muito maior, que foi capaz de se comunicar com bancos de dados e possibilitava a usuários desenvolver simples aplicativos dinâmicos para *web*. Rasmus resolveu disponibilizar o código fonte do PHP/FI para que todos possam ver, para que qualquer pessoa possa usá-lo, bem como fixar *bugs* e melhorar o código.

PHP/FI, o que representava **Personal Home Page/Forms Interpreter**, incluía algumas funcionalidades básicas do PHP que nós conhecemos hoje. Tinha variáveis em Perl, interpretação automática de variáveis de formulário e sintaxe embutida no HTML. A sua própria sintaxe era similar à do Perl, porém muito mais limitada, simples e um pouco inconsistente.

Em 1997, PHP/FI 2.0, a segunda versão da implementação C, obteve milhares de usuários ao redor do mundo (estimado), com cerca de 50.000 domínios reportando que tinha instalado, o que representa cerca de 1% dos domínios na internet. Embora houvesse milhares de pessoas contribuindo com pequenos códigos para esse projeto, ele ainda estava no grande projeto de um único homem.

PHP/FI 2.0 foi oficialmente lançado somente em novembro de 1997, após passar a maior parte da sua vida em versões beta. Ele foi rapidamente substituído pelos *alphas* do PHP 3.0.

PHP 3 – O PHP 3.0 foi a primeira versão que se assemelha ao PHP que nós conhecemos hoje. Ela foi criada por Andi Gutmans e Zeev Suraski em 1997 e foi totalmente reescrita, após eles descobrirem que o PHP/FI 2.0 poderia ajudá-los a desenvolver suas próprias aplicações de e-Commerce de um projeto da Universidade. Em um esforço cooperativo e iniciativa de começar o PHP/FI a partir da base-usuário existente, Andi, Rasmus e Zeev decidiram cooperar e anunciar o PHP 3.0 como o sucessor oficial do PHP/FI 2.0, e o desenvolvimento do PHP/FI 2.0 foi descontinuado.

Uma das maiores características do PHP 3.0 era sua forte capacidade de extensibilidade. Além de oferecer aos usuários finais uma infraestrutura sólida para diversos bancos de dados, protocolos e APIs, a extensibilidade do PHP 3.0 atraiu dezenas de desenvolvedores para se juntar e submeter novos módulos. Esta é a chave do tremendo sucesso do PHP 3.0. Outras características-chave introduzidas no PHP 3.0 foram o suporte à sintaxe para orientação a objetos e uma sintaxe muito mais poderosa e consistente.

A nova versão da linguagem foi disponibilizada com um novo nome, que eliminou a impressão do limitado uso pessoal que o PHP/FI 2.0 possuía. Ela foi nomeada simplesmente de 'PHP'.

Ao final de 1998, o PHP obteve uma base de dezenas de milhares de usuários (estimativa) e centenas de milhares de *websites* relatando que o tinham instalado. Em seu pico, o PHP 3.0 foi instalado em aproximadamente 10% dos servidores *web* da internet.

O PHP 3.0 foi oficialmente lançado em junho de 1998, depois de ter passado aproximadamente nove meses em testes públicos.

PHP 4 – Esta versão foi baseada numa nova engenharia e acompanhada de uma série de novas características, sendo oficialmente lançada em maio de 2000, pela dupla de engenheiros: Zeev e Andi, quase dois anos após o seu predecessor, o PHP 3.0. Além do altíssimo melhoramento da *performance* desta versão, o PHP 4.0 incluiu outras características-chave como o suporte para muitos servidores *web*, sessões HTTP, *buffer* de saída, maneiras mais seguras de manipular *input* de usuários e muitas construções novas na linguagem.

A partir daí, o PHP começou a ser usado por centenas de milhares de desenvolvedores (estimativa), e muitos milhões de *sites* reportam que têm o PHP instalado, que explica os mais de 20% de domínios da internet.



Para complementar o histórico sobre a linguagem PHP, consulte também Gilmore (2008, p. 1-8).

PHP 5 – Foi lançado em julho de 2004, depois de um longo desenvolvimento e vários *pre-releases*. É dirigido principalmente pelo seu núcleo, a Zend Engine 2.0, com um novo modelo de objeto e dezenas de outras novas funcionalidades.

2.3 Como fazer rodar o PHP

Como mencionado anteriormente, o PHP é uma linguagem de *script server-side*. Isso significa que, embora os usuários não necessitem instalar um novo *software*, é necessário ter o PHP configurado em seu servidor. A linguagem PHP deve ser listada como parte do pacote com o Apache, tal como visto na primeira aula. Se o seu servidor de conteúdo (caso possua algum) não suporta o PHP, você pode solicitar ao seu provedor de hospedagem para instalá-lo, já que é livre fazer o seu *download* e a sua instalação.

2.4 Introduzindo a linguagem PHP

Escrever códigos PHP é realmente muito simples. Não é necessário nenhum *software* especializado, sendo necessário apenas um editor de texto (como o Bloco de notas do Windows ou outro específico, tal como o MAX's HTML – já visto na disciplina Aplicações para Web I). Entretanto, existem vários editores orientados para criação de programas PHP, bem como ambientes de desenvolvimento específicos para esse fim (GILMORE, 2008, p. 36-37; RAMOS et al. 2007, p.15-29; PHPEDITOR, 2012).

2.5 Declarando PHP

Scripts PHP são sempre colocados entre duas *tags*. Elas dizem ao seu servidor para analisar as informações que se encontram entre elas e executá-las. Existem três formas diferentes de compartimentar *scripts* PHP. São as seguintes:

- Código Resumido

```
<?  
código do PHP  
?>
```

- Código Normal

```
<?php  
código do PHP  
?>
```

- Código Estendido

```
<script language="php">  
código do PHP  
</script>
```

2.6 Primeiro *script* PHP

O primeiro *script* PHP será escrito de forma muito básica. Tudo o que ele vai fazer é mostrar todas as informações sobre o PHP do seu servidor. Digite o seguinte código em seu editor de texto:

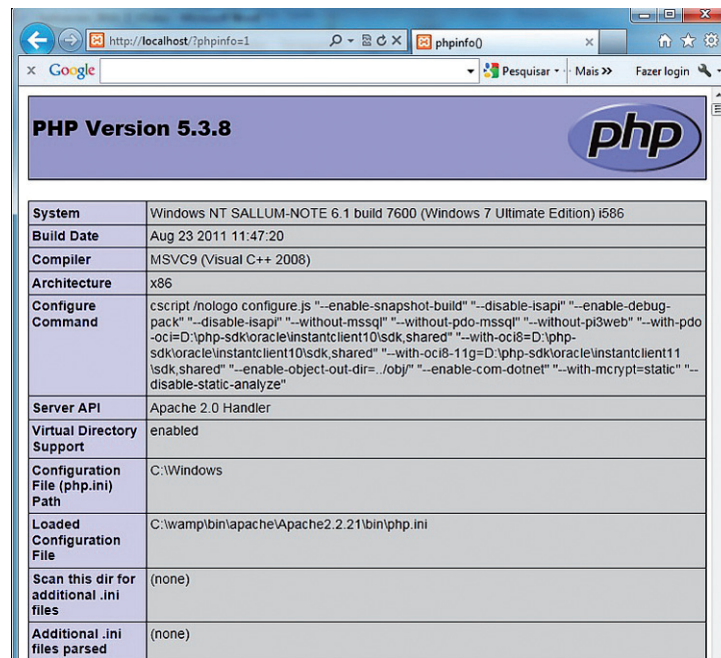
```
<?php  
phpinfo();  
?>
```

Como podemos ver, isso, na verdade, é apenas uma linha de código. É uma função *phpinfo* padrão do PHP que irá dizer ao servidor para mostrar uma tabela padrão com as informações sobre a configuração do servidor (GILMORE, 2008, p.18).

Outra coisa que se deve observar nesse exemplo é o símbolo de pontuação (;) ponto e vírgula terminando a linha de comando. Isso é muito importante. Tal como acontece com muitos outros *scripts* e linguagens de programação, quase todas as linhas são terminadas com um ponto e vírgula e, se for ignorado, o sistema imprimirá um erro.

2.7 Teste de *script* PHP

Agora que você terminou o *script*, é necessário salvá-lo com um nome qualquer, por exemplo: "t.php". Em seguida, faça o *upload* (cópia) para o servidor de forma normal ou, se o servidor se encontra no seu próprio computador (ou *pendrive*), copie para a pasta que está representada como **www** (ou **htdocs** se o Apache é o *Xampp*). Agora, usando o navegador, acesse a URL do *script*. Se ele está funcionando corretamente (e se o PHP está instalado no servidor), você deve visualizar uma página repleta de informações sobre o PHP, semelhante à imagem da Figura 2.1 a seguir.



The image shows a screenshot of a web browser window displaying the output of the PHP `phpinfo()` function. The browser's address bar shows `http://localhost/?phpinfo=1`. The page title is "PHP Version 5.3.8" and features the PHP logo. Below the header is a table containing system and configuration details.

System	Windows NT SALLUM-NOTE 6.1 build 7600 (Windows 7 Ultimate Edition) i586
Build Date	Aug 23 2011 11:47:20
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--disable-isapi" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8-11g=D:\php-sdk\oracle\instantclient11\sdk,shared" "--enable-object-out-dir=.\obj" "--enable-com-dotnet" "--with-mcrypt=static" "--disable-static-analyze"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\wamp\bin\apache\Apache2.2.21\bin\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)

Figura 2.1: Página gerada a partir da execução do arquivo `phpinfo.php`

Fonte: Internet Explorer, versão 9.0

Se o *script* não funcionar e aparecer uma página em branco, pode ter havido erro no código.

2.8 Comentários em PHP

No PHP, usamos `//` para fazer um comentário em uma única linha ou `/*` e `*/` para fazer um grande bloco de comentário. Veja os exemplos a seguir.

```
<html>
<body>

<?php
// Este é um comentário de uma linha

/*
Este é um
bloco de
comentário
*/
?>

</body>
</html>
```

Ao executar o *script* anterior, é claro que nada será mostrado, já que não existem comandos que façam alguma saída para a tela, apenas comentários.

2.9 Variáveis

As variáveis são usadas para armazenar valores, como cadeias de texto, números ou *arrays* vetores). Assim, quando uma variável é declarada, ela pode ser usada repetidamente outra vez em seu *script*.

Uma característica interessante e diferente das outras linguagens é que todas as variáveis do PHP começam com um sinal de `$`. Desse modo, a maneira correta de declarar uma variável em PHP é

```
$var_name = "value";
```

É comum novos programadores PHP esquecerem o sinal `$`, no início da variável. Nesse caso, o programa não vai funcionar, além de poder gerar algum tipo de erro.

Continuando, vamos ilustrar o processo de utilização de variáveis, criando uma variável contendo uma *string* e uma variável contendo um número:

```
<?php
$txt = "Ei mamãe, olha eu na web!";
$x=16;
?>
```

Neste exemplo, foi criada a variável do tipo *string*: **\$txt**, ao mesmo tempo em que lhe foi atribuído o valor: "Ei mamãe, olha eu na web!" e foi criada a variável do tipo inteiro: **\$x** já contendo o valor 16.

2.10 PHP é uma linguagem fracamente tipada

Como vimos anteriormente, no PHP, uma variável não precisa ser declarada antes de receber um valor. No exemplo anterior, pudemos verificar que não há a necessidade de dizer ao PHP qual tipo da variável (se inteiro, real, caracter, etc.), bastando apenas atribuir o que se deseja a variável e pronto: está criada e atribuída!

O compilador PHP converte automaticamente a variável para o tipo de dado correto, dependendo de seu valor atribuído.

Em uma linguagem de programação fortemente "tipada", é imprescindível declarar (definir) o tipo e o nome da variável antes de usá-la. Em outras palavras, no PHP a variável é declarada automaticamente já ao usá-la (MELO; NASCIMENTO, 2007, p. 52).

2.11 Regras de denominação para as variáveis

- Um nome de variável deve começar com uma letra ou um sublinhado "_";
- Um nome de variável só pode conter caracteres alfanuméricos e sublinhados (AZ, _ AZ, 0-9 e _).

Um nome de variável não deve conter espaços. Se um nome de variável é mais do que uma palavra, devem ser separadas por um sublinhado (**\$my_string**), ou com capitalização (**\$myString**).

2.12 Operadores do PHP

Esta seção lista os diferentes operadores utilizados no PHP. Semelhantemente aos já vistos em JavaScript, os operadores são usados para efetuar operações aritméticas, de atribuição, de comparação e lógicos.

a) Operadores aritméticos

O PHP é uma linguagem que, como outras linguagens procedurais, possui um conjunto de operadores aritméticos (ou matemáticos). O Quadro 2.1 relaciona e exemplifica uma lista desses operadores.

Quadro 2.1: Operadores aritméticos disponibilizados pelo PHP.			
Operador	Descrição	Exemplo	Resultado
+	Soma	$x = 2$ $x + 2$	4
-	Subtração	$x = 2$ $x - 5$	3
*	Multiplicação	$x = 4$ $x * 5$	20
/	Divisão	$15 / 5$ $5 / 2$	3 2.5
%	Módulo (resto da divisão)	$5 \% 2$ $10 \% 8$ $10 \% 2$	1 2 0
++	Incremento	$x = 5$ $x ++$	$x = 6$
--	Decremento	$x = 5$ $x --$	$x = 4$

Fonte: Tansley (2002, p. 24-29); Niederauer (2007, p. 9)

b) Operadores de atribuição

Os operadores de atribuição são responsáveis por atribuir ou armazenar valores em variáveis (ou constantes). O Quadro 2.2, a seguir, lista os vários modos de se atribuir valores ao mesmo tempo que se efetuam operações aritméticas.

Quadro 2.2: Operadores de atribuição disponibilizados pelo PHP.

Operador	Exemplo	É o mesmo que
=	y = x	y = x (x é atribuído a y)
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x / y =	x = x / y
.	y = x.	x = x . y (concatenação de strings)
% =	y = x%	x = x % y

Fonte: Tansley (2002, p. 24-29); Melo e Nascimento (2007, p.59); Niederauer (2007, p.10)

c) Operadores de comparação

No Quadro 2.3, a seguir, estão listados os operadores de comparação, responsáveis por estabelecer as diferenças ou igualdades entre dois operandos.

Quadro 2.3: Operadores de comparação disponibilizados pelo PHP.

Operador	Descrição	Exemplo
==	é igual a	5 == 8 retorna <i>false</i>
!=	não é igual	5 != 8 retorna <i>true</i>
<>	não é igual	5 <> 8 retorna <i>true</i>
>	é maior do que	5 > 8 retorna <i>false</i>
<	é inferior	5 < 8 retorna <i>true</i>
>=	é maior ou igual a	5 >= 8 retorna <i>false</i>
<=	é inferior ou igual a	5 <= 8 retorna <i>true</i>

Fonte: Tansley (2002, p. 24-29); Niederauer (2007, p.10); Melo e Nascimento (2007, p. 59)

A-Z

True

Verdadeiro - traduzindo da língua inglesa.

False

Falso – traduzindo da língua inglesa.

d) Operadores lógicos

O Quadro 2.4 lista os três operadores lógicos disponibilizados pelo PHP. Esses operadores são responsáveis por efetuar ligações entre operações de comparação.

Quadro 2.4: Operadores lógicos disponibilizados pelo PHP.

Operador	Descrição	Exemplo
& &	e	x = 6 y = 3 (x < 10 & & y > 1) retorna <i>true</i>
	ou	x = 6 y = 3 (x == 5 y == 5) retorna <i>false</i>
!	não	x = 6 y = 3 !(x == y) retorna <i>true</i>

Fonte: Niederauer (2007, p. 10); Melo (2008, p. 59)

e) Operador de concatenação

Existe apenas um operador de operação com *string* em PHP. É o operador de concatenação (.) ponto. Ele é usado para colocar dois valores de *string* juntos.

Para concatenar (ou juntar) duas variáveis *string*, usa-se o operador de concatenação. Vejamos o exemplo abaixo que concatena duas *strings*: **\$txt1** e **\$txt2**.

```
$txt1=" Ei mamãe";  
$txt2=", olha eu na web!";  
echo $txt1 . " " . $txt2;  
?>
```

A saída do código acima será:

```
Ei mamãe, olha eu na web!
```

Se olharmos o código acima, veremos que usamos o operador de concatenação duas vezes. Isso se dá porque tivemos que inserir uma terceira *string* (espaço), para separar as duas outras *strings* (**\$txt1** e **\$txt2**).

Resumo

Estudamos aqui os conceitos, origens e evoluções da linguagem PHP bem como seus criadores e colaboradores. Vimos a respeito das características básicas de funcionamento da linguagem e suas necessidades de execução. Entendemos a estrutura sintática de seus *scripts* ou códigos-fonte. Entendemos como criar, nomear variáveis e atribuir valores às mesmas bem como as regras de utilização. Estudamos os seus operadores aritméticos, de atribuição, de comparação, lógicos e de concatenação da linguagem. Ao longo desta aula, aplicamos exemplos de *scripts*, com visualização dos resultados gerados a partir deles.

Atividades de aprendizagem

1. Pesquise e assinale Verdadeiro (V) ou Falso (F) as afirmações abaixo:

a) Pode-se escrever *scripts* PHP que rodem do lado do cliente.

() Verdadeiro

() Falso

- b)** Para rodar o PHP é necessário um servidor *web*.
- ☐ Verdadeiro
☐ Falso
- c)** O uso de *short-tags*, tais como `<? e ?>` é mais benéfico do que o das outras *tags* porque assim perde-se menos tempo em escrever o código.
- ☐ Verdadeiro
☐ Falso
- d)** PHP é mantido por uma comunidade de programadores da internet e está acessível a todas as pessoas.
- ☐ Verdadeiro
☐ Falso
- e)** PHP é uma linguagem de programação compilada.
- ☐ Verdadeiro
☐ Falso
- f)** PHP é uma linguagem de programação *link*-editada.
- ☐ Verdadeiro
☐ Falso
- g)** PHP faz questão do uso de `( ; )` como finalizador de linha de comando.
- ☐ Verdadeiro
☐ Falso
- h)** PHP é uma linguagem que prima por exigir a declaração de tipos de variáveis.
- ☐ Verdadeiro
☐ Falso

i) PHP é uma linguagem, tipo *script*, que executa o programa no seu computador e envia o resultado para um servidor.

() Verdadeiro

() Falso

j) Para executar o PHP, de qualquer jeito, precisa-se de um servidor *web* rodando, como, por exemplo, o Apache.

() Verdadeiro

() Falso

2. O *script* abaixo apresenta vários erros. Indique o erro e a linha correspondente:

```
1 <html>
2
3     <?php
4         Sstatus = "FALSE";
5         $x + $y = $k;
6         $_ = 10 + 3.14;
7         $dia semana = "quinta-feira";
8 45 = $a;
9     $expression= 1
10 </script>
11
12 <body>
13
14 </body>
15 </html>
```

Linha	Erro

Poste os resultados dos exercícios propostos nesta aula no fórum criado para esta finalidade.

Aula 3 – PHP: estruturas condicionais e funções de tempo

Objetivos

Compreender as estruturas condicionais *if ... else* e *switch ... case*.

Conhecer as funções que manipulam data e hora.

3.1 Estrutura condicional *if ... else*.

Declarações condicionais são usadas para executar ações diferentes, baseadas em diferentes condições (MELO; NASCIMENTO, 2007, p. 62-65).

Muitas vezes, quando escrevemos algum código, desejamos executar ações diferentes para diferentes decisões.

Podemos usar a declaração condicional em seu código para fazer isso. Assim, em PHP, temos as seguintes declarações condicionais:

- ***if*** – use esta instrução para executar um código somente se uma condição especificada for verdadeira.
- ***... else if*** – usar essa instrução para executar algum código se uma condição for verdadeira e um outro código se a condição for falsa.
- ***... elseif ... else*** – usar esta instrução para selecionar um dos vários blocos de código a ser executado.
- ***Instrução switch*** – use esta instrução para selecionar um dos muitos blocos de código a ser executado (como veremos no próximo tópico).

3.1.1 A estrutura *if*

Use a instrução ***if*** para executar um código somente se uma condição especificada for verdadeira.

Sintaxe

```
if (condição)
    código a ser executado se condição é verdadeira;
```

O exemplo a seguir irá imprimir “Oba!!! Hoje é sexta-feira!” se o dia atual é uma sexta-feira:

```
<html>
<body>

<?php
$semana = 5; // Número de dias úteis na semana
$mes = 4.5; // Número médio de semanas em um mês
$ano = 12; // Número de meses em um ano
$uteis = ($semana * $mes * $ano); // Calcula o número de dias úteis no ano
if ($uteis >= 270)
    echo “Eh galera, este ano a malhação será grande!”;
?>

</body>
</html>
```

Observe que não existe uma condição FALSA (*else*) nesta sintaxe. O código é executado apenas se a condição especificada for VERDADEIRA.

3.1.2 A declaração *if ... else*

Use a instrução ***if ... else*** para executar algum código se uma condição for verdadeira e um outro código se uma condição for falsa.

Sintaxe

```
if (condição)
    código a ser executado se condição é verdadeira;
else
    código a ser executado se condição é falsa;
```

Exemplo

O exemplo a seguir irá imprimir “Oba!!! Hoje é sexta-feira!” se o dia atual é uma sexta-feira; caso contrário, ele vai de saída “Tenha um bom dia!”:

```
<html>
<body>

<?php
$semana = 5; // Número de dias úteis na semana
$mes = 4.5; // Número médio de semanas em um mês
$ano = 12; // Número de meses em um ano
$uteis = ($semana * $mes * $ano); // Calcula o número de dias úteis em 1
ano
if ($uteis >= 270) // testa se o número de dias úteis calculado é maior ou
igual a 270
echo "Eh galera, este ano a malhação será grande!";
else
echo "Este ano vamos ter moleza.";
?>

</body>
</html>
```

Se mais de uma linha deve ser executada, tanto se a condição for verdadeira ou falsa, as linhas devem ser colocadas dentro de chaves "{" e "}":

```
<html>
<body>

<?php
$semana = 5; // Número de dias úteis na semana
$mes = 4.5; // Número médio de semanas em um mês
$ano = 12; // Número de meses em um ano
$uteis = ($semana * $mes * $ano); // Calcula o número de dias úteis no ano
if ($uteis >= 270) {
    echo "Eh galera, este ano a malhação poderá ser grande!";
    $diferenca = 365 - $uteis;
    echo "<br> E pelo visto, vamos ter apenas $diferenca dias de folga...";
}
else
echo "Este ano vamos ter moleza.";
?>

</body>
</html>
```



Observe que, como no exemplo anterior, podemos inserir *tags* HTML dentro dos comandos *echo*. Elas serão executadas na sua respectiva funcionalidade. No caso acima, uma quebra de linha
 será inserida antes da frase: "E pelo visto, vamos ter apenas 95 dias de folga..."

3.1.3 A instrução *elseif ... else* ...

Use a instrução composta: ***if ... elseif ... else*** para selecionar um dos vários blocos de código a ser executado.

Sintaxe

```
if (condição 1)
    código a ser executado se condição1 é verdadeira;
else
    código a ser executado se condição1 é falsa;
elseif (condição 2)
    código a ser executado se condição2 é verdadeira;
else
    código a ser executado se condição2 é falsa.
```

Exemplo

Conforme o *script* a seguir, será mostrada a mensagem “Eh galera, este ano a malhação será grande!” se o número de dias úteis (**\$uteis**) for maior que 270 dias, ou a mensagem: “Gente, este ano a malhação será normal!”, se for igual a 270, ou “Este ano vamos ter moleza.” se dias úteis não for nem maior (>), nem igual (=) a 270 dias – neste caso, só poderá ser maior (>).

```
<html>
<body>

<?php
$semana = 5; // Número de dias úteis na semana
$mes = 4.5; // Número médio de semanas em um mês
$ano = 12; // Número de meses em um ano
$uteis = ($semana * $mes * $ano); // Calcula o número de dias úteis no ano
if ($uteis > 270)
    echo "Eh galera, este ano a malhação será grande!";
elseif ($uteis == 270)
    echo "Gente, este ano a malhação será normal.";
else
    echo "Este ano vamos ter moleza.";
?>

</body>
</html>
```

3.1.4 Utilizando operador ternário como estrutura de decisão

Outra forma de controle refere-se ao uso do operador ternário (veja operadores de comparação) que pode ser usado da seguinte maneira:

```
<?php
$v = 1 ;
$R = ( 1 == $v )? 'Sim' : 'Não' ; // em outras palavras: if($v ==1) echo 'Sim';
else echo 'Não' // onde $R = 'Sim'
$R = ( 3 == $v )? 'Sim' : 'Não'; // em outras palavras: if($v ==1) echo 'Sim';
else echo 'Não' // onde $R = 'Não'
echo ( 1 == $v )? 'Sim' : 'Não' , // "sim" será impresso
// Outra forma:
$V = "Qualquer Valor" ;
$R = ( $V )? "nenhum valor": "" ;
$V = "" ;
echo ( $V )? "nenhum valor": "" ; // "nenhum valor" será impresso porque
$V é FALSE
?>
```

Como podemos observar no código anterior, ao escrevermos, por exemplo, “(**\$var** == 1)? **\$V** = 0: **\$V** = 1;” significa estabelecer a questão: “Se variável **\$var** é igual a 1, então **\$V** recebe 0; senão, **\$V** recebe 1”.

Como sabemos, é possível ter **if**’s “aninhados” dentro de uma única declaração, utilizando parênteses adicionais. Por exemplo, em vez de termos:

```
<?php
if ( $a == 1 || $a == 2 ) {
    if ( $b == 3 || $b == 4 ) {
        if ( $c == 5 || $d == 6 ) {
            //Bloco de código, aqui.
        }
    }
}
?>
```

Podemos simplesmente escrever:

```
<?php
if (( $a == 1 || $a == 2 ) & & ( $b == 3 || $b == 4 ) & & ( $c == 5 || $c == 6 )) {
    //Bloco de código, aqui.
}
?>
```

3.2 Instrução *switch*

Como dito anteriormente, as declarações condicionais são usadas para executar ações diferentes baseadas em diferentes condições. Com a instrução **switch** não é diferente (MELO, NASCIMENTO, 2007, p. 64). Assim, use essa para selecionar um dos muitos blocos de código para ser executado.

Sintaxe

```
switch(n) {
case label1:
    código a ser executado se n for igual a label1;
    break;
case label2:
    código a ser executado se n for igual a label2;
    break;
default:
    código a ser executado se n for diferente de todas as opções;
}
```

O código anterior é executado da seguinte forma: primeiro nós temos uma única expressão **n** (na maioria das vezes uma variável), que é avaliada uma vez. O valor da expressão é então comparado com os valores para cada caso na estrutura. Se houver uma correspondência, o bloco de código associado a esse processo é executado.

Use o comando **break** para evitar que o próximo código não seja executado automaticamente. A instrução default é usada se nenhuma correspondência nos **cases** for encontrada.



Exemplo

```
<html>
<body>

<?php
switch ($valor) {
case 1:
    echo "valor é igual a 1";
    break;
case 2:
    echo "valor é igual a 2";
    break;
case 3:
    echo "valor é igual a 3";
    break;
default:
    echo "valor é diferente de 1, 2 e 3";
}
?>
</body>
</html>
```

3.3 Funções de data e hora

As funções de data e hora permitem extrair e formatar a data e a hora no servidor. Elas são parte do núcleo do PHP e nenhuma instalação é necessária para as suas utilizações.



Estas funções dependem das configurações locais do servidor! O comportamento das funções de data e hora é afetado pelas configurações no arquivo php.ini.

O Quadro 3.1 descreve as funções data e hora utilizadas em PHP.

Quadro 3.1: Funções de data/hora.	
Função	Descrição
<code>checkdate()</code>	Valida uma data Gregoriana.
<code>date_create()</code>	Retorna um novo objeto <i>DateTime</i> .
<code>date_date_set()</code>	Define a data.
<code>date_format()</code>	Retorna a data formatada de acordo com o formato dado.
<code>date_modify()</code>	Altera o <i>timestamp</i> .
<code>date_parse()</code>	Retorna um <i>array</i> associativo com detalhes sobre uma dada data.
<code>date_sun_info()</code>	Retorna um <i>array</i> com informações sobre pôr do sol/nascer do sol e o início/fim do dia.
<code>date_sunrise()</code>	Retorna a hora em que o sol nasce através dos parâmetros dia e local.
<code>date_sunset()</code>	Retorna a hora em que o sol se põe através dos parâmetros dia e local.
<code>date_time_set()</code>	Define o tempo.
<code>date()</code>	Formata a data e a hora local.
<code>getdate()</code>	Obtém data/hora.
<code>gettimeofday()</code>	Obtém hora/local.
<code>gmdate()</code>	Formata uma data/hora GMT/CUT.
<code>gmmktime()</code>	Obtém um <i>timestamp</i> Unix para uma data GMT.
<code>gmstrftime()</code>	Formata uma hora/data GMT/CUT de acordo com as configurações locais.
<code>idate()</code>	Formata local time/date como inteiro.
<code>localtime()</code>	Obtém a hora local.
<code>microtime()</code>	Retorna um <i>timestamp</i> Unix em microssegundos.
<code>mktime()</code>	Obtém um <i>timestamp</i> Unix para uma data.
<code>sleep()</code>	Atrasa a execução do <i>script</i> em segundos.
<code>strftime()</code>	Formata uma hora/data de acordo com as configurações locais.
<code>time()</code>	Retorna o <i>timestamp</i> Unix atual.
<code>timezone_version_get()</code>	Obtém a versão da <i>timezone</i> .
<code>usleep()</code>	Atrasa a execução do <i>script</i> em milissegundos.

Fonte: Niederauer (2007, p.31-34); http://www.php.net/manual/pt_BR/ref.datetime.php

A seguir, estão exemplificadas três funções de data e hora para ilustrar as funcionalidades desta instrução.

`date()`

A função **`date()`** formata hora e data.

Sintaxe

date(formato)

Acompanhe no Quadro 3.2 as diversas formatações aceitas pela função **date()**.

Quadro 3.2: Parâmetro da função date().

Parâmetro	Descrição
	Parâmetro Obrigatório.
	Especifica como retornar o resultado:
	• d - O dia do mês (01-31). • D - Uma representação textual de um dia (três letras). • j - dia do mês sem zeros (1-31).wl ('L' minúsculo) - Uma representação textual de um dia. • N - A norma ISO-8601 uma representação numérica do dia (1 para segunda-feira a 7 para domingo). • S - O sufixo Inglês ordinal para o dia do mês (2ª caracteres, nd, rd ou th Funciona bem com j). • w - Uma representação numérica do dia (0 para domingo a 6 para sábado). • z - O dia do ano (de 0 a 365). • W - O número de semana ISO-8601 do ano (semanas a partir de segunda-feira). • F - Uma representação textual de um mês (janeiro a dezembro).
Formato	• m - A representação numérica de um mês (de 01 a 12). • M - Uma representação textual curta de um mês (três letras). • n - A representação numérica de um mês, sem zeros (1-12). • t - O número de dias do mês em questão. • L - Se é um ano bissexto (1 se for um ano bissexto, 0 caso contrário). • o - O número do ano ISO-8601. • Y - Uma representação de quatro dígitos de um ano. • Y - Uma representação de dois dígitos do ano. • a - Minúsculas am ou pm. • A - Maiúsculas AM ou PM. • B - <i>Swatch</i> tempo Internet (000-999). • g - Formato 12-horas de uma hora (1-12).
Continua	

- G - Formato 24-horas de uma hora (0-23).
- h - formato 12-horas de uma hora (01 a 12).
- H - formato 24-horas de uma hora (00 a 23).
- i - Minutos com zeros à esquerda (00-59).
- s - Segundos, com zeros à esquerda (00-59).
- e - O fuso horário identificador (Exemplos: UTC, Atlântico / Açores).
- l (i capital) - Se a data está no horário de luz do dia (1 se horário de verão, 0 caso contrário).
- O - Diferença para *Greenwich time* (GMT) em horas (exemplo: 0100)
- T - o fuso horário configuração da máquina PHP (exemplos: EST, MDT).
- Z - o fuso horário deslocamento em segundos. O oeste de UTC é negativo, e o deslocamento leste da UTC é positivo (-43.200-43.200).
- c - A norma ISO-8601 data (por exemplo, 2004-02-12T15: 19:21 +00:00).
- R - O RFC 2822 data formatada (por exemplo, *Thu*, 21 dez 2000 16:01:07 +0200).
- U - o segundo desde a Era Unix (*January 1 1970 00:00:00 GMT*).

Formato

Conclusão

Fonte: Niederauer (2007, p.31)

Exemplo

```
<?php
echo("Operações utilizando date():<br>");
echo(date("l") . "<br>");
echo(date("l dS \of F Y h:i:s A") . "<br>");
echo("28 de Maio de 1961 caiu num(a) ".date("l", mktime(0,0,0,5,28,19
61))."<br>");
echo(date(DATE_RFC822) . "<br>");
echo(date(DATE_ATOM,mktime(0,0,0,5,28,1961)) . "<br><br>");
echo("Operações utilizando gmdate():<br>");
echo(gmdate("l") . "<br>");
echo(gmdate("l dS \of F Y h:i:s A") . "<br>");
echo("28 de Maio de 1961 caiu num(a) ".gmdate("l", mktime(0,0,0,5,28,
1961))."<br>");
echo(gmdate(DATE_RFC822) . "<br>");
echo(gmdate(DATE_ATOM,mktime(0,0,0,5,28,1961)) . "<br>");
?>
```

A saída do código acima poderia ser algo como a listagem abaixo. Acompanhe cada linha de código acima e veja seu efeito e/ou respectiva saída a seguir.

```
Operações utilizando date():
Wednesday
Wednesday 16th of March 2011 12:36:51 PM
28 de Maio de 1961 caiu num(a) Sunday
Wed, 16 Mar 11 12:36:51 -0300
1961-05-28T00:00:00-03:00
```

```
Operações utilizando gmdate():
Wednesday
Wednesday 16th of March 2011 03:36:51 PM
28 de Maio de 1961 caiu num(a) Sunday
Wed, 16 Mar 11 15:36:51 +0000
1961-05-28T03:00:00+00:00
```

Observamos no *script* acima que as funções *gmdate()* e *date()* são similares.

A seguir são exemplificadas duas formas de utilização da instrução de data e hora.

microtime()

A ***microtime()*** retorna o **timestamp** atual em microssegundos.

Sintaxe

```
$s=microtime(get_as_float)
```

O Quadro 3.3, a seguir, apresenta o único parâmetro opcional com a respectiva funcionalidade da função (método) *microtime()*.

Quadro 3.3: Parâmetro da função *microtime()*.

Parâmetro	Descrição
<i>get_as_float</i>	Opcional. Quando definido para <i>true</i> , especifica que a função deve retornar um <i>float</i> , em vez de uma sequência de números inteiros.

Fonte: Niederauer (2007, p. 33)

A-Z

Timestamp

É o momento em que um evento é registrado por um computador e não o tempo do evento em si. É representado em segundos passados desde 1/1/1970.

Se o opcional *get_as_float* é definido para TRUE então um *float* (em segundos) é retornado.



O parâmetro ***get_as_float*** foi adicionado ao PHP 5!

Exemplo

```
<?php
// Captura o tempo neste instante
$time_start = microtime(true);

// Aguarda 1s (microsegundo ~= 0,000001)
usleep(1000000);

// Captura o tempo apos 1 microsegundo
$time_end = microtime(true);

// Mostra a diferença entre a captura do tempo anterior e o atual
$time = $time_end - $time_start;
echo "Já se passaram $time segundos\n";
?>
```

A saída do código acima poderia ser:

```
Já se passaram 1.0000171661377 segundos
```

time()

A função ***time()*** retorna o tempo atual como um *timestamp* Unix.

Sintaxe

```
time(void)
```

Chamar essa função é idêntica a chamar a função ***mktime()*** sem parâmetros, ou a data com parâmetro "U".

Exemplo

```
<?php
$t=time(); // time() retorna o número de segundos desde 01/01/1970 até
data/hora atuais
echo $t . "<br />";
//Traduz o dia da semana (Inglês/Português)
switch(date("D")) {
case "Sun":
    $semana = "Domingo "; break;
case "Mon":
    $semana = "Segunda-feira "; break;
case "Tue":
    $semana = "Terça-feira "; break;
case "Wed":
    $semana = "Quarta-feira "; break;
case "Thu":
    $semana = "Quinta-feira "; break;
case "Fri":
    $semana = "Sexta-feira "; break;
case "Sat":
    $semana = "Sábado "; break;
}
echo $semana . (date("d F Y",$t));
?>
```

A saída do código acima poderia ser:

```
1300446023
Sexta-feira 18 March 2011
```

Note que o nome do mês ficou em inglês. Entretanto, se desejar, pode-se traduzir, bastando para isto criar uma estrutura **switch-case**, como a construída para os dias da semana.

Resumo

O objetivo desta aula foi tratar sobre as estruturas de repetição e do objeto de data e hora. Aprendemos as estruturas condicionais, **if ... else** e **switch ... case**. Estudamos como utilizá-las e estruturá-las através de exemplos e visualizações dos respectivos resultados.

Tivemos a oportunidade de criar objetos do tipo data e visualizar a maioria de suas funções. Estudamos sobre como criar instâncias de data e hora e manipulá-las conforme nossas necessidades. Vimos, também, as diversas atribuições e atributos de operações dos objetos.

Atividades de aprendizagem

1. Digite todos os códigos exemplos desta aula e execute-os comparando os resultados obtidos com as respectivas saídas aqui ilustradas. Além disso, poste os códigos digitados no fórum criado para esta atividade, bem como um breve relatório descrevendo os resultados das suas execuções.
2. Utilizando a estrutura de decisão **if ... else**, crie um programa em PHP que analise quatro variáveis numéricas (**\$a**, **\$b**, **\$c** e **\$d**, por exemplo) e mostre seus respectivos conteúdos (utilizando instrução **echo**) em ordem crescente, independentemente do valor que cada variável possa ter.
3. Atribua três valores para as variáveis **\$a**, **\$b** e **\$c**. Esses valores representam os coeficientes da equação de segundo grau dada por: $ax^2 + bx + c$. Elabore um *script* em PHP que, inicialmente, verifique se os valores lidos formam uma equação de segundo grau ($a \neq 0$), e, posteriormente, se formarem, calcule as respectivas raízes da equação, mostrando-as com o comando **echo**.
4. Atribua três valores para as variáveis **\$a**, **\$b** e **\$c**. Esses valores representam os comprimentos dos lados de um triângulo. Crie um programa em PHP que verifique se estes valores formam um triângulo. Para formar um triângulo, um lado não pode ser maior que a soma dos outros dois. Depois, verifique que tipo de triângulo é:
 - a) Equilátero = três lados iguais;
 - b) Isósceles = dois lados iguais e um diferente;
 - c) Escaleno = três lados diferentes.

5. Escolha cinco funções do quadro de funções data/hora que não foram citadas ainda nos exemplos descritos nesta aula e escreva um *script* em PHP para cada função escolhida e que demonstre as respectivas funcionalidades.
6. Crie um *script*, utilizando estruturas condicionais, que retorne o dia da semana **em português** tendo como referência a data de hoje.

Poste suas respostas no fórum criado para estas atividades no ambiente virtual de ensino-aprendizagem.

Aula 4 – PHP: estruturas de repetição e funções externas

Objetivos

Entender as estruturas de repetição **while** e **for**.

Conhecer a criação de funções externas.

4.1 A estrutura de repetição

O comando **while** executa um bloco de código um número especificado de vezes ou enquanto uma condição especificada for verdadeira (MELO; NASCIMENTO, 2007, p. 65-69).

Normalmente, quando escrevemos blocos de códigos, desejamos que eles sejam executados várias vezes seguidas. Em vez de adicionar várias linhas iguais e repetidas em um *script*, podemos usar *loops* para realizar uma tarefa como essa, em poucas linhas.

Em PHP, temos as seguintes declarações de *looping*:

- **while** – percorre um bloco de código enquanto uma condição especificada for verdadeira.
- **do ... while** – percorre um bloco de código uma vez e depois repete o ciclo enquanto a condição especificada for verdadeira.
- **for** – percorre um bloco de código um determinado número de vezes.
- **foreach** – percorre um bloco de código para cada elemento em uma matriz.

4.1.1 O laço *while*

O laço **while** executa um bloco de código enquanto uma condição for verdadeira.

Sintaxe

```
while (condição) {  
    código a ser executado;  
}
```

Exemplo

O exemplo a seguir define um *loop* que começa com $i = 1$, aumentando de 1 em 1, termina com $i = 10$.

```
<html>  
<body>  
  
<?php  
$i=1;  
while($i <= 10) {  
    echo "linha " . $i . "<br>";  
    $i++;  
}  
?>  
  
</body>  
</html>
```

A saída será:

```
linha 1  
linha 2  
linha 3  
linha 4  
linha 5  
linha 6  
linha 7  
linha 8  
linha 9  
linha 10
```

4.1.2 O laço **do ... while**

O laço **do ... while** sempre irá executar o bloco de código uma primeira vez. Em seguida ele irá verificar a condição e repetirá o *loop* enquanto a condição for verdadeira.

Sintaxe

```
do {  
    Código a ser executado;;  
}  
while (condição);
```

Exemplo

O *script* a seguir define um *loop* que começa com **\$i** = 1. Assim, a saída "linha1" será mostrada e as demais linhas serão mostradas enquanto a condição for verdadeira.

```
<html>  
<body>  
  
<?php  
$i=1;  
do {  
    $i++;  
    echo "linha " . $i . "<br>";  
}  
while ($i<=10);  
?>  
  
</body>  
</html>
```

A saída será:

```
linha 1  
linha 2  
linha 3  
linha 4  
linha 5  
linha 6  
linha 7  
linha 8  
linha 9  
linha 10
```

4.1.3 O laço *for*

O laço ***for*** executa um bloco de código um número especificado de vezes e é usado quando você sabe de antemão quantas vezes o *script* deve ser executado.

Sintaxe

```
for (inicialização; condição; incremento) {  
  código a ser executado;  
}
```

Parâmetros (separados por vírgula):

- *inicialização*: geralmente usado para definir um contador (inicialização da variável de controle)
- *condição*: avalia para cada iteração do *loop*. Se for avaliado como *TRUE*, o *loop* continuará a ser executado. Se for avaliado como *FALSE*, o *loop* termina.
- *incremento*: geralmente usado para incrementar um contador (mas pode ser qualquer código a ser executado no final do *loop*).



Cada um dos parâmetros ao lado pode ser vazio ou ter múltiplas expressões.

Exemplo

O próximo *script* define um *loop* que começa com **\$i** = 1. O *loop* vai continuar a funcionar enquanto **\$i** for inferior ou igual a 10. O contador **\$i** vai aumentar de 1 em 1ª cada iteração ou *looping*, conforme exemplo ilustrado a seguir:

```
<html>
<body>

<?php
for ($i=1; $i<=10; $i++) {
    echo "linha " . $i . "<br>";
}
?>

</body>
</html>
```

A saída será:

```
linha 1
linha 2
linha 3
linha 4
linha 5
linha 6
linha 7
linha 8
linha 9
linha 10
```

4.2 Funções externas

Existe uma infinidade de funções já construídas no PHP, como veremos nas próximas aulas e o verdadeiro poder do PHP vem de suas funções. Nele, existem mais de 700 funções embutidas.

Entretanto, nesta aula, vamos mostrar como criar suas próprias funções (MELO; NASCIMENTO, 2007, p. 69-72; GILMORE, 2008, p. 89-99).



Funções complementares podem ser visualizadas através do *site* oficial do PHP em http://br.php.net/manual/pt_BR

4.2.1 A criação de funções

As funções criadas pelos programadores são normalmente executadas por uma chamada direta a elas em qualquer parte do corpo da página.

Sintaxe

```
function functionName() {  
    //código a ser executado;  
}
```

Orientações sobre a criação de funções no PHP:

- dê um nome à função que reflita o que ela faz;
- o nome da função pode começar com uma letra ou um sublinhado (não um número);
- nomes de funções não devem conter caracteres especiais, tais como: acentos e pontuações.

Exemplo

Uma função que escreve um nome quando ela é chamada:

```
<html>  
<body>  
  
<?php  
function meuNome() {  
    echo " William Sallum";  
}  
  
echo "Meu nome é: ";  
meuNome();  
?>  
  
</body>  
</html>
```

A saída respectiva será:

```
Meu nome é: William Sallum
```

4.2.2 Adicionando parâmetros às funções

Para adicionar mais funcionalidade à função, nós podemos adicionar parâmetros. Um parâmetro é como uma variável. Os parâmetros são especificados depois do nome da função, dentro dos parênteses, tal como visto nos exemplos a seguir.

Exemplo 1

No exemplo a seguir são escritos vários objetos na cor azul, com a passagem de um parâmetro:

```
<html>
<body>

<?php
function meusObjetos($obj) {
    if ($obj != "")
        echo $obj . " é Azul.<br>";
    else
        echo "Não foi informado o objeto!";
}
$a = "O meu carro ";
meusObjetos ($a);
$a = "Minha casa";
meusObjetos ($a);
$a = "Meu time ";
meusObjetos ($a);
?>

</body>
</html>
```

A saída será:

```
O meu carro é Azul.  
Minha casa é Azul.  
Meu time é Azul.
```

Entretanto, se houver alguma chamada, a função **meusObjetos()** com algum conteúdo vazio, a mensagem: "Não foi informado o objeto!" será mostrada.

Exemplo 2

A seguinte função trabalha com o recebimento de dois parâmetros. Veja o exemplo.

```
<html>  
<body>  
  
<?php  
function areaRet($base,$altura) {  
    $area = $base * $altura / 2;  
    echo "O retângulo com base=$base e altura=$altura tem área  
    ".$area."<br>";  
}  
areaRet (5,4);  
areaRet (7,3);  
areaRet (12.5,6);  
  
?>  
  
</body>  
</html>
```

A saída será:

```
O retângulo com base=5 e altura=4 tem área 10  
O retângulo com base=7 e altura=3 tem área 10.5  
O retângulo com base=12.5 e altura=6 tem área 37.5
```

Para deixar uma função retornar um valor, vamos utilizar a instrução **return**, conforme o exemplo a seguir.

Exemplo 3

```
<html>
<body>

<?php
function somaDosNumeros($numero) {
    if($numero <= 0)
        return "Erro: o número deve ser maior que 0!";
    $total = 0;
    for($i = 1; $i <= $numero; $i++)
        $total += $i;
    return $total;
}
$n = 7;
echo "Somatório de $n = " . somaDosNumeros($n);
?>

</body>
</html>
```

É claro que no nosso exemplo o número passado como parâmetro é positivo, mas para o caso de esse *script* ser adaptado para receber um número qualquer digitado pelo usuário, os testes de positividade e de nulidade devem ser feitos através do comando **if** que, caso seja verdadeiro, retornará a mensagem: "Erros: o número deve ser maior que 0!". Mas, se o número é positivo (e no nosso exemplo ele é: 7), o resultado da execução do *script* será:

Somatório de 7 = 28

Ou seja, dentro da função **somaDosNumeros** o *looping* efetuará os seguintes passos até chegar à resposta final: 1 + 2 + 3 + 4 + 5 + 6 + 7, que é igual a 28.

Resumo

Aprendemos nesta aula sobre como estruturar e operar as estruturas de repetição tais como: o laço (ou *looping*) **while**, que pode manter iterações mediante uma condição lógica verdadeira; do laço **do ... while**, semelhante à estrutura **while**, diferenciando apenas no fato que na laço **do ... while**, sempre será executada a primeira iteração independentemente da condição lógica ser ou não verdadeira. Aprendemos sobre o laço **for**, que executa iterações finitas, mediante início e fim numericamente determinados.

Estudamos também como criar funções personalizadas, conforme necessidades estabelecidas pelo programador. Vimos como trabalhar nas assinaturas (estrutura única que identifica cada função) das funções. Conhecemos a técnica de criar funções com ou sem parâmetro de entrada, bem como seu corpo de execução e os retornos de valores como resultado da operação dos parâmetros de entrada e das instruções do corpo da função.

Atividades de aprendizagem

1. Digite todos os códigos exemplos desta aula e execute-os comparando os resultados obtidos com as respectivas saídas aqui ilustradas. Além disso, poste os códigos digitados no fórum criado para esta atividade, bem como um breve relatório descrevendo os resultados das suas execuções.
2. Faça uma função PHP que receba um número inteiro, calcule e imprima a tabuada multiplicativa desse número.
3. Crie uma função em PHP que, utilizando estrutura de repetição, receba como parâmetro um número inteiro positivo qualquer, apure todos os números primos que existirem entre esse número e o número 1 e mostre-os.

Poste suas respostas no fórum para estas atividades no ambiente virtual de ensino-aprendizagem.

Aula 5 – PHP: funções matemáticas e matrizes

Objetivo

Conhecer as funções matemáticas e matrizes.

5.1 Funções matemáticas

As funções matemáticas podem lidar com valores dentro da faixa de tipos inteiros e *float*. Estas funções fazem parte do núcleo do PHP (internos) e não há nenhuma instalação necessária para utilizar estas funções.

A seguir, conforme apresentado no Quadro 5.1, foram relacionadas as funções mais comumente utilizadas.

Quadro 5.1: Funções matemáticas do PHP.		
Função	Descrição	Versão PHP
<i>abs()</i>	Retorna o valor absoluto de um número.	3
<i>acos()</i>	Retorna o arco cosseno de um número.	3
<i>acosh()</i>	Retorna o cosseno hiperbólico inverso de um número .	4
<i>asin()</i>	Retorna o arco seno de um número.	3
<i>asinh()</i>	Retorna o seno hiperbólico inverso de um número.	4
<i>atan()</i>	Retorna o arco tangente de um número como um valor numérico entre $-\pi/2$ e $+\pi/2$ radianos.	3
<i>atan2()</i>	Retorna o ângulo theta de um ponto (x, y) como um valor numérico entre $-\pi$ e $+\pi$ radianos.	3
<i>atanh()</i>	Retorna a tangente hiperbólica inversa de um número.	4
<i>base_convert()</i>	Converte um número de uma base para outra.	3
<i>bindec()</i>	Converte um número binário para um número decimal.	3
<i>ceil()</i>	Retorna o valor de um número arredondado para o número inteiro mais próximo.	3
<i>cos()</i>	Retorna o cosseno de um número.	3
<i>cosh()</i>	Retorna o cosseno hiperbólico de um número.	4
<i>decbin()</i>	Converte um número decimal para um número binário.	3
<i>dechex()</i>	Converte um número decimal para um número hexadecimal.	3
<i>decoct()</i>	Converte um número decimal para um número octal.	3
<i>deg2rad()</i>	Converte graus em radiano.	3
<i>exp()</i>	Retorna o valor de E^x .	3
<i>expm1()</i>	Retorna o valor de $E^x - 1$.	4
Continua		

<i>floor()</i>	Retorna o valor de um número arredondado para baixo para o número inteiro mais próximo.	3
<i>fmod()</i>	Retorna o resto (módulo) de uma divisão.	4
<i>hexdec()</i>	Converte um número hexadecimal em decimal.	3
<i>hypot()</i>	Retorna o comprimento da hipotenusa de um triângulo retângulo.	4
<i>is_finite()</i>	Retorna <i>true</i> se um valor é um número finito.	4
<i>is_infinite()</i>	Retorna <i>true</i> se um valor é um número infinito.	4
<i>is_nan()</i>	Retorna <i>true</i> se um valor não é um número.	4
<i>lcg_value()</i>	Retorna um número pseudoaleatório no intervalo de (0,1).	4
<i>log()</i>	Retorna o logaritmo natural (base e) de um número.	3
<i>log10()</i>	Retorna o logaritmo de base 10 de um número.	3
<i>max()</i>	Retorna o número com o valor mais elevado de dois números especificados.	3
<i>min()</i>	Retorna o número com o menor valor de dois números especificados.	3
<i>mt_getrandmax()</i>	Retorna o maior valor possível que pode ser devolvido por <i>mt_rand()</i> .	3
<i>octdec()</i>	Converte um número octal em um número decimal.	3
<i>pi()</i>	Retorna o valor de PI.	3
<i>pow()</i>	Retorna o valor de x para a potência de y.	3
<i>rad2deg()</i>	Converte um número em radianos para um grau.	3
<i>rand()</i>	Retorna um inteiro aleatório.	3
<i>round()</i>	Arredonda um número para o inteiro mais próximo.	3
<i>sin()</i>	Retorna o seno de um número.	3
<i>sinh()</i>	Retorna o seno hiperbólico de um número.	4
<i>sqrt()</i>	Retorna a raiz quadrada de um número.	3
<i>srand()</i>	Semeia o gerador de números aleatórios.	3
<i>tan()</i>	Retorna a tangente de um ângulo.	3
<i>tanh()</i>	Retorna a tangente hiperbólica de um ângulo.	4
Conclusão		

Fonte Niederauer (2007, p.69-72); http://www.php.net/manual/pt_BR/book.math.php

O Quadro 5.2 apresenta a relação das constantes matemáticas disponibilizadas pelo PHP.

Quadro 5.2: Constantes matemáticas do PHP.		
Constante	Descrição	Versão PHP
<i>M_E</i>	Retorna e (aprox. 2,718).	4
<i>M_EULER</i>	Retorna a constante de Euler (aproximadamente 0,577).	4
<i>M_LNPI</i>	Retorna o logaritmo natural de PI (aproximadamente 1,144).	4
<i>M_LN2</i>	Retorna o logaritmo natural de 2 (aproximadamente 0,693).	4
<i>M_LN10</i>	Retorna o logaritmo natural de 10 (aprox. 2,302).	4
<i>M_LOG2E</i>	Retorna o logaritmo de base 2 de E (aproximadamente 1,442).	4
<i>M_LOG10E</i>	Retorna o logaritmo de base 10 de E (aproximadamente 0,434).	4
<i>M_PI</i>	Retorna PI (aproximadamente 3,14159).	3
Continua		

M_PI_2	Retorna PI / 2 (aproximadamente 1,570).	4
M_PI_4	Retorna PI / 4 (cerca de 0,785).	4
M_1_PI	Retorna 1/PI (aproximadamente 0,318).	4
M_2_PI	Retorna 2/PI (aproximadamente 0,636).	4
M_SQRTPI	Retorna a raiz quadrada de PI (aproximadamente 1,772).	4
M_2_SQRTPI	Retorna 2/ raiz de PI (aproximadamente 1,128).	4
M_SQRT1_2	Retorna a raiz quadrada de 1/2 (aproximadamente 0,707).	4
M_SQRT2	Retorna a raiz quadrada de 2 (aprox. 1,414).	4
M_SQRT3	Retorna a raiz quadrada de 3 (aprox. 1,732).	4
		Conclusão

Fonte: http://www.php.net/manual/pt_BR/math.constants.php

A seguir alguns exemplos de uso de funções matemáticas:

• **Abs()**

A função **abs()** retorna o valor absoluto de um número. E a funcionalidade de seu parâmetro é apresentada no Quadro 5.3.

Sintaxe

abs(x)

Quadro 5.3: Parâmetro da função abs() .	
Parâmetro	Descrição
x	Obrigatório. Valor numérico. Se o número é do tipo <i>float</i> , o tipo de retorno também é <i>float</i> , se não, retorna inteiro.

Fonte: Niederauer (2007, p. 69)

Exemplo

```
<?php
echo(abs(3.14) . "<br>");
echo(abs(-4) . "<br>");
echo(abs(7));
?>
```

A saída do código acima será:

```
3.14
4
37
```

- ***deg2rad()***

A função ***deg2rad()*** converte graus em radianos e seu parâmetro é apresentado no Quadro 5.4 a seguir.

Sintaxe

`deg2rad(número de graus)`

Quadro 5.4: Parâmetro da função *deg2rad()*.

Parâmetro	Descrição
Número de graus	Obrigatório. Especificar o grau a converter.

Fonte: Niederauer (2007, p.70)

Exemplo 1

```
<?php
echo deg2rad("30") . "<br>";
echo deg2rad("45") . "<br>";
echo deg2rad("60") . "<br>";
echo deg2rad("360");
?>
```

A saída do código acima será:

```
0.5235987755983
0.78539816339745
1.0471975511966
6.2831853071796
```

Exemplo 2

```
<?php
$g = 90;
$r = deg2rad($g);
echo "$g Graus = $r radianos";
?>
```

A saída do código acima será:

```
90o Graus = 1.5707963267949 radianos
```

• **cos()**

A função trigonométrica **cos()**, cujo único parâmetro apresentado no Quadro 5.5, retorna o cosseno de um número.

Sintaxe

`cos(x)`

Quadro 5.5: Parâmetro da função cos() .	
Parâmetro	Descrição
x	Obrigatório. Um número.

Fonte: Niederauer (2007, p.70)

Observe que:

1. A função **cos()** retorna um valor numérico entre -1 e 1, que representa o cosseno do ângulo.
2. As funções trigonométricas recebem como parâmetros números radianos.



Exemplo

A seguir iremos retornar o cosseno de números diferentes:

```
<?php
echo(cos(1) . "<br>");
echo(cos(-1) . "<br>");
echo(cos(0) . "<br>");
echo(cos(M_PI) . "<br>");
echo(cos(2*M_PI)) // vide quadro das constantes matemáticas
?>
```

A saída do código acima será:

```
0.54030230586814
0.54030230586814
1
-1
1
```

- ***sqrt()***

A função ***sqrt()***, com seu respectivo parâmetro, apresentado no Quadro 5.6, retorna a raiz quadrada de um número.

Sintaxe

`sqrt(x)`

Quadro 5.6: Parâmetro da função *sqrt()*.

Parâmetro	Descrição
r	Obrigatório. Um número qualquer ≥ 0 .

Fonte: Niederauer (2007, p.72)



A função ***sqrt()*** irá retornar “-1.#IND” ou “NAN” se o parâmetro **r** for um número negativo.

Exemplo

Vamos obter a raiz quadrada de números diferentes:

```
<?php
echo(sqrt(0) . "<br>");
echo(sqrt(16) . "<br>");
echo(sqrt(27) . "<br>");
echo(sqrt(0.64) . "<br>");
echo(sqrt(-5))
?>
```

A saída do código acima será:

```
0
4
5.1961524227066
0.8
NAN
```

- ***pow()***

A função ***pow(b,e)*** submete uma base **b** à potência **e**, cujos parâmetros podem ser vistos no Quadro 5.7 a seguir.

Sintaxe

`pow(b,e)`

Quadro 5.7: Parâmetros da função `pow()`.

Parâmetro	Descrição
b	Obrigatório. É o número base da potência.
e	Obrigatório. É o expoente.

Fonte: Niederauer (2007, p.72)

Exemplo

```
<?php
echo pow(2,2) . "<br>";
echo pow(3,2) . "<br>";
echo pow(-3,2) . "<br>";
echo pow(-5,-2) . "<br>";
echo pow(16,0.5); //Obs.: qualquer número inteiro elevado a 0.5 = sua raiz
quadrada
?>
```

5.2 Matrizes

Um *array* é uma matriz ou vetor que pode armazenar vários valores em uma única variável.

Se você tem uma lista de itens (de autores de livros, por exemplo), armazenando os livros em simples variáveis, poderia ser semelhante a esta:

```
$liv1 = "Drummond";
$liv2 = "Bandeira";
$liv3 = "Lispector";
```

Uma matriz pode conter todos os valores variáveis sob um único nome. E você pode acessar os valores referentes ao nome da matriz indicando apenas o seu índice ou a posição em que ele se encontra dentro dessa matriz.

Cada elemento da matriz tem o seu próprio índice, para que possa ser facilmente acessado.

No PHP, existem três tipos de matrizes:

- **matriz numérica** – uma matriz com um índice numérico.
- **matriz associativa** – uma matriz na qual cada chave ID está associada a um valor.
- **matriz multidimensional** – uma matriz contendo um ou mais *arrays*

5.2.1 Matrizes numéricas

Matrizes numéricas armazenam cada elemento com um índice numérico. Existem dois métodos para criar uma matriz numérica:

1. No exemplo a seguir, os índices são automaticamente atribuídos (o índice começa em 0):

```
$liv = array("Drummond", "Bandeira", "Lispector", "Oliveira");
```

2. No exemplo a seguir, vamos atribuir o índice manualmente:

```
$liv[0] = "Drummond";
```

```
$liv[1] = "Bandeira";
```

```
$liv[2] = "Lispector";
```

```
$liv[3] = "Oliveira";
```

Exemplo

Você acessa os valores por referência ao nome da matriz e do índice:

```
<?php
$liv[0]="Drummond";
$liv[1]="Bandeira";
$liv[2]="Lispector";
$liv[3]="Oliveira";
echo $liv[0] . ", " . $liv[1] . " e " . $liv[3]. " são autores homens, enquanto "
.liv[2]. " é uma autora";
?>
```

O código acima irá mostrar:

Drummond, Bandeira e Oliveira são autores homens, enquanto Lispector é uma autora.

5.2.2 Matrizes associativas

Em um **array** associativo, cada chave ID está associada a um valor.

Armazenar dados específicos sobre valores associativos: uma matriz numérica nem sempre é a melhor maneira de fazê-lo.

Com **arrays** associativos, podemos usar os valores como chaves de busca e atribuir-lhes valores.

Exemplo 1

Usamos uma matriz de cidades atribuídas aos respectivos estados:

```
<?php
    $cidades = array("MG"=>"Belo Horizonte", "PR"=>"Curitiba",
    "PE"=>"Recife");
    echo $cidades['PR'];
?>
```

Exemplo 2

Este exemplo é o mesmo do exemplo 1, mas mostra uma maneira diferente de criação da matriz:

```
$cidades['MG'] = "Belo Horizonte";
$cidades['PR'] = "Curitiba";
$ages['JPE'] = "Recife";
```

As chaves de identificação podem ser usadas em um *script*:

```
<?php
$cidades['MG'] = "Belo Horizonte";
$cidades['PR'] = "Curitiba";
$ages['JPE'] = "Recife";
echo "A capital de PR é: " . $ages['PR'];
?>
```

O código acima irá mostrar:

```
Curitiba
```

5.2.3 Matrizes multidimensionais

Em uma matriz multidimensional, cada elemento na matriz principal também pode ser um *array*. E cada elemento no *sub-array* pode ser uma matriz, e assim por diante.

Exemplo 1

Vamos criar uma matriz multidimensional, com chaves de identificação atribuídas automaticamente:

```
$familias = array ("Sallum"=>array ("Mayumi","Seiji","Harumi" ),  
"Silva"=>array ("José" ,"Maria"));
```

O *array* acima, ao ser impresso, ficaria conforme abaixo:

```
array (  
[Sallum] => array (  
[0] => Mayumi  
[1] => Seiji  
[2] => Harumi )  
[Silva] => array (  
[0] => José  
[1] => Maria ) )
```

Exemplo 2

Vamos tentar apresentar um único valor a partir da matriz acima:

```
echo "A " . $familias['sallum'][1] . " é parte da família Sallum.";
```

O código acima irá imprimir:

```
A Harumi é parte da família Sallum.
```

As funções de matriz permitem manipular matrizes e, como vimos, o PHP suporta *arrays* simples e multidimensional. Há também funções específicas para preencher *arrays* de consultas a banco de dados.

Assim como as outras funções já faladas aqui, as funções de matriz são parte do núcleo do PHP e não necessitam de nenhuma instalação para utilizá-las.

O Quadro 5.8, a seguir, relaciona as funções de *array* mais comumente utilizadas.

Quadro 5.8: Funções de <i>array</i> do PHP		
Função	Descrição	Versão
<code>array()</code>	Cria uma matriz.	3
<code>array_change_key_case()</code>	Retorna uma matriz com todas as chaves em maiúsculas ou minúsculas.	4
<code>array_chunk()</code>	Divide um <i>array</i> em pedaços de matrizes.	4
<code>array_combine()</code>	Cria um <i>array</i> usando um <i>array</i> para chaves e outro para valores.	5
<code>array_count_values()</code>	Retorna uma matriz com o número de ocorrências para cada valor.	4
<code>array_diff()</code>	Compara a matriz de valores e retorna as diferenças.	4
<code>array_diff_assoc()</code>	Compara as chaves de matriz e valores, e retorna as diferenças.	4
<code>array_diff_key()</code>	Compara as chaves de matriz, e retorna as diferenças.	5
<code>array_key_exists()</code>	Verifica se a chave especificada existe na matriz.	4
<code>array_keys()</code>	Retorna todas as chaves de um <i>array</i> .	4
<code>array_merge()</code>	Funde um ou mais <i>arrays</i> em um <i>array</i> .	4
<code>array_merge_recursive()</code>	Funde um ou mais <i>arrays</i> em um <i>array</i> .	4
<code>array_product()</code>	Calcula o produto dos valores de um <i>array</i> .	5
<code>array_push()</code>	Insere um ou mais elementos no final de uma matriz.	4
<code>array_rand()</code>	Retorna um ou mais chaves aleatórias de uma matriz.	4
<code>array_reduce()</code>	Retorna uma matriz como uma string, usando uma função definida pelo usuário.	4
<code>array_reverse()</code>	Retorna um <i>array</i> na ordem inversa.	4
<code>array_search()</code>	Procura um <i>array</i> para um determinado valor e retorna a chave.	4
<code>array_shift()</code>	Remove o primeiro elemento de uma matriz e retorna o valor do elemento removido.	4
<code>array_slice()</code>	Retorna partes selecionadas de um <i>array</i> .	4
<code>array_splice()</code>	Remove e substitui elementos específicos de um <i>array</i> .	4
<code>array_sum()</code>	Retorna a soma dos valores de um <i>array</i> .	4
<code>array_unique()</code>	Remove o valores duplicados de um <i>array</i> .	4
<code>array_unshift()</code>	Adiciona um ou mais elementos ao início de um <i>array</i> .	4
<code>array_values()</code>	Retorna todos os valores de um <i>array</i> .	4
<code>arsort()</code>	Ordena um <i>array</i> em ordem decrescente mantendo a associação entre o índice.	3
<code>asort()</code>	Ordena um <i>array</i> mantendo a associação entre índices.	3
<code>count()</code>	Conta os elementos de um <i>array</i> , ou propriedades de um objeto.	3
<code>current()</code>	Retorna o elemento corrente em um <i>array</i> .	3
Continua		

<i>each()</i>	Retorna o par chave/valor corrente de um <i>array</i> .	3
<i>end()</i>	Define o ponteiro interno de um <i>array</i> para o seu último elemento.	3
<i>extract()</i>	Importações de variáveis para a tabela de símbolos a partir de uma matriz.	3
<i>in_array()</i>	Verifica se um valor especificado em uma matriz.	4
<i>key()</i>	Retorna uma chave de um <i>array</i> .	3
<i>krsort()</i>	Ordena um <i>array</i> pelas chaves em ordem decrescente.	3
<i>ksort()</i>	Classifica um <i>array</i> pelas chaves.	3
<i>list()</i>	Atribui variáveis como se fossem uma matriz.	3
<i>natcasesort()</i>	Ordena um <i>array</i> utilizando o case-sensitive "ordem natural" algoritmo.	4
<i>natsort()</i>	Ordena um <i>array</i> utilizando uma "ordem natural" algoritmo.	4
<i>next()</i>	Avança o ponteiro interno de um <i>array</i> .	3
<i>pos()</i>	Sinônimo de "current()".	3
<i>prev()</i>	Retrocede o ponteiro interno do <i>array</i> .	3
<i>range()</i>	Cria um <i>array</i> contendo uma faixa de elementos.	3
<i>reset()</i>	Define o ponteiro interno de um <i>array</i> para o primeiro elemento.	3
<i>rsort()</i>	Ordena um <i>array</i> em ordem inversa.	3
<i>shuffle()</i>	Embaralha um <i>array</i> .	3
<i>sizeof()</i>	Sinônimo de <i>count()</i> .	3
<i>sort()</i>	Ordena um <i>array</i> .	3
<i>uasort()</i>	Ordena um <i>array</i> utilizando uma função definida pelo usuário e manter a associação entre índices.	3
<i>uksort()</i>	Ordena um <i>array</i> pelas chaves utilizando uma função definida pelo usuário.	3
<i>usort()</i>	Ordena um <i>array</i> pelos valores utilizando uma função definida pelo usuário.	3
Conclusão		

Fonte: Niederauer (2007, p. 18-24); Melo e Nascimento (2007, p.75-79) http://www.php.net/manual/pt_BR/book.array.php

A seguir, serão exemplificadas quatro funções de *array* para ilustrar a utilização desta instrução:

- ***array_search()***

A função ***array_search()***, cujos parâmetros encontram-se listados no Quadro 5.9, efetua pesquisa de uma matriz através de um valor e retorna a sua respectiva chave.

Sintaxe

`array_search(valor,array,obs)`

Quadro 5.9: Parâmetros da função `array_search()`.

Parâmetro	Descrição
valor	Obrigatório. Especifica o valor a pesquisar.
matriz	Obrigatório. Especifica a matriz em que será feita a pesquisa.
Obs.	Opcional. Valores possíveis: • <i>true</i> • <i>false</i> - Padrão Quando definido como <i>true</i> , o número 5 será tomado tal como informado (quanto a sua tipologia) (ver exemplo 2).

Fonte: Niederauer (2007, p. 20)

Exemplo 1

```
<?php
$chave=array("1"=>"Amarelo","2"=>"Azul","3"=>"Branco");
echo array_search("Amarelo",$chave);
?>
```

A saída do código acima será:

1

Exemplo 2

```
<?php
$chave=array("1"=>"9","2"=>9,"3"=>"9");
echo array_search(9,$chave,true);
?>
```

A saída do código acima será:

2

- **`array_reverse()`**

A função **`array_reverse()`** retorna um *array* na ordem inversa. Os respectivos parâmetros são relacionados no Quadro 5.10 a seguir.

Sintaxe

`array_reverse(matriz,preserva)`

Quadro 5.10: Parâmetros da função `array_reverse()`.

Parâmetro	Descrição
matriz	Obrigatório. Especifica uma matriz.
preserva	Opcional. Os valores possíveis: • <i>true</i> • <i>false</i> Especifica se a função deve preservar as chaves do <i>array</i> ou não.

Fonte: Niederauer (2007, p. 20)

Exemplo 1

```
<?php
$chave=array("1"=>"Amarelo","2"=>"Azul","3"=>"Branco");
print_r(array_reverse($chave));
?>
```

A saída do código acima será:

```
Array (
[0] => Branco
[1] => Azul
[2] => Amarelo )
```

Exemplo 2

```
<?php
$chave=array("1"=>"Amarelo","2"=>"Azul","3"=>"Branco");
print_r(array_reverse($chave,true));
?>
```

A saída do código acima será:

```
Array (
[3] => Branco
[2] => Azul
[1] => Amarelo )
```

Observe no exemplo 2 que as chaves foram preservadas.

- **`count()`**

A função **`count()`** retorna a quantidade de elementos de um **`array`** ou propriedades de um objeto. Vide os respectivos parâmetros no Quadro 5.11 a seguir.

Sintaxe

`count(matriz,modo)`

Quadro 5.11: Parâmetros da função <code>count()</code>	
Parâmetro	Descrição
matriz	Obrigatório. Especifica a matriz ou objeto para contar.
modo	Opcional. Especifica o modo da função. Valores possíveis: • 0 - padrão. Não detecta matrizes (interna) • 1 - Detecta matrizes Nota: Este parâmetro foi adicionado no PHP 4.2.

Fonte: Niederauer (2007, p. 22)

A função `count()` pode retornar 0 se a variável não está definida, mas também pode retornar 0 se uma variável contém um `array` vazio. A função **`isset()`** pode ser usada para testar se uma variável é definida.



Exemplo

```
<?php
$chave=array("1"=>"Amarelo","2"=>"Azul","3"=>"Branco");
echo count($chave);
?>
```

A saída do código acima será:

4

- ***sort()***

A função ***sort()*** ordena um *array* pelos valores. Esta função define novas chaves para os elementos do *array* e as chaves existentes serão removidas.

Essa função retorna *TRUE* se tudo foi sorteado ou *FALSE* se houve falhas (Quadro 5.12).

Sintaxe

`sort(matriz,tipo)`

Quadro 5.12: Parâmetros da função *sort()*.

Parâmetro	Descrição
matriz	Obrigatório. Especifica a matriz a ser ordenada.
tipo	Opcional. Especifica como ordenar os valores da matriz. Os valores possíveis são: • SORT_REGULAR – Padrão. Trata os valores como são (não modifica o tipo) • SORT_NUMERIC – Trata valores numericamente • SORT_STRING – Trata os valores como <i>strings</i> • SORT_LOCALE_STRING – Trata os valores como <i>strings</i>, baseado em configurações locais.

Fonte: Niederauer (2007, p. 24)

Exemplo

```
$chave=array("1"=>"amarelo","2"=>"Azul","3"=>"Branco");
sort($chave);
print_r($chave);
?>
```

A saída do código acima será:

```
Array (
  [0] => Amarelo
  [1] => Azul
  [2] => Branco )
)
```

- ***array_sum()***

array_sum() retorna a soma de todos os valores na matriz. Vide o parâmetro no Quadro 5.13 a seguir.

Sintaxe

`array_sum(matriz)`

Quadro 5.13: Parâmetro da função *array_sum()*.

Parâmetro	Descrição
matriz	Obrigatório. Especifica uma matriz.

Fonte: Niederauer (2007, p. 21)

Exemplo

```
<?php
$valores=array(0=>"10",1=>"33",2=>"47");
echo array_sum($valores);
?>
```

A saída do código acima será:

```
90
```

Há, entretanto, a necessidade de visualizarmos o conteúdo das matrizes de forma dinâmica e seletiva. Para isto, tomaremos a estrutura de repetição ***foreach***, conforme a seguir.

5.3 O laço ***foreach***

O loop ***foreach*** é usado para percorrer matrizes. Esta estrutura permite varrer matrizes sem que se saiba antecipadamente o seu tamanho ou dimensão.

Sintaxe

```
foreach ($matriz as $valor) {
    código a ser executado;
}
```

Para cada iteração, o valor do elemento da matriz atual é atribuído a **\$valor** (e o ponteiro do *array* é movido de um elemento) – para a próxima iteração do laço, você estará olhando para o valor do elemento seguinte da matriz.

Exemplo

A seguir, é demonstrado um *loop* que irá imprimir os valores da matriz de dados:

```
<html>
<body>

<?php
$mat=array("Amarelo","Azul","Branco");
foreach ($mat as $valor) {
    echo $valor . "<br>";
}
?>

</body>
</html>
```

A saída deverá ser a seguinte:

```
Amarelo
Azul
Branco
```

Resumo

Conhecemos as diversas funções matemáticas responsáveis pela maioria das operações realizadas por *scripts* que executam cálculos matemáticos por diversos objetivos. Vimos utilizar as funções matemáticas, seus respectivos parâmetros e retornos.

Nesta aula tivemos a oportunidade de trabalhar e entender as funcionalidades de objetos do tipo *array*. Vimos que eles servem para o armazenamento, manutenção e disponibilização de dados consecutivos em duas dimensões (índice e dados). Definimos e estudamos sua estrutura de armazenamento e métodos de acesso e demais operações.

Além de conhecer a maioria de suas funções, estudamos várias delas com exemplos e visualizações dos resultados de suas respectivas aplicações práticas. Compreendemos tais aplicações nas três formas de apresentação desses objetos, que são: *arrays* numéricos, que armazenam dados em índices numéricos; *arrays* associativos, que armazenam dados com índices mistos e criados pelo programador ou pelo próprio sistema e, por fim, *arrays* multidimensionais, que são elaborados em qualquer das duas formas anteriores, para trabalhar em mais de duas dimensões. Vimos como utilizar uma forma de estrutura de repetição para termos acesso ao conteúdo das matrizes sem o necessário conhecimento prévio de sua dimensão; é o caso do laço ***foreach***, que executa iterações conforme o número de elementos dentro de um objeto *array*.

Atividades de aprendizagem

1. Digite todos os códigos exemplos desta aula e execute-os comparando os resultados obtidos com as respectivas saídas aqui ilustradas. Além disso, poste os códigos digitados no fórum criado para esta atividade, bem como um breve relatório descrevendo os resultados das suas execuções.
2. Escolha cinco funções de cada quadro listadas nesta aula (matemática e matriz) que não foram citadas nos exemplos e escreva *scripts* em PHP que demonstrem as suas respectivas funcionalidades, utilizando chamadas a funções com passagem de parâmetros.
3. Desenvolva um *script* em PHP que ordene uma matriz numérica de 10 elementos, em ordem crescente, cujo conteúdo pode ser constituído por quaisquer elementos.

Dica: pesquise sobre o método de ordenação denominado “método da bolha” e utilize-o para o desenvolvimento do *script*.

4. Crie um programa que imprima o maior e o menor número dentro de uma matriz numérica qualquer (suponha que não se saiba a dimensão da matriz).
5. Elabore um programa que imprima todos os possíveis números que estejam repetidos dentro de uma matriz numérica qualquer (suponha que não se saiba a dimensão da matriz).

Poste suas respostas no fórum para estas atividades no ambiente virtual de ensino-aprendizagem.

Aula 6 - PHP: funções *string* e funções de inserção de arquivos externos

Objetivos

Conhecer os operadores da linguagem e as funções *string*.

Saber como inserir arquivos externos ao *script*.

6.1 Variáveis do tipo *string*

Uma variável *string* é usada para armazenar e manipular texto. Também são usadas para guardar valores que contêm caracteres. Neste tópico, vamos olhar para as funções mais comuns e os operadores usados para manipular *strings* em PHP.

Uma *string* pode ser usada diretamente em uma função ou pode ser armazenada em uma variável.

Abaixo, o *script* PHP atribui o texto “Olá Mundo” para uma variável *string* chamada **\$txt**:

```
<?php
$txt=“Ei mamãe, olha eu na web!”;
echo $txt;
?>
```

O comando **echo** promove a saída ou impressão do conteúdo da variável **\$txt**, conforme abaixo:

```
Ei mamãe, olha eu na web!
```

Agora, vamos tentar usar algumas funções diferentes para manipular sequências de caracteres.

6.1.1 Função *strlen()*

A função ***strlen()*** é usada para retornar o comprimento de uma sequência de caracteres.

No exemplo abaixo podemos descobrir o comprimento de uma *string*.

```
<?php
echo strlen("Ei mamãe, olha eu na web!");
?>
```

A saída do código acima será:

```
25
```

O comprimento de uma sequência é frequentemente usado em *scripts* ou outras funções, quando é importante saber quando a sequência termina (ou seja, em um *loop* ou laço de repetição, nós iríamos querer parar o *loop* após o último caractere da sequência).

6.1.2 Função *strpos()*

A função ***strpos()*** é usada para pesquisar dentro de uma sequência de caracteres. Se uma correspondência for encontrada, esta função irá retornar a posição da sequência que se está procurando. Se nenhuma correspondência for encontrada, ela irá retornar *FALSE*.

Vamos ver se conseguimos encontrar a sequência "mamãe" em nosso texto:

```
<?php
echo strpos("Ei mamãe, olha eu na web!", "mamãe");
?>
```

A saída do código acima será:

```
3
```

A posição inicial da palavra "mamãe" encontrada em nossa sequência é 3. A razão de ser 3 (e não 4), é que a primeira posição na sequência inicia-se de 0, e não de 1.

6.1.3 Relação das funções *string*

As funções *string* permitem manipular cadeias de caracteres e para isto não há necessidade de qualquer instalação extra para utilizar essas funções; elas já fazem parte do compilador da linguagem. Abaixo se encontra um quadro (Quadro 6.1) com várias funções mais utilizadas do PHP com as respectivas versões de funcionalidade.

Quadro 6.1: Funções <i>string</i> mais utilizadas em PHP.		
Função	Descrição	Versão PHP
<i>bin2hex()</i>	Converte uma sequência de caracteres ASCII com valores hexadecimais.	3
<i>chr()</i>	Retorna um caractere a partir de um valor ASCII especificado.	3
<i>chunk_split()</i>	Divide uma <i>string</i> em várias partes menores.	3
<i>count_chars()</i>	Conta quantas vezes um caractere ASCII ocorre dentro de uma sequência e retorna a informação.	4
<i>explode()</i>	Quebra uma <i>string</i> em um <i>array</i> .	3
<i>fprintf()</i>	Grava uma <i>string</i> formatada para um fluxo de saída especificado.	5
<i>html_entity_decode()</i>	Converte entidades HTML em caracteres.	4
<i>htmlentities()</i>	Converte caracteres em entidades HTML.	3
<i>implode()</i>	Retorna uma <i>string</i> a partir dos elementos de um <i>array</i> .	3
<i>localeconv()</i>	Obtém a informação da formatação numérica.	4
<i>ltrim()</i>	Tira espaços em branco do lado esquerdo de uma <i>string</i> .	3
<i>nl2br()</i>	Insere quebra de linha HTML antes de cada nova linha em uma <i>string</i> .	3
<i>number_format()</i>	Formata um número com os milhares agrupados.	3
<i>ord()</i>	Retorna o valor ASCII do primeiro caractere de uma <i>string</i> .	3
<i>print()</i>	Saída de uma <i>string</i> .	3
<i>printf()</i>	Saída de uma <i>string</i> formatada.	3
<i>rtrim()</i>	Tira espaço em branco do lado direito de uma <i>string</i> .	3
<i>similar_text()</i>	Calcula a similaridade entre duas sequências.	3
<i>soundex()</i>	Calcula a chave <i>soundex</i> de uma <i>string</i> .	3
<i>sscanf()</i>	Interpreta a entrada de uma <i>string</i> de acordo com um formato.	4
<i>str_ireplace()</i>	Substitui alguns caracteres em uma <i>string</i> (maiúsculas e minúsculas).	5
<i>str_repeat()</i>	Repete uma sequência um determinado número de vezes.	4
<i>str_replace()</i>	Substitui alguns caracteres em uma <i>string</i> (maiúsculas e minúsculas).	3
<i>str_shuffle()</i>	Aleatoriamente embaralha todos os caracteres em uma <i>string</i> .	4
<i>str_split()</i>	Divide uma <i>string</i> em um <i>array</i> .	5
<i>str_word_count()</i>	Conta o número de palavras em uma <i>string</i> .	4
<i>strcasecmp()</i>	Compara duas sequências (maiúsculas e minúsculas).	3
<i>strpos()</i>	Localiza a primeira ocorrência de uma <i>string</i> dentro de outra <i>string</i> (apelido para <i>strstr()</i>).	3
<i>strcmp()</i>	Compara duas sequências de caracteres (<i>case-sensitive</i>).	3
<i>strip_tags()</i>	Tira as <i>tags</i> HTML e PHP de uma <i>string</i> .	3
Continua		

<i>strpos()</i>	Retorna a posição da primeira ocorrência de uma <i>string</i> dentro de outra sequência de caracteres (maiúsculas e minúsculas).	5
<i>strlen()</i>	Retorna o comprimento de uma <i>string</i> .	3
<i>strncmp()</i>	Comparação dos <i>n</i> primeiros caracteres da <i>string</i> (<i>case-sensitive</i>).	4
<i>strpos()</i>	Retorna a posição da primeira ocorrência de uma <i>string</i> dentro de outra sequência de caracteres (<i>case-sensitive</i>).	3
<i>strrchr()</i>	Localiza a última ocorrência de uma <i>string</i> dentro de outra <i>string</i> .	3
<i>strrpos()</i>	Encontra a posição da última ocorrência de uma <i>string</i> dentro de outra sequência de caracteres (maiúsculas e minúsculas).	5
<i>strrpos()</i>	Encontra a posição da última ocorrência de uma <i>string</i> dentro de outra sequência de caracteres (<i>case-sensitive</i>).	3
<i>strstr()</i>	Localiza a primeira ocorrência de uma <i>string</i> dentro de outra sequência de caracteres (<i>case-sensitive</i>).	3
<i>strtok()</i>	Divide uma <i>string</i> em <i>strings</i> menores.	3
<i>strtolower()</i>	Converte uma <i>string</i> para letras minúsculas.	3
<i>strtoupper()</i>	Converte uma <i>string</i> para letras maiúsculas.	3
<i>substr()</i>	Retorna uma parte de uma <i>string</i> .	3
<i>substr_count()</i>	Conta o número de vezes que ocorre uma <i>substring</i> em uma <i>string</i> .	4
<i>substr_replace()</i>	Substitui parte de uma <i>string</i> com outra <i>string</i>	4
<i>trim()</i>	Retira espaços em branco de ambos os lados de uma sequência de caracteres.	3
<i>fprintf()</i>	Grava uma <i>string</i> formatada para um fluxo de saída especificado.	5
Conclusão		

Fonte: Gilmore (2008, p.185-220); Niederauer (2007, p.117-123); http://php.net/manual/pt_BR/ref.strings.php

Para exemplificar, a seguir mostramos a utilização de algumas funções *string*:

- a) ***bin2hex()*** – esta função converte uma sequência de caracteres ASCII em valores hexadecimais. O Quadro 6.2 informa o parâmetro dessa função. A sequência pode ser convertida de volta usando a função ***pack()***.

Sintaxe

`bin2hex(string)`

Quadro 6.2: Parâmetro da função *bin2hex()*.

Parâmetro	Descrição
<i>string</i>	Parâmetro obrigatório. É a <i>string</i> a ser convertida.

Fonte: Niederauer (2007, p. 117)

No exemplo, a seguir, vamos converter um valor de sequência de binário para hexadecimal e vice-versa:

```
<?php
$var = "Ei mamãe!";
echo bin2hex($var) . "<br>";
echo pack("H*",bin2hex($var)) . "<br>";
?>
```

A saída do código acima será:

```
4569206d616de36521
Ei mamãe!
```

b) *chr()* – retorna um caractere do valor ASCII especificado e o seu parâmetro é informado no Quadro 6.3 a seguir.

Sintaxe

```
chr(valor)
```

Quadro 6.3: Parâmetro da função *chr()*.

Parâmetro	Descrição
valor	Parâmetro obrigatório. É um valor ASCII.

Fonte: Niederauer (2007, p. 117)

O parâmetro **valor** pode ser especificado em decimal, octal ou hexadecimal. Valores octais são definidas por um 0 à esquerda, enquanto os valores hexadecimais são definidas por um **0x**.



Exemplo

```
<?php
echo chr(65)."<br>";
echo chr(065)."<br>";
echo chr(0x65)."<br>";
?>
```

A saída do código acima será:

```
A
5
e
```

- c) **`count_chars()`** – retorna quantas vezes um caractere ASCII ocorre dentro de uma sequência e retorna a informação. Para executar esta função faz-se necessário o parâmetro *Mode*, informado no Quadro 6.4 com suas respectivas opções.

Sintaxe

```
count_chars(string,mode)
```

Quadro 6.4: Parâmetros da função `count_chars()`.

Parâmetro	Descrição
<i>string</i>	Parâmetro obrigatório. É a sequência a ser verificada.
<i>Mode</i>	Parâmetro opcional. Ele especifica os modos que serão retornados. 0 é o padrão. Os modos diferentes de retorno são: 0 - um <i>array</i> com o valor ASCII como chave e número de ocorrências como o valor. 1 - um <i>array</i> com o valor ASCII como chave e número de ocorrências como valores, apenas lista ocorrências maior do que zero. 2 - um <i>array</i> com o valor ASCII como chave e número de ocorrências como valores, apenas lista as ocorrências iguais a zero. 3 - uma <i>string</i> com todos os personagens diferentes usados. 4 - uma <i>string</i> com todos os caracteres não utilizados.

Fonte: Niederauer (2007, p. 117)

Exemplo 1

Neste exemplo, usaremos a função **count_chars()** com um dos *modos* que verifica a cadeia. O modo 1 irá retornar um *array* com o valor ASCII como chave e quantas ocorrências houver (como no exemplo abaixo, o valor ASCII para as letras “a” e “e” ocorre duas vezes, cada uma). Nesse exemplo, a letra “e” é representada pelo código 101.

```
<?php
$var = "ei mamãe!";
print_r(count_chars($var,1));
?>
```

A saída do código acima será:

```
Array
( [32] => 1
  [33] => 1
  [97] => 1
  [101] => 2
  [105] => 1
  [109] => 2
  [227] => 1)
```

Exemplo 2

Neste próximo exemplo, usaremos o **count_chars()** com o modo 3 para verificar a cadeia. Este modo irá retornar uma *string* com todos os caracteres utilizados, sem repetição:

```
<?php
$var = "ei mamãe!";
echo count_chars($var,3);
?>
```

A saída do código acima será:

```
!aeimã
```

- d) **explode()** – esta função quebra uma *string* em um *array*. Para a sua execução é necessário informar obrigatoriamente dois parâmetro e um parâmetro opcional. Vide a relação dos parâmetros no Quadro 6.5 a seguir.

Sintaxe

```
explode(separator,string,limit)
```

Quadro 6.5: Parâmetros da função *count_explode()*.

Parâmetro	Descrição
separador	Obrigatório. Especifica o caractere onde ocorre a quebra.
string	Obrigatório. Contém a sequência a ser quebrada.
limite	Opcional. Especifica o número máximo de retorno de elementos da matriz.

Fonte: Niederauer (2007, p. 118)



O parâmetro separador não pode ser uma sequência vazia.

Exemplo

Neste exemplo, vamos quebrar uma *string* em um *array*, tendo como parâmetro das quebras o caractere de espaço (" "):

```
<?php
$var = "Ei mamãe, olha eu na web!";
print_r (explode(" ", $var));
?>
```

A saída do código acima será:

```
Array
( [0] => Ei
  [1] => mamãe.
  [2] => Olha
  [3] => eu
  [4] => na
  [5] => web! )
```

- e) **str_repeat()** – esta função repete uma sequência de um determinado número de vezes. Ela necessita de dois parâmetros, conforme Quadro 6.6 a seguir.

Sintaxe

```
str_repeat(string,vezes)
```

Quadro 6.6: Parâmetros da função *str_repeat()*.

Parâmetro	Descrição
<i>string</i>	Obrigatório. Especifica a sequência a ser repetida.
<i>vezes</i>	Obrigatório. Especifica o número de vezes que a sequência será repetida. Deve ser maior ou igual a 0.

Fonte: Niederauer (2007, p. 120)

Exemplo

```
<?php
echo str_repeat("*",5);
?>
```

A saída do código acima será:

```
* * * * *
```

- f) **strlen()** – esta função retorna o comprimento de uma *string* e já foi explicada no subtópico 6.1.1.
- g) **strtoupper()** – esta função converte uma *string* em maiúsculo, bastando informar como parâmetro a própria *string* que se deseja converter. Vide Quadro 6.7 a seguir.

Sintaxe

```
strtoupper(string)
```

Quadro 6.7: Parâmetro da função *strtoupper()*.

Parâmetro	Descrição
<i>string</i>	Obrigatório. Especifica a sequência de caracteres para converter.

Fonte: Niederauer (2007, p.122)

Exemplo

```
<?php
echo strtoupper("Ei Mamãe!");
?>
```

A saída do código acima será:

EI MAMÃE!

h) *substr()* – é uma das funções mais utilizadas para manipulação de *string* no PHP. Ela retorna uma parte de uma cadeia (ao seu *string*), a partir da entrada de dois parâmetros, conforme Quadro 6.8. a seguir.

Sintaxe

```
substr(string,início, tamanho)
```

Quadro 6.8: Parâmetros da função *strtoupper()*.

Parâmetro	Descrição
<i>string</i>	Obrigatório. A string de entrada.
início	Obrigatório. Especifica onde começar na cadeia: Um número positivo que indica uma posição específica, dentro da cadeia; Um número negativo indica que deve iniciar em uma posição especificada a partir do final da <i>string</i> ; 0 - Inicia no primeiro caractere da <i>string</i> ;
comprimento	Opcional. Especifica o comprimento da <i>string</i> retornada. O padrão é para o final da <i>string</i> : Um número positivo - O comprimento a ser retornado a partir do parâmetro indicado em início ; Número Negativo - O comprimento a ser retornado a partir do final da <i>string</i> .

Fonte: Niederauer (2007, p.122)



Se o parâmetro **início** for um número negativo e o parâmetro **comprimento** for menor ou igual ao parâmetro **início**, o parâmetro **comprimento** inicia em 0.

Exemplo 1

```
<?php  
echo substr("Ei mamãe!",3);  
?>
```

A saída do código acima será:

```
mamãe!
```

Exemplo 2

```
<?php  
echo substr("Ei mamãe!",3,3);  
?>
```

A saída do código acima será:

```
mam
```

6.2 Incluindo arquivos – *Server Side Includes (SSI)*

O PHP permite inserir o conteúdo de um arquivo PHP em um outro arquivo PHP; para isto, basta usarmos a função ***include()*** ou a função ***require()***.

As duas funções são idênticas em todos os sentidos, exceto pela manipulação de erros:

- ***include()*** gera um aviso, mas o *script* continuará a execução.
- ***require()*** gera um erro fatal, e o *script* pára de executar.

Essas duas funções são usadas para criar funções, cabeçalhos, rodapés ou elementos que serão reutilizados em várias páginas.

Esses comandos são muito úteis no sentido de poupar reprogramação de *scripts*. Por exemplo, podemos criar um *script* de menu que poderá ser utilizado por diversos outros *scripts*, bastando, para isto, que estes *scripts* façam a chamada (utilizando ***include()*** ou ***require()***) a estas funções, em vez de atualizar os *links* em todas as suas páginas *web*.

6.2.1 A função ***include()***

A função ***include()*** insere todo o conteúdo de um arquivo especificado no arquivo atual.

Se ocorrer um erro, a função ***include()*** gera um aviso, mas o *script* continuará a execução.

Exemplo 1

Suponha que você tenha um arquivo de cabeçalho padrão, chamado “cabec.php”. Para incluir o arquivo de cabeçalho em uma página, use a função ***include()***:

```
<html>
<body>

<?php
include("cabec.php");
?>
<p>Esta página possui um cabeçalho padrão,
incorporado pelo arquivo cabec.php </p>
</body>
</html>
```

Exemplo 2

Suponha que temos um arquivo de menu padrão, chamado “menu.php” e que deve ser usado em todas as páginas. Este arquivo pode ser deste tipo:

```
<a href="/index.php">Home</a>
<a href="/cabec_cefetmg.php">CEFET-MG</a>
<a href="/cabec_ufmg.php">UFMG</a>
<a href="/cabec_etec.php">e-TEC</a>
<a href="/contato.php">Contato</a>
```

Todas as páginas do *site* deve incluir este arquivo menu. Veja como isso pode ser feito:

```
<html>
<body>

<div class="leftmenu">
  <?php
  include("menu.php");
  ?>
</div>

<p>Esta página possui um menu padrão,
incorporado pelo arquivo menu.php </p>

</body>
</html>
```

Se verificarmos o código-fonte da página acima em um navegador, ele será parecido com o abaixo ilustrado:

```
<html>
<body>

<div class="menu_menu">
  <a href="/index.php">Home</a>
  <a href="/cabec_cefetmg.php">CEFET-MG</a>
  <a href="/cabec_ufmg.php">UFMG</a>
  <a href="/cabec_etec.php">e-TEC</a>
  <a href="/contato.php">Contato</a>
</div>

<p>Esta página possui um menu padrão,
incorporado pelo arquivo menu.php </p>

</body>
</html>
```

6.2.2 A função **require()**

A função **require()** é idêntica a **include()**, exceto pelo fato de ela lidar com os erros de forma diferente.

Se ocorrer um erro durante a execução do *script*, a função **include()** gerará um aviso, mas o *script* continuará a execução. Entretanto, a função **require()** gerará um erro.

Em suma, recomenda-se usar a função **require()** em vez de **include()**, pois os *scripts* não devem continuar após um erro.

Resumo

Nesta aula contextualizamos todas as funcionalidades de objetos do tipo *string*. Definimos e estudamos os seus operadores. Além de nos instruir sobre a maioria de suas funções, estudamos várias dessas funções com exemplos e visualizações dos resultados de suas respectivas aplicações práticas, tais como: função **strlen()** e **strpos()**.

Finalmente, estudamos como incluir arquivos ou pedaços de códigos-fonte PHP externos a um programa, utilizando as instruções de inserção de arquivos: **include()** e **require()** e identificamos qual a diferença entre uma e outra.

Atividades de aprendizagem

1. Digite todos os códigos exemplos desta aula e execute-os, comparando os resultados obtidos com as respectivas saídas aqui ilustradas. Além disso, poste os códigos digitados no fórum criado para esta atividade, bem como um breve relatório descrevendo os resultados das suas execuções.
2. Escolha cinco funções do quadro de funções de *string* que não foram citadas nos exemplos e escreva *scripts* em PHP que demonstrem as suas respectivas funcionalidades.
3. Desenvolva um *script* (principal) que imprima o volume de um cubo e a hipotenusa de um triângulo retângulo a partir da chamada a duas funções, cujos códigos-fonte estejam em um arquivo externo. O aluno deverá desenvolver tanto o *script* que irá imprimir o que se pede quanto o arquivo que contenha as funções que calculam o volume do cubo e a hipotenusa do triângulo.

Dados:

- a) a passagem dos parâmetros para as funções deve ser feita no *script* principal;
- b) o arquivo externo (que contém o código das funções) deve ser incluído no *script* principal através de **include()** ou **require()**;
- c) a função que calcula o volume de um cubo deve receber um parâmetro de entrada e deve retornar o resultado correspondente;
- d) a função que calcula a hipotenusa de um triângulo retângulo deve receber dois parâmetros de entrada (cateto1 e cateto2) e deve retornar o resultado correspondente;
- e) Formulário:
 - i. Volume do cubo $\rightarrow V = L^3$, onde V é o volume e L um dos lados;
 - ii. Hipotenusa $\rightarrow H = \sqrt{C_a^2 + C_o^2}$ onde H é a hipotenusa, C_a e C_o os catetos adjacente e oposto, respectivamente, do triângulo retângulo.

Poste suas respostas no fórum para estas atividades no ambiente virtual de ensino-aprendizagem.

Aula 7 – PHP: manipulação de formulários

Objetivo

Utilizar formulários e os tipos de transferências de dados entre páginas utilizando os operadores **\$_GET** e **\$_POST**.

7.1 Manipulação de formulários

O PHP utiliza as variáveis de ambiente: **\$_GET** e **\$_POST** para recuperar informações de formulários produzidas pelos usuários.

O mais importante a se notar, quando se lida com formulários HTML e PHP, é que qualquer elemento de formulário de uma página HTML, automaticamente estará disponível para seus **scripts** PHP (TANSLEY, 2002, p. 123-134; GILMORE, 2008, p. 281-294).

Exemplo

O exemplo a seguir contém um formulário HTML com dois campos de entrada e um botão de envio:

```
<html>
<body>

<form action="pag_entrada.php" method="post">
  Nome: <input type="text" name="ID" >
  Mês_Nascimento: <input type="text" name="mes" >
  <input type="submit" >
</form>

</body>
</html>
```

Uma vez preenchido o formulário acima e se clicando no botão enviar, os dados desse formulário serão enviados para um arquivo PHP, chamado "pag_entrada.php", tal como identificado como atributo do **form** acima.

Agora vamos mostrar como o programa que deverá receber esses dados pode se parecer:

```
<html>
<body>

Olá
<?php
    echo $_GET["ID"];
?>
!<br>
Você nasceu em
<?php
    echo $_GET["mes"];
?>
</body>
</html>
```

Supondo que nos campos ID e mês do formulário sejam preenchidos com "William Sallum" e "Maio", respectivamente, a saída poderia ser algo assim:

```
Olá William Sallum!
Você nasceu em Maio.
```

A forma de se passar dados de formulários HTML para programas do PHP é mostrada acima. No PHP as variáveis especiais **\$_GET** e **\$_POST** serão explicadas nas próximas seções.

7.1.1 Validação de formulário

As entradas de formulários do usuário devem ser validadas no *browser*, sempre que possível (através de *scripts* tais como os utilizados em JavaScript). A validação dos dados no navegador é mais rápida e reduz a carga do servidor.

Você deve considerar a validação no servidor se a entrada do usuário será inserida em um banco de dados; mesmo assim, a validação deve ser considerada no próprio cliente (*browser* local). Uma boa maneira de validar um formulário no servidor é enviar o formulário para si, em vez de saltar para uma página diferente. O usuário vai então receber as mensagens de erro na mesma página do formulário. Isto torna mais fácil descobrir o erro.

Enviar os dados para si significa ou colocar o nome do seu programa no atributo de formulário, *action* (por exemplo: se seu arquivo de programa que contém o formulário a ser enviado se chama *meuProg.php*) tal como *action="meuProg.php"* ou atribuir o símbolo "#" no lugar do próprio nome do arquivo, tal como: *action="#"*.

7.1.2 A função **\$_GET**

A variável "superglobal" **\$_GET** é usada para coletar os valores de um formulário com o *method = "get"*.

A informação enviada num formulário com o método GET é visível para todos (pois será exibida na barra de endereços do navegador) e tem limites para a quantidade de dados a serem enviados. Veja o exemplo a seguir.

Exemplo

```
<form action="pag_entrada.php" method="get">
  Name: <input type="text" name="ID" />
  Mes: <input type="text" name="mes" />
  <input type="submit" value="Enviar" />
</form>
```

Quando o usuário clica no botão "Enviar", a URL enviada para o servidor poderia ser algo parecido com isto:

```
http://www.cefetmg.br/pag_entrada.php?ID=William&mes=Maio
```

O arquivo "pag_entrada.php" agora pode usar a variável superglobal **\$_GET** para coletar os dados do formulário (os nomes dos campos do formulário serão automaticamente as chaves do array **\$_GET**):

```
Olá
<?php
echo $_GET["ID"];
?>
!<br>
Você nasceu em
<?php
echo $_GET["mes"];
?>
```

7.1.3 Usando *method = "GET"*

Quando usamos o método = "*GET*" em formulários HTML, todos os nomes de variáveis e valores são exibidos no URL.

Este método não deve ser usado no caso de envio de senhas ou de outras informações sigilosas! No entanto, isso pode ser útil em alguns casos, tal como didaticamente no ensino da linguagem.



O método **GET** não é adequado para valores de variáveis muito grande. Não devem ser usados com valores superiores a 2.000 caracteres.

7.1.4 A função **`$_POST`**

A função interna **`$_POST`** é usada para coletar os valores de um formulário com o "*method = POST*".

A informação enviada num formulário com o método *POST* é invisível para os outros e não tem limites para a quantidade de dados a enviar.



Por uma questão de padronização, existe um tamanho máximo de 8 Mb para o método **POST**, o qual pode ser alterado pela configuração de **`post_max_size`** no arquivo php.ini.

Exemplo

```
<form action="pag_entrada.php" method="post">
  Name: <input type="text" name="ID" />
  Mes: <input type="text" name="mes" />
  <input type="submit" value="Enviar" />
</form>
```

Quando o usuário clica no botão "Enviar", o URL será parecido com este:

```
http://www.cefetmg.br/pag_entrada.php
```

Note que neste URL, diferentemente do exemplo apresentado em **`$_GET`**, as variáveis e seus respectivos conteúdos não aparecem na linha de endereço.

O arquivo “pag_entrada.php” agora pode usar a função **\$_POST** para coletar dados do formulário (os nomes dos campos do formulário serão automaticamente as chaves do array **\$_POST**):

```
Welcome <?php echo $_POST["fname"]; ?>!  
You are <?php echo $_POST["age"]; ?> years old.
```

7.1.5 Usando o parâmetro *method* = “POST”

A informação enviada num formulário com o método **POST** é invisível para os outros e não tem limites para a quantidade de dados a enviar.

No entanto, porque as variáveis não são exibidas na URL, não é possível registrar a página.

7.2 Upload de arquivos

Com o PHP, é possível fazer *upload* de arquivos para o servidor. O PHP permite que os usuários façam *upload* de arquivos de um formulário, o que é muito útil em várias circunstâncias (GILMORE, 2008, p. 513-523).

Veja o seguinte formulário HTML para *upload* de arquivos:

```
<html>  
<body>  
  
<form action="arq_upload.php" method="post" enctype="multipart/  
form-data">  
  Arquivo para Up-load:  
  <input type="file" name="arq" id="arq" >  
  <br>  
  <input type="submit" name=" Enviar" value="Enviar" />  
</form>  
  
</body>  
</html>
```

Observe o seguinte sobre o formulário HTML acima:

- O atributo **enctype** do **<form>** especifica qual tipo de conteúdo será usado ao enviar o formulário.
- O atributo *"Multipart / form-data"* é utilizado quando um formulário enviar dados binários, tais como imagens, vídeos, sons, etc.
- O atributo **type = "file"** da tag **<input>** especifica que a entrada deve ser processada como um arquivo. Por exemplo, quando visualizado em um navegador, haverá um botão ao lado do campo de entrada que, ao ser pressionado, disponibilizará a janela do Windows Explorer com finalidade de que o usuário procure e selecione o arquivo a ser enviado (*upload*).

Permitir que os usuários façam *upload* de arquivos é um grande risco à segurança. É importante analisar as condições de permissões para permitir que apenas os usuários de confiança possam realizar o *upload* de arquivos.

A seguir (Figura 7.1) se encontra a página que deverá ser apresentada conforme o HTML descrito anteriormente.

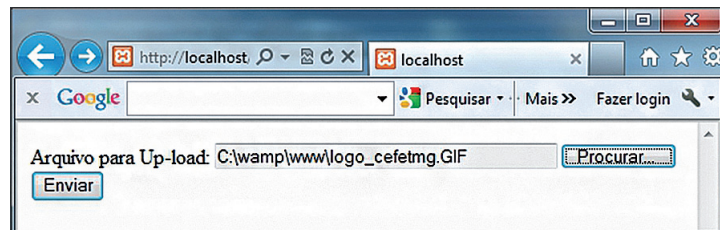


Figura 7.1: Janela resultante de formulário HTML com utilização de campo FILE

Fonte: Elaborada pelo autor e visualizada pelo Internet Explorer versão 9.0

Nesse exemplo, especificamente, podemos observar que o arquivo selecionado é o "logo_cefet.gif", que se encontra no endereço: C:\Users\Sallum\Documents\, da máquina do usuário.

7.2.1 Criando *script* que recebe *upload* de arquivo de envio

O arquivo “*arq_upload.php*” contém o código que faz *upload* de um arquivo, conforme a seguir:

```
<?php

if ($_FILES["arq"]["error"] > 0) {
    echo "Erro: " . $_FILES["arq"]["error"] . "<br>";
}
else {
    echo "Arquivo para Upload: " . $_FILES["arq"]["name"] . "<br>";
    echo "Tipo: " . $_FILES["arq"]["type"] . "<br>";
    echo "Tamanho: " . round(($_FILES["arq"]["size"] / 1024), 2) . "
Kb<br>";
    echo "Local de armazenamento: " . $_FILES["arq"]["tmp_name"];
}
?>
```

Usando a matriz global representada pela variável global **`$_FILES`**, podemos carregar arquivos de um computador cliente para o servidor remoto.

Nesse *script*, o primeiro parâmetro é o nome do formulário de entrada; o segundo, o “tipo” do arquivo; o terceiro, o “tamanho” do arquivo (em *bytes*) e, finalmente, o quarto, o local de destino “*tmp_name*”, onde este arquivo deverá ser armazenado. Assim, a relação destas palavras-chave com suas respectivas sintaxes é:

- **`$_FILES`** [“Arquivo”] [“name”] – o nome do arquivo enviado
- **`$_FILES`** [“Arquivo”] [“type”] – o tipo do arquivo enviado
- **`$_FILES`** [“Arquivo”] [“size”] – o tamanho em *bytes* do arquivo enviado
- **`$_FILES`** [“Arquivo”] [“tmp_name”] – o nome da cópia temporária do arquivo armazenado no servidor
- **`$_FILES`** [“Arquivo”] [“error”] – o código de erro resultante do *upload* de arquivos



Como visto anteriormente, a função ***round()***, aqui apresentada, arredonda o número em duas casas decimais.

Após clicar no botão “Enviar”, será apresentada uma janela como a da Figura 7.2, com a saída produzida pelo *script* PHP mostrado anteriormente.

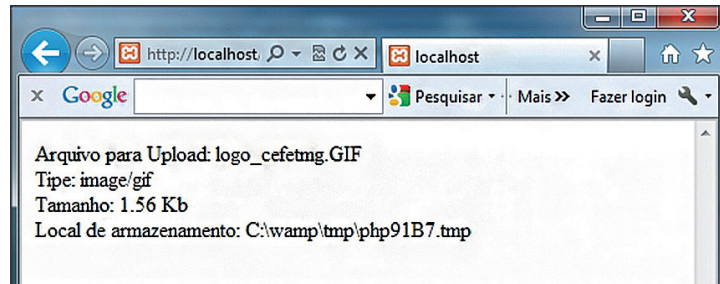


Figura 7.2: Janela refletindo o resultado do *upload*

Fonte: Elaborada pelo autor e visualizada pelo Internet Explorer versão 9.0

Esta é uma maneira muito simples de se fazer *upload* de arquivos. Por razões de segurança, você deve adicionar restrições sobre o que o usuário tem permissão para enviar.

7.2.2 Restrições em *upload*

Às vezes, por medida de segurança e administração de espaço físico nos servidores, faz-se necessário determinar regras e/ou restrições para os *uploads*.

Neste roteiro, acrescentamos algumas restrições para a execução de *upload* do arquivo. Aqui, o usuário só pode fazer *upload* de arquivos GIF ou JPEG e o tamanho do arquivo deve ser inferior a 20 kb. Observe o *script* abaixo:

```

<?php
if ((($_FILES["arq"]["type"] == "image/gif") ||

    ($_FILES["arq"]["type"] == "image/jpeg") ||

    ($_FILES["arq"]["type"] == "image/pjpeg")) &&

    ($_FILES["arq"]["size"] < 20000)) {
    if ($_FILES["arq"]["error"] > 0) {

        echo "Erro: " . $_FILES["arq"]["error"] . "<br>";
    }
    else {

        echo "ARQUIVO PARA UPLOAD: " . $_FILES["arq"]["name"] .
"<br>";

        echo "TIPO: " . $_FILES["arq"]["type"] . "<br />";

        echo "TAMANHO: " . round(($_FILES["arq"]["size"] / 1024), 2) . "
Kb<br>";

        echo "LOCAL DE ARMAZENAMENTO: " . $_FILES["arq"]["tmp_
name"];
    }
}
else {
    echo "Arquivo inválido!";
}
?>

```

Para o Internet Explorer reconhecer jpg, o tipo deve ser pjpeg; para o Firefox, deve ser jpeg.



7.2.3 PHP: gravando o arquivo transferido

Os exemplos acima criar uma cópia temporária dos arquivos enviados na pasta temporária do Apache no servidor, que neste nosso exemplo é: C:\xampp\tmp\

Os arquivos temporários copiados desaparecem quando o *script* termina. Para armazenar o arquivo enviado é preciso copiá-lo para um local diferente. Nesse sentido, desenvolvemos os *scripts* a seguir:

```
<?php
if ((($_FILES["arq"]["type"] == "image/gif") ||
($_FILES["arq"]["type"] == "image/jpeg") ||
($_FILES["arq"]["type"] == "image/pjpeg")) &&
($_FILES["arq"]["size"] < 20000)) {
if ($_FILES["arq"]["error"] > 0) {
echo "Return Code: " . $_FILES["arq"]["error"] . "<br />";
}
else {
echo "ARQUIVO PARA UPLOAD: " . $_FILES["arq"]["name"] . "<br>";
echo "TIPO: " . $_FILES["arq"]["type"] . "<br />";
echo "TAMANHO: " . round(($_FILES["arq"]["size"] / 1024), 2) . "
Kb<br>";
echo "ARQUIVO TEMPORÁRIO: " . $_FILES["arq"]["tmp_name"];
if (file_exists($_FILES["arq"]["name"])) {
echo "<br>O ARQUIVO: " . $_FILES["arq"]["name"] . " JÁ EXISTE. ";
}
else {
move_uploaded_file($_FILES["arq"]["tmp_name"], $_FILES["arq"]
["name"]);
echo "<br>ARMAZENADO COMO: " . $_FILES["arq"]["name"];
}
}
}
else {
echo "ARQUIVO INVÁLIDO!";
}
?>
```

O *script* acima verifica se o arquivo já existe, se não, ele copia o arquivo para a pasta que representa “www”, local.

A página gerada pelo *script* acima é assim ilustrada na Figura 7.3 a seguir.

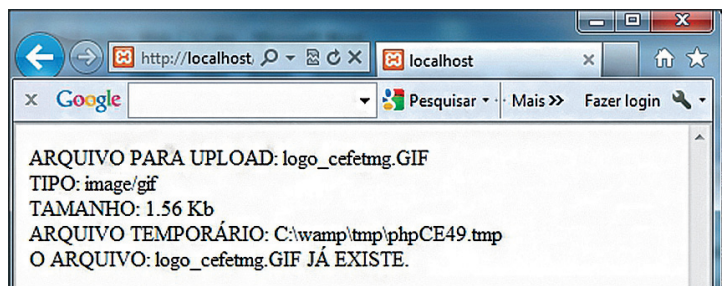


Figura 7.3: Janela refletindo o resultado do *upload* conforme código-fonte anterior

Fonte: Elaborada pelo autor e visualizada pelo Internet Explorer versão 9.0.

Resumo

Vimos nesta aula o poder que os formulários HTML assumem, transformando-se de simples formulários estáticos a formulários dinâmicos. Estudamos aqui a criação e manipulação de formulários, tal como vistas nas aulas de HTML. Para tal, aprendemos como transportá-los entre páginas. Como vimos, e dentro desse contexto, essa transação de dados somente tornou-se possível com a utilização das funções **`$_GET`** e **`$_POST`**.

Estudamos a técnica de se fazer *upload* de arquivos utilizando formulários e a função **`$_FILE`**. Esta função permite disponibilizar em matriz os dados básicos de arquivos selecionados para sua transferência. Assim, estudamos que, utilizando a função **`move_uploaded_file()`**, podemos transferir um arquivo de um lugar para outro.

Atividades de aprendizagem

1. Digite todos os códigos exemplos desta aula e execute-os, comparando os resultados obtidos com as respectivas saídas aqui ilustradas.
2. Ao clicar no botão “Enviar” do formulário HTML, ilustrado na Figura 7.4, são transferidos todos os dados desse formulário para o arquivo **`mostra_dados.php`**. Escreva o programa `mostra_dados.php` que deverá receber esses dados e, em seguida, mostre-os utilizando a tag de cabeçalho **`<h3>`**.

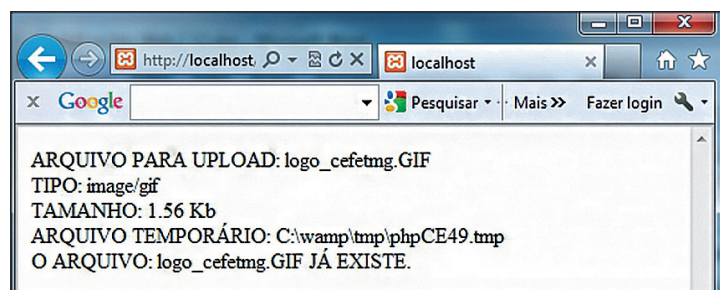


Figura 7.4: Modelo de formulário

Fonte: Elaborada pelo autor e visualizada pelo Internet Explorer versão 9.0

Eis, a seguir (Figura 7.5), o código-fonte (apenas da parte do formulário) ilustrado na Figura 7.4:

```
<HTML>
<BODY>
<h1> Formulário de cadastro de alunos </h1>

<form action="mostra_dados.php" method="POST">
  Nome: <input type="text" name="nome" id="nome" size=55><br>
  Endereço: <input type="text" name="endereco" id="endereco" size=55><br>
  Cidade: <input type="text" name="cidade" id="cidade" size=5><br>
  UF:
    <select name="uf">
      <option value="MG">MG</option>
      <option value="SP">SP</option>
      <option value="RJ">RJ</option>
      <option value="ES">ES</option>
    </select><br><br>
    <center>
      <input type="submit" name="submit" value="Enviar">
    </center>
  </form>
</BODY>
</HTML>
```

Figura 7.5: Código-fonte PHP que elabora o formulário visualizado na figura anterior

Fonte: Elaborada pelo autor.

Ao final desta atividade, você deverá postar os códigos aqui desenvolvidos no fórum criado para este fim.

Aula 8 – PHP: manipulando o banco de dados MySQL

Objetivos

Compreender a conexão com banco de dados.

Entender a manipulação e armazenamento de dados.

8.1 Conexão de banco de dados com PHP

O MySQL é um sistema de banco de dados gratuito, o qual é muito utilizado com a linguagem PHP (RAMOS, 2007, p. 91-110).

MySQL é o sistema mais popular de banco de dados *open-source* já existente. Os dados do MySQL são armazenadas em objetos de banco de dados chamados de tabelas bidimensionais. Uma tabela bidimensional é uma coleção de entradas de dados relacionados e que consiste em colunas e linhas.

Bancos de dados são úteis quando o armazenamento de informações se dá de forma categórica, ou seja, por categoria. Por exemplo: uma empresa pode ter um banco de dados com as seguintes tabelas: “Funcionários”, “Produtos”, “Clientes” e “Pedidos”.

8.1.1 Uma tabela de banco de dados

Um banco de dados, na maioria das vezes, contém uma ou mais tabelas. Cada tabela é identificada por um nome (por exemplo, “turmas” ou “alunos”). As tabelas contêm registros (linhas) com dados pertinentes aos contextos, ou seja: as respectivas finalidades delas.

A seguir se encontra um exemplo de uma tabela (Tabela 8.1) chamada “Alunos”:

Tabela 8.1: Registros de alunos.			
Entrada	Nome	Endereço	Cidade
2010	José A.W. C.	R. das flores, 34	Belo Horizonte
2011	Maria K. J.	Av. imaginária, 123	Formiga
2011	Antônio L.	R. da invenção, 10	Cajuru

Fonte: Elaborada pelo autor

A tabela anterior contém três registros dispostos em três linhas (um para cada aluno) e quatro colunas (Entrada, Nome, Endereço e Cidade).

Consultas

Uma consulta é uma pergunta (ou um pedido de informação), formulada por meio de linguagem própria do banco de dados; no caso do MySQL é a própria SQL – *Structured Query Language*, ou Linguagem de Consulta Estruturada.



Outras informações a respeito
dessa magnífica linguagem
podem ser obtidas em:
[http://www.dca.fee.unicamp.br/
cursos/PooJava/javadb/sql.html](http://www.dca.fee.unicamp.br/cursos/PooJava/javadb/sql.html)

A SQL é uma linguagem de pesquisa declarativa para banco de dados relacional (base de dados relacional). Muitas das características originais do SQL foram inspiradas na álgebra relacional.

Com o MySQL, podemos consultar um banco de dados para obter informações específicas e ter um conjunto de registros retornado, mediante uma consulta prévia.

Como exemplo de utilização da linguagem SQL no banco de dados MySQL, vejamos a seguinte consulta:

```
SELECT Nome FROM alunos
```

A consulta acima selecionará todos os dados de “Nome” da coluna da tabela de alunos, e irá retornar um conjunto de registros como este:

Tabela 8.2: Listagem de nomes

Nome
José A.W. C.
Maria K. J.
Antônio L.

Fonte: Elaborada pelo autor

8.1.2 Criando uma conexão com um Banco de Dados MySQL

Antes de acessar dados em um banco de dados, é necessário que estabeleçamos uma conexão com este banco.

No PHP, isso é feito com a função ***mysql_connect*** (*parâmetros*). Veja no Quadro 8.1 a relação dos parâmetros utilizados.

Sintaxe

```
mysql_connect(servidor, usuário, senha);
```

Quadro 8.1: Parâmetros da função `mysql_connect()`.

Parâmetro	Descrição
servidor	Opcional. Especifica o servidor a se conectar. O valor padrão é <code>"localhost: 3306"</code> ou seja: computador (<code>localhost</code>) e porta de conexão (3306 – que é padrão do MySQL)
usuário	Opcional. Especifica a identificação do usuário para <i>login</i> . O valor padrão é o nome do usuário cadastrado previamente no servidor.
senha	Opcional. Especifica a senha para <i>login</i> . O padrão é <code>" "</code> (vazio)

Fonte: http://www.php.net/manual/pt_BR/ref.mysql.php

Existem outros parâmetros, mas os listados acima são os mais importantes.



Exemplo

No exemplo a seguir nós armazenamos a conexão em uma variável (**\$conexao**) para uso posterior no *script*. A função interna `"die"` será executada se a conexão falhar e a função `mysql_error()` deverá informar o tipo de erro ocorrido:

```
<?php
$conexao = mysql_connect("localhost","sallum","");
if (!$conexao) {
    die("Falha na conexão: " . mysql_error());
}
?>
```



Visite o endereço a seguir para mais referências a respeito:
http://br.php.net/manual/pt_BR/function.mysql-connect.php

8.1.3 Finalizando uma conexão

A conexão será fechada automaticamente quando o *script* terminar. Para fechar a conexão antes, usaremos a função **`mysql_close()`**:

```
<?php
$conexao = mysql_connect("localhost","sallum","");
if (!$conexao) {
    die("Falha na conexão!" . mysql_error() . mysql_close($conexao));
}
?>
```

Observe que a função ***mysql_close()*** necessita de um parâmetro que identifique a conexão a ser encerrada. Se nada for informado, ***mysql_close*** encerrará a conexão corrente.

8.1.4 Inserindo dados no MySQL

A instrução ***INSERT INTO*** é utilizada para inserir novos registros em uma tabela do tipo SQL, como é o caso do MySQL.

Sintaxe

É possível escrever a instrução ***INSERT INTO*** de duas formas: a primeira não especifica o nome da coluna onde os dados serão inseridos, apenas os seus valores:

```
INSERT INTO nome_tabela  
VALUES (valor1, valor2, valor3,...)
```

A segunda especifica tanto os nomes das colunas quanto os valores a serem inseridos:

```
INSERT INTO nome_tabela (coluna1, coluna2, coluna3,...)  
VALUES (valor1, valor2, valor3,...)
```

Para o PHP executar as instruções acima, devemos usar a função ***mysql_query()***. Esta função é utilizada para enviar uma consulta ou um comando a uma conexão com o MySQL.

Exemplo

No capítulo anterior criamos uma tabela chamada “alunos”, com quatro colunas: “entrada, nome, endereço e cidade”. Vamos usar a mesma tabela, neste exemplo, o qual adiciona dois novos registros para a tabela de “alunos”:

```

<?php
$conexao = mysql_connect("localhost","sallum",""); // Solicita conexão
if (!$conexao) {
    die('Falha na conexão: ' . mysql_error());
}

mysql_select_db("Turma", $conexao); //Seleciona o BD Turma em localhost

//Os comando abaixo inserem dois registros na tabela alunos
mysql_query("INSERT INTO alunos (entrada, nome, endereço, cidade)
VALUES ('2011', 'Joana A. B.', 'Av. Sábio Salomão, 35', 'Divinópolis')");
mysql_query("INSERT INTO alunos (entrada, nome, endereço, cidade)
VALUES ('2010', 'Andréa A.', 'Av. Sol, 33', 'Betim')");

mysql_close($conexao); // finaliza a conexão com o BD
?>

```

A nossa tabela ficará como a Tabela 8.3 a seguir.

Tabela 8.3: Listagem de registros de nomes.			
Entrada	Nome	Endereço	Cidade
2010	José A.W. C.	R. das flores, 34	Belo Horizonte
2011	Maria K. J.	Av. imaginária, 123	Formiga
2011	Antônio L.	R. da invenção, 10	Cajuru
2011	Joana A. B.	Av. Sábio Salomão, 35	Divinópolis
2010	Andréa A.	Av. Sol, 33	Betim

Fonte: Elaborada pelo autor

8.1.5 Inserindo dados de formulário em um BD

Agora vamos criar um formulário HTML que pode ser usado para adicionar novos registros para a tabela alunos.

```
<html>
<body>

<form action="insere_dados.php" method="post">
Dada de Entrada: <input type="text" name="f_data">
Nome: <input type="text" name="f_nome">
Endereço: <input type="text" name="f_ender">
Cidade:<input type="text" name="f_cidade">
<input type="Enviar">
</form>

</body>
</html>
```

Ao clicarmos no botão "Enviar" do formulário HTML, no exemplo acima, os dados serão enviados para o programa "insere_dados.php".

O arquivo "insere_dados.php" recupera os dados enviados por **\$_POST** e se conecta ao banco de dados "Turma", no servidor "localhost". Daí, este programa utiliza a função **mysql_query()** para inserir os dados recebidos de **\$_POST** e armazena-os na tabela alunos.

Eis a seguir o programa "insere_dados.php":

```

<?php
$conexao = mysql_connect("localhost","sallum","");
if (!$conexao) {
    die('Falha na conexão: ' . mysql_error());
}

mysql_select_db("Turma", $conexao);

// Comando abaixo insere os dados capturados
$insere="INSERT INTO alunos(Entrada, Nome, Endereço, Cidade)
VALUES ('$_POST[f_data]','$_POST[f_nome]','$_POST[f_ender]','$_POST[f_cidade]')";

if (!mysql_query($insere, $conexao)) { // Efetua a inserção e testa se houve
êxito
    die('Erro: ' . mysql_error());
}
echo "1 registro inserido."; // Se passou pelo teste anterior, mostra mensa-
gem

mysql_close($conexao); //finaliza conexão com servidor
?>

```

8.1.6 Selecionando dados em um BD

A instrução **SELECT** é usada para selecionar dados de um banco de dados.

Sintaxe

```

SELECT coluna(s)
FROM nome_tabela

```

Para executar a instrução acima, devemos usar a função **mysql_query()**. Esta função é útil para enviar uma consulta ou um comando a uma conexão com o MySQL.

Exemplo

O exemplo a seguir seleciona todos os dados armazenados na tabela alunos (O caractere * seleciona todas as colunas da tabela):

```

<?php
$con = mysql_connect("localhost","sallum","");
if (!$conexao) {
    die("Falha na conexão: " . mysql_error());
}

mysql_select_db("Turma", $conexao); // Seleciona BD Turma do servidor
Localhost
// Efetua consulta de todos os dados de todas as colunas
$resultado = mysql_query("SELECT * FROM alunos");

// Varre linha a linha cada registro recuperado, mostrando apenas os nomes
e cidades
while($linha = mysql_fetch_array($resultado)) {
    echo $linha['Nome'] . " " . $linha['Cidade'];
    echo "<br>";
}

mysql_close($conexao); //Finaliza conexão com servidor
?>

```

O exemplo acima armazena os dados retornados pela função **mysql_query()** na variável **\$resultado**.

Em seguida, usamos a função **mysql_fetch_array()** para retornar as linhas do conjunto de registros. Este comando funciona como uma matriz. Cada chamada para **mysql_fetch_array()** retornará a próxima linha no conjunto de registros. Cada iteração produzida pelo laço **while** recuperará a próxima linha do conjunto de todos os registros recuperados. Para imprimir o valor de cada linha, usamos a variável **\$linha** ['Nome'] e **\$linha** ['cidade'].

A saída do código acima será tal como é ilustrada na Tabela 8.4 a seguir.

Tabela 8.4: Listagem de registros de nomes e respectivas cidades.

Nome	Cidade
José A.W. C.	Belo Horizonte
Maria K. J.	Formiga
Antônio L.	Cajuru
Joana A. B.	Divinópolis
Andréa A.	Betim

Fonte: Elaborada pelo autor

8.1.7 Utilizando tabela HTML para receber os resultados da pesquisa

O exemplo a seguir seleciona os mesmos dados do exemplo acima, mas vai exibi-los em uma tabela HTML:

```
<?php
$con = mysql_connect("localhost","sallum","");
if (!$conexao) {
    die('Falha na conexão: ' . mysql_error());
}

mysql_select_db("Turma", $conexao); // Seleciona BD Turma do servidor Localhost
// Efetua consulta de todos os dados de todas as colunas
$resultado = mysql_query("SELECT * FROM alunos");

// Varre linha a linha cada registro recuperado, mostrando apenas os nomes e cidades
while($linha = mysql_fetch_array($resultado)) {
    echo $linha['Nome'] . " " . $linha['Cidade'];
    echo "<br>";
}

echo "<table border='1'>
<tr>
<th>Nome</th>
<th>Cidade</th>
</tr>";
while($linha = mysql_fetch_array($resultado)) {
    echo "<tr>";
    echo "<td>" . $linha['FirstName'] . "</td>";
    echo "<td>" . $linha['cidade'] . "</td>";
    echo "</tr>"; }

echo "</table>";

mysql_close($conexao);
?>
```

A saída do código acima será conforme é ilustrada na Tabela 8.5 a seguir.

Tabela 8.5: Listagem de registros de nomes e respectivas cidades.	
Nome	Cidade
José A.W. C.	Belo Horizonte
Maria K. J.	Formiga
Antônio L.	Cajuru
Joana A. B.	Divinópolis
Andréa A.	Betim

Fonte: Elaborada pelo autor

8.1.8 Utilizando a cláusula **WHERE**

A cláusula **WHERE** é usada para filtrar registros mediante algum critério de seleção. Em outras palavras, esta cláusula é usada para extrair apenas os registros que satisfazem a algum critério especificado.

Sintaxe

```
SELECT coluna(s)
FROM nome_tabela
WHERE coluna operador valor
```

Para executarmos a instrução acima, devemos utilizar a função **mysql_query()**. Esta função é utilizada para enviar uma consulta ou um comando a uma conexão com o MySQL.

Exemplo

O exemplo a seguir seleciona todas as linhas da tabela alunos com o seguinte critério: ONDE Entrada = '2011':

```
<?php
$con = mysql_connect("localhost","sallum","");
if (!$conexao) {
    die('Falha na conexão: ' . mysql_error());
}

mysql_select_db("Turma", $conexao); // Seleciona BD Turma do servidor Localhost
// Seleciona todos os dados de todas as colunas onde Entrada se deu em 2011
$resultado = mysql_query("SELECT * FROM alunos WHERE Entrada = '2011'");

while($linha = mysql_fetch_array($resultado)) {
    echo $linha['Entrada'] . " " . $linha['Nome'];
    echo "<br>";
}
?>
```

A saída do código acima será (Tabela 8.6):

Tabela 8.6: Listagem de registros mediante critério de seleção.	
Entrada	Nome
2011	Maria K. J.
2011	Antônio L.
2011	Joana A. B.

Fonte: Elaborada pelo autor

Note que na tabela resultado (Tabela 8.6) são listados apenas os registros cujas entradas ocorreram em 2011.

Note, também, que do dado 2011 é colocado entre aspas simples, pois é uma *string* da coluna (ou campo) Entrada.

8.1.9 Ordenando registros pela palavra-chave

A palavra-chave **ORDER BY** é utilizada para classificar os dados em um conjunto de registros por ordem crescente ou decrescente do campo-chave da tabela. Entretanto, a palavra-chave **ORDER BY** classifica os registros em ordem crescente por padrão. Mas, se desejarmos classificar os registros em ordem decrescente, podemos usar a palavra chave **DESC** à frente da palavra-chave **ORDER BY**, conforme sintaxe apresentada a seguir.

Sintaxe

```
SELECT coluna(s)
FROM nome_tabela
ORDER BY coluna(s) ASC | DESC
```

Exemplo

O exemplo a seguir seleciona todos os dados armazenados na tabela alunos e classifica-os pela coluna nome, colocando-os em ordem alfabética crescente (A-Z):

```
<?php
$con = mysql_connect("localhost","sallum","");
if (!$conexao) {
    die('Falha na conexão: ' . mysql_error());
}

mysql_select_db("Turma", $conexao); // Seleciona BD Turma do servidor Localhost
// Seleciona todos os dados de todas as colunas, ordenados por Nome decrescente
$result = mysql_query("SELECT * FROM alunos ORDER BY Nome ASC");
while($linha = mysql_fetch_array($resultado)) {
    echo $linha['Entrada'] . " " . $linha['Nome'] . " " . $linha['Endereco']
    . " " .
        $linha['Cidade'];
    echo "<br>";
}
?>
```

A saída do código acima deverá ser tal como a Tabela 8.7 a seguir.

Tabela 8.7: Listagem de registros mediante ordem alfabética dos nomes.			
Entrada	Nome	Endereço	Cidade
2010	Andréa A.	Av. Sol, 33	Betim
2011	Antônio L.	R. da invenção, 10	Cajuru
2011	Joana A. B.	Av. Sábio Salomão, 35	Divinópolis
2010	José A.W. C.	R. das flores, 34	Belo Horizonte
2011	Maria K. J.	Av. imaginária, 123	Formiga

Fonte: Elaborada pelo autor

Observe que a tabela resultado acima lista os dados ordenados por nome, em que cada coluna ou característica adjacente acompanha o seu respectivo nome.



8.1.10 Alterando os dados de uma tabela

A instrução **UPDATE** é utilizada para modificar e atualizar os dados existentes em uma tabela.

Sintaxe

```
UPDATE nome_tabela  
SET coluna1=valor, coluna2=valor2,...  
WHERE coluna = valor
```

Observe a cláusula *WHERE* no *UPDATE*. A cláusula *WHERE* especifica o registro ou registros que devem ser atualizados. Se omitirmos a cláusula *WHERE*, todos os registros serão atualizados!

Para executar a instrução acima, devemos usar a função **mysql_query()**. Como já sabemos, esta função é utilizada para enviar uma consulta ou um comando a uma conexão com o MySQL.

Exemplo

Anteriormente, criamos uma tabela chamada “alunos”. Vamos utilizá-la para visualizar o exemplo que atualiza alguns dados na tabela alunos, conforme mostrado a seguir:

```

<?php
$con = mysql_connect("localhost","sallum","");
if (!$conexao) {
    die('Falha na conexão: ' . mysql_error());
}

mysql_select_db("Turma", $conexao); // Seleciona BD Turma do servidor Localhost
// Seleciona todos os dados de todas as colunas, ordenados por Nome decrescente

mysql_query("UPDATE alunos SET Cidade = 'Divinópolis'
WHERE Entrada = '2011'");
while($linha = mysql_fetch_array($resultado)) {
    echo $linha['Entrada'] . " " . $linha['Nome'] . " " . $linha['Endereco']
    . " " .
    $linha['Cidade'];
    echo "<br>";
}
?>

```

Após a atualização, a tabela “alunos” ficará assim, tal como ilustrado na Tabela 8.8 a seguir.

Tabela 8.8: Listagem de registros mediante ordem alfabética dos nomes com dados atualizados.

Entrada	Nome	Endereço	Cidade
2010	Andréa A.	Av. Sol, 33	Betim
2011	Antônio L.	R. da invenção, 10	Divinópolis
2011	Joana A. B.	Av. Sábio Salomão, 35	Divinópolis
2010	José A.W. C.	R. das flores, 34	Belo Horizonte
2011	Maria K. J.	Av. imaginária, 123	Divinópolis

Fonte: Elaborada pelo autor

Como podemos observar na tabela resultado (Tabela 8.8) anteriormente listada, todos os registros de alunos com entrada em 2011 tiveram suas cidades de origem alteradas para “Divinópolis”.

8.1.11 Como excluir dados em um banco de dados

A instrução **DELETE** é usada para excluir registros em uma tabela.

Sintaxe

```
DELETE FROM nome_tabela  
WHERE coluna = valor
```

Observe a cláusula **WHERE** na sintaxe **DELETE**. A cláusula **WHERE** especifica o registro ou registros que devem ser excluídos. Mas atenção: Se omitirmos a cláusula **WHERE**, todos os registros serão excluídos!

Para executarmos a instrução acima, devemos usar a função **mysql_query()**.



Exemplo

Utilizando a nossa tabela de alunos, vamos mostrar um exemplo, conforme a seguir, sobre a exclusão de alunos, onde entrada = '2010':

```
<?php  
$con = mysql_connect("localhost", "sallum", "");  
if (!$conexao) {  
    die('Falha na conexão: ' . mysql_error());  
}  
  
mysql_select_db("Turma", $conexao); // Seleciona BD Turma do servidor  
Localhost  
// Deleta todos os dados que têm Entrada igual '2010'  
mysql_query("DELETE FROM alunos WHERE Entrada='2010'");  
  
mysql_close($conexao);  
?>
```

Depois da eliminação, a tabela deverá assumir a seguinte forma (Tabela 8.9):

Tabela 8.9: Listagem de registros mediante ordem alfabética com registros deletados.

Entrada	Nome	Endereço	Cidade
2011	Antônio L.	R. da invenção, 10	Divinópolis
2011	Joana A. B.	Av. Sábio Salomão, 35	Divinópolis
2011	Maria K. J.	Av. imaginária, 123	Divinópolis

Fonte: Elaborada pelo autor

8.1.12 Configurando o MySQL para execução no PHP

O comportamento das funções MySQL é afetado pelas configurações no arquivo **php.ini**, o qual é o arquivo de configurações da linguagem e pode ser alterado pelo programador.

O Quadro 8.2 fornece algumas opções de configuração do MySQL:

Quadro 8.2: Opções de configuração do MySQL.			
Nome	Default	Descrição	Variável de ambiente
<i>mysql.allow_persistent</i>	"1"	Habilita ou não conexões persistentes.	PHP_INI_SYSTEM
<i>mysql.max_persistent</i>	"-1"	Estabelece o número máximo de conexões persistentes por processo.	PHP_INI_SYSTEM
<i>mysql.trace_mode</i>	"0"	Quando ajustado para "1", os avisos e erros do SQL serão exibidos. Disponível desde o PHP 4.3.	PHP_INI_ALL
<i>mysql.default_port</i>	NULL	O número padrão da porta TCP a ser utilizada.	PHP_INI_ALL
<i>mysql.default_socket</i>	NULL	O nome padrão do socket para utilizar. Disponível desde o PHP 4.0.1.	PHP_INI_ALL
<i>mysql.default_host</i>	NULL	Define o servidor padrão (não se aplica em SQL safe mode).	PHP_INI_ALL
<i>mysql.default_user</i>	NULL	Define o nome de usuário padrão (não se aplica em SQL safe mode).	PHP_INI_ALL
<i>mysql.default_password</i>	NULL	Define uma senha padrão (não se aplica em SQL safe mode).	PHP_INI_ALL
<i>mysql.connect_timeout</i>	"60"	Determina o tempo limite de conexão em segundos.	PHP_INI_ALL

Fonte: http://www.php.net/manual/pt_BR/ref.mysql.php

O Quadro 8.3, a seguir, lista as funções mais utilizadas em MySQL através do PHP.

Quadro 8.3: Funções mais utilizadas em MySQL.		
Função	Descrição	PHP
<code>mysql_affected_rows()</code>	Retorna o número de linhas afetadas na operação anterior do MySQL.	3
<code>mysql_change_user()</code>	Em desuso. Modifica o usuário para a conexão atual MySQL.	3
<code>mysql_client_encoding()</code>	Retorna o nome do conjunto de caracteres para a conexão atual.	4
<code>mysql_close()</code>	Fecha uma conexão não persistente MySQL.	3
<code>mysql_connect()</code>	Abre uma conexão não persistente MySQL.	3
<code>mysql_create_db()</code>	Em desuso. Cria um novo banco de dados MySQL. Use <code>mysql_query()</code> ao invés.	3
<code>mysql_data_seek()</code>	Move o ponteiro de registro.	3
<code>mysql_db_name()</code>	Retorna um nome de banco de dados de uma chamada para <code>mysql_list_dbs()</code> .	3
<code>mysql_db_query()</code>	Em desuso. Envia uma consulta MySQL. Use <code>mysql_select_db()</code> e <code>mysql_query()</code> ao invés.	3
<code>mysql_drop_db()</code>	Em desuso. Exclui um banco de dados MySQL. Use <code>mysql_query()</code> ao invés.	3
<code>mysql_errno()</code>	Retorna o número de erro da operação MySQL passado.	3
<code>mysql_error()</code>	Retorna a descrição de erro da operação MySQL passado.	3
<code>mysql_escape_string()</code>	Em desuso. Escapa uma string para uso com o <code>mysql_query</code> . Use <code>mysql_real_escape_string()</code> ao invés.	4
<code>mysql_fetch_array()</code>	Retorna uma linha de um conjunto de registros como uma matriz associativa e/ou uma matriz numérica.	3
<code>mysql_fetch_assoc()</code>	Retorna uma linha de um conjunto de registros como uma matriz associativa.	4
<code>mysql_fetch_field()</code>	Retorna informações da coluna de um conjunto de registros como um objeto.	3
<code>mysql_fetch_lengths()</code>	Retorna o comprimento do conteúdo de cada campo em uma linha do resultado.	3
<code>mysql_fetch_object()</code>	Retorna uma linha de um conjunto de registros como um objeto.	3
<code>mysql_fetch_row()</code>	Retorna uma linha de um conjunto de registros como uma matriz numérica.	3
<code>mysql_field_flags()</code>	Retorna os <i>flags</i> associados a um campo em um conjunto de registros.	3
<code>mysql_field_len()</code>	Retorna o comprimento máximo de um campo em um conjunto de registros.	3
<code>mysql_field_name()</code>	Retorna o nome de um campo em um conjunto de registros.	3
continua		

<i>mysql_field_seek()</i>	Move o ponteiro do resultado para um campo especificado.	3
<i>mysql_field_table()</i>	Retorna o nome da tabela que o campo especificado esta.	3
<i>mysql_field_type()</i>	Retorna o tipo de um campo em um conjunto de registros.	3
<i>mysql_free_result()</i>	Free resultado da memória.	3
<i>mysql_get_client_info()</i>	Retorna informações sobre o cliente MySQL.	4
<i>mysql_get_host_info()</i>	Retorna informações do servidor MySQL.	4
<i>mysql_get_proto_info()</i>	Retorna informações do protocolo MySQL.	4
<i>mysql_get_server_info()</i>	Retorna informações sobre o servidor MySQL.	4
<i>mysql_info()</i>	Retorna informações sobre a última consulta.	4
<i>mysql_insert_id()</i>	Retorna o ID AUTO_INCREMENT gerado pela operação INSERT anterior.	3
<i>mysql_list_dbs()</i>	Lista as bases de dados disponíveis em um servidor MySQL.	3
<i>mysql_list_fields()</i>	Em desuso. Lista campos de tabela MySQL. Use <i>mysql_query()</i> ao invés.	3
<i>mysql_list_processes()</i>	Lista os processos MySQL.	4
<i>mysql_list_tables()</i>	Em desuso. Lista tabelas em um banco de dados MySQL. Use <i>mysql_query()</i> ao invés.	3
<i>mysql_num_fields()</i>	Retorna o número de campos de um <i>recordset</i> .	3
<i>mysql_num_rows()</i>	Retorna o número de linhas em um conjunto de registros.	3
<i>mysql_pconnect()</i>	Abre uma conexão persistente MySQL.	3
<i>mysql_ping()</i>	Verifica a conexão de um servidor ou reconecta se não houver conexão.	4
<i>mysql_query()</i>	Executa uma consulta em um banco de dados MySQL.	3
<i>mysql_real_escape_string()</i>	Escapa uma <i>string</i> para usar em SQLs.	4
<i>mysql_result()</i>	Retorna o valor de um campo em um conjunto de registros.	3
<i>mysql_select_db()</i>	Define o banco de dados MySQL ativos.	3
<i>mysql_stat()</i>	Retorna o <i>status</i> atual do sistema do servidor MySQL.	4
<i>mysql_tablename()</i>	Em desuso. Retorna o nome da tabela do campo. Use <i>mysql_query()</i> ao invés.	3
<i>mysql_thread_id()</i>	Retorna a identificação da <i>thread</i> atual.	4
<i>mysql_unbuffered_query()</i>	Executa uma consulta em um banco de dados MySQL (sem <i>extracção/buffering</i> o resultado).	4
Conclusão		

Fonte: http://www.php.net/manual/pt_BR/ref.mysql.php.

A função ***mysql_fetch_array()*** usa uma constante para os diferentes tipos de matrizes de resultado. O Quadro 8.4, a seguir, lista as constantes definidas:

Quadro 8.4: Constantes definidas do MySQL.	
Constante	Descrição
MYSQL_ASSOC	As colunas são retornadas na matriz com o nome do campo como índice da matriz.
MYSQL_BOTH	As colunas são retornadas na matriz tendo ambos os índices, numérico e o nome do campo, como índice da matriz.
MYSQL_NUM	As colunas são retornadas na matriz tendo um índice numérico (índice começa em 0).

Fonte: http://www.php.net/manual/pt_BR/mysql.constants.php

Resumo

Estudamos nesta aula a conexão ao banco de dados MySQL. Vimos como estabelecer conexão com um servidor de banco de dados e como criar tabelas de dados nesse servidor utilizando instruções PHP.

Entendemos com esta aula como inserir, alterar, excluir e selecionar dados mediante os vários critérios de nossa escolha.

Finalmente, vimos como configurar o gerenciador de banco de dados, MySQL para sua execução no PHP.

Atividades de aprendizagem

1. Digite todos os códigos exemplos desta aula e execute-os, comparando os resultados obtidos com as respectivas saídas aqui ilustradas.
2. Suponha que exista no MySQL o banco de dados chamado de TESTE e que nesse banco exista a tabela CLIENTES. Suponha, também, que esta tabela possua os seguintes campos (colunas): Nome, Sexo e Peso. Crie um programa em PHP que tenha as quatro seguintes funções para operações em Banco de Dados:
 - a) função de conexão com um banco de dados que receba os seguintes parâmetros:
 - i. Nome do servidor;
 - ii. Nome do usuário;
 - iii. Senha.

- b) função que efetue consulta à tabela e que receba os seguintes parâmetros:
- i. Nome do banco de dados (TESTE)
 - ii. Nome da tabela (CLIENTES)
 - iii. Condição de seleção para consulta. Por exemplo: **"WHERE Sexo='F' "**
- c) função que exclua registros na tabela e receba os seguintes parâmetros:
- i. Nome do banco de dados (TESTE)
 - ii. Nome da tabela (CLIENTES)
 - iii. Condição de seleção para exclusão. Por exemplo: **"WHERE Peso > 100"**
- d) função que feche e encerre a conexão com o banco de dados.

```
<?php
function conecta($serv, $usu, $pass) {

    // corpo da função;

}

function consulta($BD, $tab, $condicao) {

    // corpo da função;

}

function exclui($BD, $tab, $condicao) {

    // corpo da função;

}

function fecha($ID) {

    // corpo da função;

}

$ID = conecta("localhost", "fulano", "123");
$CN = consulta("TESTE", "CLIENTE", "WHERE Sexo = 'F'");
$EX = exclui("TESTE", "CLIENTE", "WHERE Peso > 100");
fecha($ID);
?>
```

Figura 8.1: Estrutura do código-fonte para elaboração das funções do exercício

Fonte: Elaborada pelo autor

3. O código mostrado na Figura 8.1 inicia o processo solicitado nesta questão. Desenvolva os corpos das funções solicitadas (a, b, c e d) e poste-as no fórum criado para esta atividade, juntamente com os códigos desenvolvidos na primeira questão.

Referências

HERRINGTON, Jack D. **PHP hacks**: dicas e ferramentas para a criação de web sites dinâmicos. Tradução João Tortello. Porto Alegre: Bookman, 2008.

GILMORE, W. Jason. **Dominando PHP e MySQL**: do iniciante ao profissional. Tradução da 3ª ed. Raquel Marques e Lúcia Kinoshita. Rio de Janeiro: AltaBooks, 2008.

MELO, Alexandre Altair; NASCIMENTO, Maurício, G. F. **PHP Profissional**. São Paulo: Novatec, 2007.

NIEDERAUER, Juliano. **Guia de consulta rápida**. São Paulo: Novatec, 2007.

PHPEDITOR. **Editor gratuito para PHP**. Disponível em: <http://filestore.softwaredownloadwebsite.com/h92/743753-phpeditor_setup_en.exe>. Acesso em: 21 fev. 2012.

RAMOS, Ricardo; SILVA, Joel; ÁLVARO, Alexandre; AFONSO, Ricardo. **PHP para profissionais**. São Paulo: Digerati Books, 2007.

TANSLEY, David. **Como criar web pages rápidas e eficientes usando PHP e MySQL**. Rio de Janeiro: Ciência Moderna, 2002.

WAMP, **WampServer 32 e 64 bits**. Disponível em: < <http://www.wampserver.com/>>. Acesso em: 28 fev. 2012.

Currículo do professor-autor



William Geraldo Sallum é professor da disciplina de AW1 ou Aplicativos para Web I, deste curso. É doutorando em Ensino de Ciências e Matemática pela UNICSUL. Possui mestrado em Tecnologia da Informação pelo Centro Federal de Educação Tecnológica de Minas Gerais – CEFET-MG (2002). É especialista em Análise de Sistemas *Lato Sensu* pelo Cenex. É graduado em Matemática pela Faculdade de Filosofia Ciências e Letras de Belo Horizonte – UNIBH. Possui habilitação para o ensino de Física e Desenho. Atualmente é professor do CEFET-MG, no Curso de Informática. É membro titular da Comissão Permanente de Pessoal Docente do CEFET-MG e atua como subcoordenador do Curso de Pós-graduação do CEFET-MG.

Tem experiência na área de Ciência da Computação, com ênfase em Sistemas de Informação, atuando principalmente nos seguintes temas: recuperação de informação em documentos textuais semiestruturados e estruturados; desenvolvimento de sistemas para área acadêmica e comercial; e desenvolvimento de aplicativos para *web*.



ISBN 978-85-99872-27-7

