PHP
e
MySQL

Autor: Leandro Correa dos Santos
leandro.admo@hotmail.com

Sumário

Revisão de HTML

HTML significa Hiper Text Markup Language – Linguagem de Marcação de Hiper Texto. Com ela, podemos formatar documentos inteiros para exibição na Internet, transformado textos simples em hipertexto.

A Internet é nada mais do que uma grande rede de computadores, formada por outras redes menores, compartilhando informações. Para que essas informações sejam compartilhadas entre os mesmos, é preciso que sejam disponibilizadas de forma que todo e qualquer computador possa interpretar. HTML é uma linguagem padrão para divulgação de documentos na rede, portanto, qualquer computador deve ser capaz de interpretá-lo.

Um documento escrito em HTML é, em geral, chamado de página web. O conjunto dessas páginas forma um site (lugar, em inglês). Seu conteúdo é chamado de hipertexto, pois pode ser compreendido por qualquer computador, enquanto que o texto comum pode ser visto em um computador de uma forma e em outro de outra forma. Um site pode conter quaisquer informações que se queira disponibilizar via internet, mas para que ele seja visto, é preciso que seja publicado. Para isso, existem os servidores web. Trata-se de computadores que armazenam e disponibilizam esses sites para visitação, a custos relativamente baixos.

Além disso, deve-se usar um programa capaz de interpretar o código HTML da página, aplicar sua formatação ao documento e exibir o resultado. Este programa é chamado de browser ou navegador. Os navegadores mais conhecidos são o Mozilla Firefox, Internet Explorer, Opera e Netscape.

HTML é uma linguagem estática, ou seja, uma vez escrito, sua estrutura permanece inalterada.

Estrutura HTML

HTML é composto por tag's, que são marcações delimitadas pelos símbolos "<" e ">", usados para indicar uma formatação. Ex: Todo documento HTML deve iniciar com a tag <html> e terminar com a tag </html>. Essas tags delimitam o documento, sendo que o que estiver fora delas não será considerado um documento HTML, e portanto não será interpretado como hiper texto, e sim como texto comum. Ex:

```
<html>
...
página web
...
</html>
```

Um documento HTML é formado basicamente por cabeçalho e corpo. No cabeçalho, fornecemos informações, como o nome da página, autor, palavras-chave para pesquisa, etc. Essas informações não ficam expostas no navegador do usuário.

Cabeçalhos iniciam com a tag <head> e terminam com </head>. A maioria tas tags HTML possuem início e final.

Dentro do cabeçalho da página definimos o título da mesma. O título da página será

o conteúdo entre as tags <title> e </title>. Esse título aparecerá na barra de título do navegador. Vejamos nosso exemplo:

```
<html>
  <head>
  <title>Minha Página Web!!!</title>
  </head>
  ...
</html>
```

Digite o código acima e salve como pagina.html e abra-o usando o navegador de sua preferência.

Agora, definiremos o corpo da página. Todo seu conteúdo será visualizado dentro do navegador. O corpo da página começa com a tag <body> e termina com </body>.

```
<html>
  <head>
    <title>Minha Página Web!!!</title>
  </head>
  <body>
    ...
  </body>
</html>
```

Altere o arquivo pagina.html para que fique como o código acima, depois salve. Já temos a estrutura de um documento HTML. Agora, vamos adicionar conteúdo a esta página.

Atributos

A grande maioria das tag's HTML possuem atributos, que são valores passados a elas para que as mesmas assumam uma formatação diferente. Exemplo: Se quisermos o corpo da página azul, podemos alterar o atributo *bgcolor* da tag *body,* alterando seu valor para blue.

```
<html>
  <head>
    <title>Minha Página Web!!!</title>
  </head>
  <body bgcolor=”blue”>
    ...
  </body>
</html>
```

Principais Tag's HTML

Vamos agora conhecer as tag's mais utilizadas na criação de uma página HTML e seus principais atributos. Há uma vasta quantidade de tag's que podem ser utilizadas, dependendo do resultado desejado. Aqui, faremos apenas um breve resumo das principais, mas sinta-se à vontade para pesquisar e descobrir novas tag's e atributos para adicionar à sua página web.

Digite os exemplos dentro do corpo da página do exemplo acima. Salve, depois abra-o no navegador de sua preferência, ou atualize o navegador usando a tecla F5.

<!-- e -->

Usados para indicar um comentário. comentários não são exibidos no navegador. São utilizados para uma melhor orientação pelo código HTML.

```
<body> <!-- início do corpo da página -->
<!-- colocar um título aqui -->
</body> <!-- fim do corpo da página -->
```

<Hn>

Define um título. n pode ser qualquer valor entre 1 e 6. Quanto maior o número, menor o tamanho do título.

```
<h1>Título tamanho 1</h1>
<h2>Título tamanho 2</h2>
```


Adiciona uma quebra de linha ao texto. forçando-o a continuar na linha de baixo. Mesmo que o texto dentro do documento ocupe apenas uma linha, o navegador fará a quebra de linha sempre que encontrar a tag
.

```
Este é um texto que <br> recebeu uma quebra de linha <br> aqui
```

<p>

Inicia um novo parágrafo, separando assim, blocos de texto distintos.

```
<p> Iniciei um parágrafo. Dentro dele, posso adiconar qualquer conteúdo e usar
<br> quebra de linha para que não ocupe apenas uma linha. <br> Quando o
parágrafo terminar, eu posso simplesmente iniciar outro <p> Parágrafos não
precisam ser encerrados, pois o início do próximo é o final do anterior.</p>
```

<a>

Essa tag indica a presença de um link. Links são ligações entre documentos HTML. Eles direcionam o navegador para um ponto dentro do documento, para outro documento

HTML, para outro arquivo ou até para outro site. Seu principal atributo é *href,* que determina para qual documento o link é direcionado. O conteúdo contido entre as tag's <a> e torna-se o link em si. Pode ser uma frase, uma imagem ou um campo do documento.

```
<a href="pagina2.html">Pagina 2</a>
<a href="pasta/arquivo.jpg">Abre uma imagem no navegador</a>
<a href="http://www.umsitequalquer.com">Direciona para outro site</a>
```

<font>

Usada para formatar texto do documento. Seus principais atributos são:

- **face:** Nome da fonte;
- **size:** tamanho da fonte;
- **color:** cor da fonte;

```
<font face="Times New Roman" size="14" color="red">Texto Formatado</font>
```

<table>

Tabelas são estruturas utilizadas para organizar e posicionar os elementos na página. Tabelas contém linhas, que por sua vez, contém células. Dentro das células, adicionamos o conteúdo da página. Seus principais atributos são:

- **bgcolor:** cor de fundo
- **width:** largura (pode ser indicada em pixels ou em percentuais)
- **height:** altura
- **border:** espessura da borda

<tr>

Marca o início de uma linha da tabela

<td>

Marca o início de uma célula dentro de uma linha. Seus principais atributos são:

- **cellspacing:** espaço entre as células
- **cellpadding:** distância entre o conteúdo e a borda da célula
- **colspan:** indica que a célula ocupa o espaço de n células (colunas)
- **rowspan:** ocupa o espaço de n linhas

```
<table border="1" width="100%" height="500" bgcolor="gray">
  <tr>
   <td cellpadding="0" colspan="3" bgcolor="pink">Célula Rosa</td>
  </tr>
  <tr>
   <td cellpadding="0" cellspacing="0" bgcolor="blue">Célula Azul</td>
```

```
   <td cellpadding="0" cellspacing="0" bgcolor="red">Célula Vermelha</td>
   <td cellpadding="0" cellspacing="0" bgcolor="green">Célula Verde</td>
  </tr>
</table>
```

Formulários

Formulários são estruturas usadas para que o visitante possa interagir com o site, enviando informações para o site. Essas informações são processadas por scripts que capturam as informações enviadas e as processam, retornando um resultado, gravando-as no banco de dados do site, etc....

O campo pertencente ao formulário é delimitado pelas tag's <form> e </form>. Dentro dessas tag's inserimos os elementos do formulário (campos de texto, botões, etc). Esses elementos irão capturar a informação fornecida pelo visitante e enviar ao script para processamento. Cada elemento possui um nome que é referenciado pelo script na captura de informações.

Os principais atributos da tag form são:

- **action:** indica o script que irá processar as informações enviadas pelo formulário
- **method:** indica o método de envio das informações

As informações de um formulário podem ser enviadas através de dois métodos: GET e POST.

Método GET

Usado para enviar pequenas quantidades de informação, visto que sua capacidade de envio é limitada. O método GET usa a barra de endereços do navegador para enviar informações para o servidor. Possui um limite de 256 caracteres, sendo que o excedente pode (e acaba) se perdendo durante a transmissão. É o método de envio mais rápido e também o menos seguro, pois as informações enviadas através deste método ficam explicitamente visíveis na barra de endereços, o que o torna um método pouco seguro, visto que informações confidenciais (senhas, números de cartões, etc) tornam-se visíveis a qualquer pessoa. A URL do site assume o seguinte aspecto:

```
http://www.sitequalquer.com?informação1=valor1&informação2=valor2
```

O sinal "?" indica que é o fim do endereço do site e o início das informações que serão enviadas através do método GET. "Informação1" e "Informação2" são as variáveis, que contém os valores que serão enviados para o servidor. "Valor1" e "valor2" são os valores das respectivas variáveis. O sinal "&" é usado para separar uma variável da outra.

Método POST

Oposto ao anterior, as informações são enviadas de forma mais segura, possibilitando o uso de criptografia e outros recursos de segurança. Não há um limite para a quantidade de informações que pode ser enviada através deste método, porém, é o mais lento. Para usá-lo.

Formulários usando GET e POST

Para utilizar qualquer um dos dois métodos mencionados acima para enviar informações de um formulário, devemos declará-lo dentro da tag <form>, como nos exemplos a seguir:

```
<form action="script.php" method="GET"> <!-- usando GET -->
<form action="script.php" method="POST"> <!-- usando POST -->
```

Elementos do Formulário

A seguir, veremos os principais elementos de um formulário. Os elementos do formulário possuem um atributo comum chamado name. Ele indica o nome de cada elemento. Este nome será tratado pelo script como uma variável (algo que contém uma informação), portanto, cada elemento deve ter um atributo name distinto, para que seja diferenciado pelo script.

Campo de Texto

Permite a entrada de texto pelo usuário. Usado para preenchimento de nomes, endereços e outras informações simples. Seus principais atributos são:

- name: nome do campo de texto
- value: valor padrão para o campo
- size: tamanho do campo
- maxlenght: quantidade máxima de caracteres aceitos pelo campo

```
Nome: <input type="text" name="nome_do_visitante" size="50" maxlenght="50"
value="Digite aqui seu nome">
```

Note que usamos a tag <input>. Esta tag pode assumir formas diversas, como campos de texto, senhas, botões, etc. O que torna-a um campo de texto é o valor do atributo type. Quando indicamos o valor text, a tag input tornou-se um campo de texto. Outras formas desta tag serão vistas a seguir.

Campo de Senha

Semelhante ao campo de texto, porém não aceita o atributo *value.* Utilizado para a digitação de informações confidenciais, principalmente senhas.

```
Senha: <input type="password" name="senha" size="12" maxlenght="12">
```

Área de Texto

Abre uma área para digitação de texto. Devemos informar o número de linhas e colunas que a área de texto irá ocupar. Isto é feito através dos atributos *rows* (linhas) e *cols* (colunas).

```
Comentário: <textarea name="comentario" rows="10"
cols="10">comentário:</textarea>
```

Listbox e Combobox

Cria uma lista de opções para serem selecionadas. O combobox mostra apenas um valor de cada vez em um menu tipo "cortina", enquanto o listbox mostra mais de um valor, em forma de "lista". Ambos utilizam a mesma tag (<select>), porém, o listbox recebe o atributo *size,* informando quantos valores serão mostrados por ele. Ambos recebem o atributo *multiple,* que permite a seleção de múltiplos valores.

```
<!-- exemplo de listbox -->
<select name="cidade" size="5" multiple>
  <option value="guaira">Guaira</option>
   <option value="umuarama">Umuarama</option>
   <option value="lovat">Lovat</option>
   <option value="cascavel">Cascavel</option>
   <option value="cianorte">Cianorte</option>
</select>

<!-- mesmo exemplo, agora como combobox -->
<select name="cidade">
   <option value="guaira">Guaira</option>
   <option value="umuarama">Umuarama</option>
   <option value="lovat">Lovat</option>
   <option value="cascavel">Cascavel</option>
   <option value="cianorte">Cianorte</option>
</select>
```

Checkbox

Cria caixas de seleção, possibilitando ao visitante selecionar mais de uma opção. O atributo *checked* faz com que a opção apareça marcada no formulário. O atributo *value* é obrigatório.

```
Selecione uma cidade:<br>
<p align="center">
    <input type="checkbox" name="pesquisa[]" value="umuarama">Umuarama<br>
    <input type="checkbox" name="pesquisa[]" value="guaira" checked>Guaira<br>
    <input type="checkbox" name="pesquisa[]" value="curitiba">Curitiba<br>
</p>
```

Obs: Note que o atributo *name* possui em seu valor, o nome *pesquisa,* seguido de colchetes. Isso será discutido mais adiante...

Botão radio

Semelhante ao Checkbox, porém, permite a seleção de apenas uma opção dentre as oferecidas. (Diferencia-se do checkbox pelo formato arredondado).

```
Enquete:<br>
Que nota você daria ao nosso formulário?<br>
<p align="center">
 <input type="radio" name="nota" value="5">Otimo
 <input type="radio" name="nota" value="4">Bom
 <input type="radio" name="nota" value="3">Regular
 <input type="radio" name="nota" value="2">Ruim
 <input type="radio" name="nota" value="1">Péssimo
```

```
</p>
```

Obs: Note que o atributo *name* de cada botão radio possui o mesmo valor. Isso será discutido mais adiante...

Botões Submit e Reset

O botão submit envia as informações do formulário para o script indicado, utilizando o método indicado (GET ou POST). O botão reset apaga tudo o que foi digitado no formulário. É indicado colocar um botão reset no formulário, caso o usuário desista de enviar o formulário ou não queira manter suas informações no computador (Ex: quando estiver usando um computador público).

O atributo *name* é opcional, mas pode ser usado, caso haja necessidade de se ter mais de um botão *submit* ou *reset* dentro de um mesmo formulário.

```
<input type="submit" name="botao" value="enviar"><!-- botão 'enviar' -->
<input type="reset" name="limpar" value="limpar"><!-- botão 'limpar' -->
```

Exercícios:

- Monte o layout do seu site. Use tabelas para organizar o conteúdo. O site deve ter um banner na parte superior, uma coluna para o conteúdo, outra coluna para o menu de navegação do usuário e um rodapé contendo seu nome;
- No espaço destinado ao conteúdo, monte seu curriculum vitae.
- Na coluna ao lado, crie uma tabela contendo 3 linhas e uma coluna. Abaixo dela, crie um formulário de enquete e peça para que o visitante dê uma nota ao seu site
- Abaixo de seu curriculum, crie outro formulário para que o visitante possa adicionar comentários ao conteúdo do site. O visitante deve informar o nome, localidade, e-mail de contato, uma nota para o conteúdo e um comentário.

PHP

O que é PHP?

PHP (Hypertext PreProcessor) é uma linguagem de programação de scripts para manipulação de páginas HTML. Criada por Rasmus Lerdorf em 1994, é amplamente utilizada na programação de web sites dinâmicos, especialmente para manipulação de banco de dados.

Características do PHP

- PHP é gratuito: Distribuído sob a licença GPL, possui seu código-fonte (código utilizado para sua criação) aberto, o que facilita a correção de eventuais erros no código, permitindo seu rápido desenvolvimento
- PHP é multiplataforma: Inicialmente foi desenvolvido para ser usado em servidores Unix/Linux (que compreendem 70% dos servidores web), ganhando uma versão para Windows e para Macintosh posteriormente. Isso faz do PHP uma linguagem capaz de ser executada independente da plataforma utilizada.
- PHP é compatível com a a maioria dos servidores web disponíveis no mercado, tendo suporte nativo para o servidor Apache (atualmente o mais utilizado no mercado).
- PHP suporta banco de dados: Possui suporte nativo ao MySQL, porém pode utilizar outros sistemas de gerenciamento de banco de dados (SGBD), dentre eles, Oracle, Sybase, mSQL, Firebird, PostgreSQL e DB2. Permite também o uso de mais de um banco de dados na mesma aplicação.
- PHP suporta uma grande variedade de protocolos, dentre eles, IMAP, POP3, SMTP, XML, LDAP, HTTP e FTP.

ASP X PHP

A linguagem ASP foi desenvolvida sob os mesmos objetivos do PHP, porém, por ter sido desenvolvida pela Microsoft, funciona apenas em plataformas Microsoft, exigindo que seu servidor tenha Windows 2003 Server instalado, por exemplo. Isso faz do ASP inviável, devido a custos de licença, podendo variar entre US$2.500,00 e US$10.000,00. Além disso, o fato do ASP ter seu código "fechado" dificulta seu desenvolvimento e a correção de bugs, tornando a linguagem suscetível a erros.

Requisitos

PHP exige apenas a instalação de um servidor web compatível. Caso queira utilizar um banco de dados, instale um que seja compatível. Recomendamos a utilização do Servidor web Apache e do banco de dados MySQL, por terem suporte nativo ao PHP.

Instalação do PHP

Ambiente Windows

A instalação em ambiente Windows é simples. O instalador irá configurar o ambiente completo para o PHP. Você só deverá informar o nome do servidor (na dúvida, deixe *localhost*, seu nome e seu e-mail para contato.

Existe um instalador para Windows que contém o servidor Apache, o PHP e o banco de dados MySQL, além de outros recursos para administração de sites. Trata-se do Xampp. O instalador pode ser baixado em www.xampp.org.

Para saber se a instalação foi concluída com sucesso, abra o editor de texto de sua preferência e adicione as seguintes linhas:

```
<?
phpinfo()
?>
```

Salve com o nome de info.php e coloque-o na pasta C:\xampp\htdocs. Abra seu navegador de internet preferido e digite na barra de endereços: http://localhost/info.php. O resultado será uma lista detalhada da configuração do PHP em seu computador. Esta lista é gerada pelo próprio php. Este script será explicado mais adiante.

Ambiente Linux

Ao contrário do que muita gente pensa, a instalação em ambiente Linux pode ser tão simples quanto a instalação em ambiente Windows, especialmente se houver conexão com a internet. Sistemas operacionais Linux são baseados em pacotes, ou seja, cada programa é um pacote ou um conjunto de pacotes que podem ser instalados, atualizados e removidos através de comandos simples ou utilitários gráficos, como o Synaptic. A instalação através da linha de comando varia em cada caso. Se você usa um sistema baseado em Debian, como Ubuntu, Kurumin ou Knoppix, você pode utilizar o comando apt da seguinte forma:

- Abra o terminal;
- Digite *su* e aperte *Enter* para poder logar-se como administrador do sistema;
- Digite a senha do administrador;
- Digite o seguinte comando:

  apt-get install apache2 php5 mysql5 mysqladmin phpmyadmin

Caso seu sistema seja baseado em Red Hat, como Fedora, Mandriva ou CentOS, substitua o apt-get do comando acima por Yum. Caso use Mandriva, substitua-o por urpmi.

```
yum install apache2 php5 mysql5 mysqladmin phpmyadmin
urpmi install apache2 php5 mysql5 mysqladmin phpmyadmin
```

Este comando fará com que o sistema instale o servidor Apache, o PHP, o banco de dados MySQL, o administrador do MySQL e o PhpMyAdmin, usado para administrar remotamente o site.

Caso prefira uma maneira mais fácil de realizar esta tarefa, basta usar o gerenciador de pacotes Synaptic. Se sua distribuição não o possui, ele pode ser instalado através do método acima. O Synaptic está disponível para qualquer distribuição Linux. Após instalado, abra-o, digite a senha do Administrador (root), marque, clicando com o mouse, os pacotes que você quer instalar. Depois clique em aplicar. Ele irá baixar os pacotes necessários e instalar em seu sistema. Para saber se a instalação foi concluída com sucesso, abra o editor de texto de sua preferência e digite o seguinte código:

```
<?
phpinfo()
?>
```

Em seguida, salve com o nome de info.php e mova-o para a pasta /var/www. Em seguida, abra o navegador de sua preferência e digite o seguinte endereço: http://localhost/info.php. O resultado será uma lista detalhada da configuração do PHP em seu computador. Esta lista é gerada pelo próprio php. Este script será explicado mais adiante.

Scripts Client-Side e Server-Side

Scripts Client-Side (Lado-Cliente) são executados no computador do próprio cliente, por exemplo, o código HTML, que é executado direto no navegador. São muito utilizados para o processamento de pequenas tarefas, tirando essa responsabilidade do servidor web, que fica livre para executar os scripts Server-Side (Lado-Servidor), que são, por exemplo, os scripts PHP. Quando o usuário clica em um botão de busca, o pedido de busca é enviado ao servidor, que encontra o script responsável pela busca e o processa. O resultado é colocado em "pacotes" e enviado de volta ao computador que requisitou a pesquisa. Um servidor pode ter muitos clientes realizando requisições desse tipo, portanto, é recomendado que pequenas verificações e tarefas menores sejam executadas por aplicações Client-Side, deixando as requisições mais importantes (como conferir senhas por exemplo) para o Servidor. PHP é uma linguagem Server-Side, ou seja, sempre será o servidor que irá executar. Se sua página contém muitos scripts PHP desnecessários, pode acabar prejudicando o servidor, tornando a conexão mais lenta. Use o PHP com cuidado ;-)

Sintaxe PHP

A linguagem PHP pode ser usada de duas formas: incorporado ao HTML ou de forma "pura", em scripts separados das páginas HTML. Em ambos sua sintaxe é a mesma. Todo código PHP é delimitado por tags de abertura e fechamento (como o html), como mostrado abaixo:

Estilo php 'longo'	Estilo php 'curto'	Estilo Javascript	Estilo ASP
<?php ... ?>	<? ... ?>	<script language= "PHP"> ... </script>	<% ... %>

Tudo que estiver dentro dessas tag's PHP será considerado script PHP, e não HTML. Código PHP digitado fora das tag's será interpretado como erro.

Obs: Cada linha de código PHP deve terminar com ";" para que a linguagem reconheça o fim de uma linha de comando e o começo de uma nova.

Comentários

Comentários no PHP podem ser de duas formas:

Comentários de uma linha

Usamos duas barras ou asterisco para iniciar um comentário de uma linha. Tudo o que for digitado após estes caracteres (incluindo código PHP) não será interpretado pelo servidor, lembrando que comentários do código serão visíveis apenas dentro do código, ou seja, o navegador não imprime um comentário na tela.

```
<?php
//este é um comentário de uma linha
#este também é um comentário de uma linha
?>
```

Comentários de várias linhas

Usados quando o comentário ocupa mais de uma linha, são delimitados pelos sinais '/*' e '*/'. Tudo que estiver entre estes dois sinais será considerado comentário e não será interpretado pelo servidor.

```
<?php
/*Este é um exemplo de comentárioque ocupa mais de uma linha. Aqui
eu não preciso colocar o delimitador em todas as linhas.
O comentário se encerra aqui */
?>
```

Variáveis em PHP

Variáveis são "recipientes", com a função de armazenar dados a fim de serem utilizados a qualquer momento pelo programa. Diferente de muitas linguagens de programação, não é necessário declarar a existência de uma variável, ou mesmo o tipo de informação que ela armazenará. Variáveis no PHP são identificadas pelo sinal "$" antes de seu nome. Os nomes de variáveis não podem possuir caracteres especiais (espaços, acentos, etc), com exceção do underline ("_"). De preferência, use nomes que indiquem o tipo de informação que a variável irá armazenar. Exemplo de nomes válidos:

```
$nome, $endereco, $idade, $data_inscricao, $20assustar, $123;
```

PHP é Case Sensitive, ou seja, diferencia maiúsculas e minúsculas. A variável $a é diferente da variável $A.

Tipos de variáveis

Variáveis devem armazenar um determinado tipo de informação. PHP não exige que este tipo seja informado, pois reconhece-o automaticamente. Os tipos são:

- Números inteiros (int ou integer): Números naturais positivos e negativos. Ex: 25, -7;

- Números Fracionários (float, double ou real): Números fracionários, também chamados de ponto flutuante ou dupla precisão. Ex: 3,14;
- Caracteres alfanuméricos (String): Valores literais. Ex: "João da Silva". Valores do tipo String devem estar dentro de aspas ou apóstrofos. Mesmo valores numéricos, se informados como strings, serão tratados como tal;
- Valores booleanos (bool): São do tipo Verdadeiro ou Falso. PHP interpreta Verdadeiro como 1 e Falso como 0. Valores booleanos serão vistos mais adiante;
- Vetores (array): Vetores são um conjunto ordenado de variáveis. Cada variável possui um índice, que pode ser numérico ou literal (no caso de array associativo). Ex: $lista[]. Os colchetes servem para que determinada posição do vetor seja referenciada. Ex: $lista[2]. Vetores no PHP iniciam em 0.
- Objeto (objetc): Utilizados na programação orientada a objetos, onde definimos uma classe, e dentro dela as variáveis e funções que estarão disponíveis aos seus objetos. Uma variável do tipo objeto pode conter uma classe com diversas variáveis e funções para serem utilizadas. Este conceito será visto mais adiante;

Atribuindo valor à uma variável

Como vimos anteriormente, variáveis podem guardar qualquer tipo de valor, mas para isso é preciso que indiquemos o valor que deve ser guardado dentro delas. Para isso, utilizamos um operador responsável por inserir na variável um determinado valor. Trata-se do sinal "=". Ex:

```
$salario = 700,00;
```

Na linha acima, a variável salário recebe o valor 700,00 que é do tipo float (fracionário). Cada linha de código do PHP (como em muitas linguagens de programação) termina com o sinal ";". Ele indica o fim de uma linha de código.

Você pode forçar a variável a receber um tipo específico de valor, indicando o tipo desejado entre parênteses, antes da atribuição. Ex: Se quisermos que a variável $salario receba o valor como uma String em vez de um inteiro, podemos realizar a conversão do valor para String da seguinte forma:

```
$salario = (string) 700,00;
```

Dessa forma, o conteúdo da variável $salario não será um valor numérico e sim um valor literal, ou seja, não há mais o valor 700,00 e sim a string "700,00".

Para atribuir a uma variável um valor do tipo String, deve-se colocar este valor entre aspas ou apóstrofo (aspas simples). Valores dentro destes sinais são considerados Strings (as diferenças entre elas serão comentadas adiante). Ex:

```
$nome = "SENAC";
$nome = 'senac';
```

Exibindo o conteúdo na tela

Podemos exibir o conteúdo de uma variável na tela com o uso do comando echo. Ele imprime na tela o conteúdo de uma variável. Também pode ser usado para imprimir código HTML. Sua sintaxe é:

```
echo "conteudo $variavel <tag>";
```

No exemplo acima, imprimimos um valor literal, o valor de uma variável e uma tag html. Também podemos utilizar apóstrofos no lugar de aspas, porém, apóstrofos mostram o nome de uma variável em vez de seu conteúdo. Ex:

```
$variavel = "qualquer";
echo 'Valor $variavel';
```

O exemplo acima imprimirá na tela: *Valor: $variavel* em vez do valor contido na variável. Para contornar este problema, podemos utilizar o operador de concatenação Ponto (“.”). Sua função é concatenar (unir) as expressões às variáveis. Dessa forma, o exemplo acima ficaria:

```
echo 'Valor '.$variavel;
```

Dessa forma, $variavel fica unida ao comando echo, que imprime seu valor na tela. Com o uso da concatenação, a variável não precisa ficar entre apóstrofo ou aspas. O resultado será:

```
Valor qualquer
```

Arrays

Array (ou Vetor) é um conjunto ordenado de variáveis. Em um array, todas as variáveis possuem um índice, que pode ser um número ou texto. O índice deve aparecer entre “[“ e “]”, logo após o identificador do array. Ex: $lista[] ou $lista[0].

No PHP, os arrrays iniciam-se em zero. Diferente de outras linguagens, você não precisa definir um número limite de posições dentro de um array, pois ele se expande dinamicamente.

Caso o vetor não seja indicado entre os colchetes, o PHP irá procurar pelo último elemento do array e atribuir o novo valor na posição seguinte.

```
$lista[]; //isso indica que a variável lista é um array
$lista[0] = 10;  //valor 10 na posição zero; a primeira posição do array
$lista[1] = 11;  //valor 11 na posição 1
$lista[] = 12;   //valor 12 é colocado logo em seguida, na posição 2
$lista[] = 13;   //valor 13 colocado na posição 3
$lista[7] = 14;  //valor 14 colocado na posição 7, deixando as anteriores vazias
$lista[] = 15;   //valor 15 colocado na posição 8 do array
```

Podemos utlizar a função array para atribuir valore a um vetor. Os valores devem estar entre parênteses e separados por vírgula. Ex:

```
$lista = array(10, 11, 12, 13, 14, 15);
```

Também é possível utilizar nomes em vez de números para referenciar um array, criando assim um array associativo. Ex:

```
$usuario['nome'] = "João";
$usuario['idade'] = 25;
$usuario['cidade'] = "Guaira";
```

```
$usuario['estado'] = "Paraná";
```

Podemos utilizar a função array para arrays associativos, indicando uma chave e um valor, separados pelo sinal "=>" (igual e maior, formando uma seta). Ex:

```
$usuario = array("nome" => "João", "idade" => 25, "cidade" => "Guaira", "estado" =>"Parana");
```

Cada posição do array possui um "nome", em vez de um valor numérico. Podemos exibir seu conteúdo dessa forma:

```
echo "O usuário ".$usuario['nome']." , de ".$usuario['idade']." anos mora em ". $usuario['cidade']." - ".$usuario['estado']."." ;
```

O resultado do exemplo acima será:

O usuário João, de 25 anos mora em Guaíra – Paraná

Podemos utilizar mais de um índice em um array, tornando-o um array multidimensional (Matriz). Uma matriz pode ser comparada a uma tabela, formada de linhas e colunas. Ex:

```
$tabela[0][0];
$tabela[0][1];
$tabela[0][2];
$tabela[1][0];
$tabela[1][1];
$tabela[1][2];
```

No exemplo acima, teríamos uma tabela com 3 linha e 3 colunas. Podemos também combinar índices associativos à matriz. Ex: Uma lista de times de futebol de cada cidade de cada estado:

```
$time['MG']['BeloHorizonte'] = "Atletico Mineiro";
$time['SP']['SaoPaulo'] = "Corinthians";
$time['RS']['PortoAlegre'] = "Gremio";
$time['PR']['Curitiba'] = "Coritiba";
```

Arrays superglobais

São arrays pertencentes ao PHP, entre eles estão o array $_GET e $_POST que armazenam o valor das variáveis enviadas pelos formulários através dos respectivos métodos GET e POST. Além deles, temos o array $GLOBALS, que armazena variáveis globais definidas no script e os arrays $_SESSION e $_COOKIE, utilizados em sessões. Todos eles serão vistos em detalhes mais adiante. A sintaxe para a utilização destes array é:

$ARRAY['variavel_ou_valor'];

Onde ARRAY é o nome do array superglobal que será utilizado e 'variavel_ou_valor' é o nome da variável ou do valor a ser extraído do array.

Operações aritméticas

Para realizar operações aritméticas no PHP utilizamos os seguintes operadores:

* para o exemplo abaixo, consideramos $a contendo o valor 4 e $b o valor 2

Operação	Operador	Exemplo	Resultado
adição	+	$a + $b	6
subtração	-	$a - $b	2
multiplicação	*	$a * $b	8
divisão	/	$a / $b	2
módulo (resto da divisão)	%	$a % $b	0

Podemos realizar uma operação aritmética e de atribuição em uma única linha. Para isso, precisamos apenas fazer com que uma variável receba o valor da operação. Ex:

```
$resultado = $a + $b * 2;
```

Lembrando que você também pode usar sinais como parênteses para indicar a ordem de procedência na operação aritmética. Ex: Qual a metade de dois mais dois? O resultado pode ser diferente, conforme a ordem de procedência dos operadores.

```
$valor = 2+2/2;
/* nesse caso, a divisão possui procedência na operação, sendo executada antes
da adição. O resultado será 3 */
$valor = (2+2)/2;
/* já neste caso, o que está entre parênteses possui procedência maior do que a
divisão, ou seja, deve ser realizado primeiro. Neste caso, o resultado será 2 */
```

Operações com Strings e Atribuição

Operador “.”

Utilizamos o operador “.” para concatenar (unir) dois valores (geralmente Strings). Ex: quando queremos unir o valor de uma variável a uma string qualquer.

```
echo “Meu nome é “.$nome.” e moro em “.$cidade.” desde ”.$data.”.”;
```

Operador “.=”

Utilizamos operadore de atribuição para adicionar a uma variável um novo conteúdo, sem que o conteúdo antigo seja perdido. Ex: Atribuímos à variável $nome o nome do usuário.

```
$nome = “Leopoldo”;
```

O conteúdo da variável é “Leopoldo”. Agora, queremos adicionar o sobrenome à variável, mas devemos fazer isso sem que o nome seja perdido. Para isso, utilizamos o operador “.=”, que tem a função de atribuir o valor à variável, sem que seu conteúdo seja perdido.

```
$nome .= “ da Silva”;
```

Agora, o conteúdo da variável $nome é "Leopoldo da Silva".

Operadores "++" e "--"

O operador "++" tem a função de incrementar em 1 o valor da variável. Utilizado em variáveis "contadoras", é equivalente à expressão "$variável = $variável + 1". Em oposição, o operador "--" decresce o valor da variável em 1. Ex: iniciamos a variável $contador como tendo o valor 1 e utilizamos os operadores "++" e "--". O resultado é:

```
$contador = 1; //variável $contador possui valor 1
$contador++;   //agora, $contador vale 2
$contador++;   //$contador passa a valer 3
$contador--;   //$contador volta a valer 2
$contador--;   //$contador vale 1
```

Operadores "+=" e "-="

Sua função é incrementar ou decrementar à variável o valor de outra variável. Equivalente à expressão "$variavel1 = $variavel1 + $variavel2" e "$variavel1 = $variavel1 – variavel2", respectivamente. Ex:

```
$valor1 = 1;  //inicia variáveis
$valor2 = 2;
$valor3 = 3;
$cont = $valor3; //$cont agora vale 3
$cont += $valor2 //$soma à $cont o valor em $valor2. $cont vale 5
$cont -= $valor1 //subtrai de $soma o valor de $valor1. $cont vale 4
```

Operadores "*=" e "/="

Semelhante aos anteriores, porém o operador "*=" multiplica o valor das variáveis, enquanto o operador "/=" divide-os. Ex:

```
$valor1 = 4;
$valor2 = 2;
$cont = valor1; //$cont vale 4
$cont *= $valor2; //equivale a $cont = $cont * $valor2. $cont vale 8
$cont /= $valor1; //equivale a $cont = $cont / $valor1. $cont vale 2
```

Operador "%="

Como podemos supor, é semelhante aos anteriores, porém, o valor acrescentado à variável é o resto da divisão. Ex:

```
$valor1 = 2;
$valor2 = 4;
$cont = $valor2; //$cont vale 4
$cont %= $valor1
/*o resto da divisão entre $cont e $valor1 (4 e 2) é acrescentado à $cont, ou
seja. $cont = $cont + ($cont % $valor1). Como o resto da divisão entre 4 e 2 é
zero, a expressão seria equivalente a $cont = $cont + 0; */
```

Operadores de comparação

São utilizados para comparar dois valores ou duas expressões. Retorna o valor 1 em caso verdadeiro e 0 em caso de falso. É através dos operadores de comparação que

construímos testes lógicos e estruturas de decisão, que permitem ao script executar determinada função se determinada condição for satisfeita. São operadores de comparação:

Operação	operador	Sintaxe
igual	==	$a == $b //retorna verdadeiro se forem iguais
diferente	!=	$a != $b //retorna verdadeiro se forem diferentes
maior/menor	< ou >	$a > $b //verdadeiro se $a maior que $b
maior ou igual/menor ou igual	>= ou <=	$a >= $b //verdadeiro se $a for maior ou igual a $b

A operações de comparação utilizando os dois últimos exemplos deve ser utilizada com cuidado. Ex: Se possuímos duas variáveis $a e $b contendo o mesmo valor (ex: o valor 10), as seguintes operações retornarão resultados diferentes:

operação	resultado
$a == $b	verdadeiro
$a != $b	falso
$a > $b	falso
$a < $b	falso
$a >= $b	verdadeiro
$a <= $b	verdadeiro

Os dois últimos exemplos retornaram o valor verdadeiro, pois a expressão avaliou se $a era maior **ou** igual e menor **ou** igual a $b.

Operadores lógicos

Comparações como as citadas no exemplo acima são possíveis graças aos operadores lógicos. Eles avaliam se uma expressão é verdadeira ou falsa e retornam o resultado, sendo valor 1 para verdadeiro e 0 para falso. Os operadores lógicos são:

Operador	Operação
and ou &&	E lógico
or ou ||	OU lógico
xor	OU “exclusivo”
!	NÃO (negação ou inversão)
&	E lógico (bit a bit)
^	XOR lógico (bit a bit)
|	OR lógico (bit a bit)
~	NÃO lógico (bit a bit)

Operações lógicas

Em todas as linguagens de programação existem as estruturas chamadas estruturas de decisão, que são blocos de comandos, escritos de forma que executem determinada função caso uma condição seja verdadeira e outra caso seja falsa. A própria estrutura avalia

a condição e se for verdadeira, executa determinada(s) função(ões); Caso seja falsa, executa função(ões) diferentes. Porém, existem casos em que mais de uma condição deve ser avaliada. Para isso, utilizamos operações lógicas. Elas avaliam duas ou mais condições e, através de operações lógicas, determinam qual será o resultado das condições avaliadas. O programa, então, irá utilizar este valor para determinar o que fazer. Existem basicamente, quatro operações lógicas chamadas: AND, OR, XOR e NOT (E, OU, OU EXCLUSIVO e NÃO). Operações lógicas permitem que o programa avalie se uma condição é verdadeira ou falsa e execute funções específicas com base nos resultados. Veremos agora as principais operações lógicas:

Operação AND

O operador AND retorna verdadeiro **apenas** quando todas as condições forem verdadeiras. Uma tabela de possíveis resultados é mostrada abaixo.

condição 1	AND	condição 2	resultado
V	AND	V	V
V	AND	F	F
F	AND	V	F
F	AND	F	F

Podemos comparar a operação lógica AND a uma multiplicação. Substituindo o Verdadeiro por 1 e Falso por 0, teremos o AND funcionando como um operador de multiplicação.

valor 1	*	valor 2	Resultado
1	*	1	1
1	*	0	0
0	*	1	0
0	*	0	0

Operação OR

A operação OR não é tão "rígida". O resultado será verdadeiro caso uma das condições seja verdadeira. Seria idêntico à uma operação de adição, porém com uma exceção: Na operação OR, a soma 1 + 1 é igual a 1. Vejamos a tabela:

condição 1	OR	condição 2	resultado
V	OR	V	V
V	OR	F	V
F	OR	V	V
F	OR	F	F

Transformando Verdadeiro em 1 e Falso em 0, teremos:

condição 1	+	condição 2	resultado
1	+	1	1
1	+	0	1
0	+	1	1
0	+	0	0

Lembrando que não há o valor 2 na operação, portanto, 1 + 1 torna-se 1.

Operação XOR

A operação XOR (OU EXCLUSIVO) tem uma particularidade: Seu resultado será verdadeiro apenas se e somente se uma das condições for verdadeira, retornando falsa caso as duas tenham o mesmo valor.

condição 1	XOR	condição 2	Resultado
V	XOR	V	F
V	XOR	F	V
F	XOR	V	V
F	XOR	F	F

Operação NOT

Trata-se da mais simples de todas: Ela apenas inverte o resultado de uma operação, ou seja, verdadeiro torna-se falso e vice-versa.

condição	NOT	resultado
V	NOT	F
F	NOT	V

Exercícios

- Escreva um script php que armazene em variáveis o nome e a idade de uma pessoa. Depois escreva na tela, em uma única mensagem, o nome e a idade da pessoa.
- Escreva um script que mostre o valor de 3 variáveis numéricas. Depois atribua à primeira variável o valor das outras duas variáveis. Logo após, acrescente 1 ao valor da segunda variável e decresça em 1 o valor da terceira variável.
- Escreva um script que armazene em variáveis o seu nome e o nome de 5 cores (em inglês). Utilizando apenas o comando echo, escreva seu nome 5 vezes na tela. Cada ocorrência do nome deve ser de uma das cores previamente armazenadas nas variáveis. Utilize a tag <font> para formatar os nomes.

Blocos de comandos

Blocos são um conjunto de linhas de código delimitadas pelos sinais "{" e "}". Seu objetivo é agrupar comandos responsáveis por determinada função. Vejamos como blocos de comando podem ser utilizados:

Estruturas de controle

Utilizamos estruturas a fim de executar determinados blocos de comandos quando necessário, possibilitando a reutilização do código. Existem 3 tipos de estruturas:

- Estruturas de Decisão: Utilizadas pelo programa para possibilitar a execução de determinado bloco de comandos caso uma condição seja satisfeita. Dessa forma, o programa executa tarefas baseado em condições pré-estabelecidas. Ex: Escrever uma mensagem na tela, caso um valor seja par e outra mensagem caso seja ímpar.
- Estruturas de Repetição: Possibilitam a execução de um determinado bloco de comandos em uma quantidade de vezes pré-determinada. Isso evita que o mesmo bloco de comandos seja reescrito diversas vezes, facilitando a compreensão do código. Estruturas de decisão podem ser combinadas à estrutura de repetição para tornar a execução mais dinâmica.
- Funções: Blocos de comandos e estruturas podem ser colocadas dentro de funções e chamadas pelo programa principal quando necessárias, tornando o programa principal mais rápido, limpo e organizado. Também facilita sua posterior manutenção. Cada função deve possuir um nome único, que segue as mesmas regras de um nome de variáveis.

Vamos agora conhecer as estruturas que podemos utilizar no PHP:

Estrutura if

A estrutura if (do inglês: "se") executa um determinado bloco, caso uma condição seja verdadeira. Sua sintaxe é:

```
<?php
if(condição){
comandos;
}
?>
```

A utilização dos delimitadores "{" e "}" é obrigatória, exceto se o bloco contiver apenas uma linha de código. Podemos colocar quantas linhas de código precisarmos dentro da estrutura. Ex:

```
<?php
if($valor1 == 2){
      echo "Estrutura if<br>";
      echo "valor da variável é 2";
}
?>
```

No exemplo acima, a estrutura executará os comandos quando o valor da variável $valor1 for igual a 2.

Existem casos em que precisamos que a estrutura execute um determinado bloco de comandos caso uma condição seja verdadeira e outro caso seja falsa. Para isso existe uma variação da estrutura if chamado de if...else ("se...então").Essa estrutura executa um determinado bloco caso a condição seja verdadeira e outro caso seja falsa. Ex:

```
<?php
if($a == $b){
      echo "Valor da variável: $valor<br>";
      echo "Variável é um número par!<br>";
}else{
      echo "Valor da variável: $valor<br>";
      echo "Variável é ímpar!<br>";
}
?>
```

A condição a ser avaliada pela estrutura if faz um teste na variável $a. Caso o resultado seja verdadeiro ($a é igual a $b), a estrutura executará o primeiro bloco de comandos. Caso seja falsa, ela irá direto para o segundo bloco de comandos, ou seja, irá "pular" a primeira parte da estrutura. Isso evita que o teste seja feito novamente, deixando o programa duas vezes mais rápido.

Estrutura while

A estrutura while é uma estrutura de repetição. Ela avalia a expressão e executa o bloco de comandos enquanto a condição for verdadeira. caso a condição retorne falsa, o while deixa de executá-la. Sua sintaxe é:

```
<?php
while(expressão){
      comandos;
}
?>
```

Devemos ter um cuidado especial a utilizar estruturas de repetição, como while. Temos que ter certeza de que há uma chance da expressão avaliada retornar o valor Falso, do contrário, irá se tornar um laço infinito, ou seja, não vai parar de executar.

Ex:

```
<?php
$cont = 1;
while($cont < 10){
      echo "Valor de \$cont: $cont<br>";
      $cont++;
}
?>
```

Na estrutura acima, definimos o valor de $cont como sendo 1. O laço while executará um pequeno bloco de comandos enquanto o valor de $cont for menor que 10. O bloco do while escreverá uma frase informando o valor de $cont, depois irá acrescentar 1 ao seu valor. Assim, o valor de $cont irá aumentar até chegar a 10, encerrando a execução do while.

Estrutura do...while

A estrutura do...while é semelhante a while, porém, enquanto a estrutura while avalia a condição antes de executar, a estrutura do...while executa o bloco antes de fazer o teste. Ex:

```
<?php
$cont = 1;
do{
      echo "Valor de \$cont: $cont<br>";
      $cont++;
}while($cont < 10)
?>
```

Estrutura for

A utilização do for é idêntica a do while, entretanto, sua sintaxe é diferenciada. Nele, definimos uma variável contadora, uma condição para o final da repetição e uma regra de modificação do contador. Sua sintaxe é:

```
<?php
for(inicialização;condição;operador){
      comandos
}
?>
```

Na inicialização, definimos e inicializamos o contador. Depois, criamos uma condição para o término da execução do for. Em seguida definimos o parâmetro de atualização da variável de controle do for. Ex:

```
<?php
for($cont=0;$cont<10;$cont++){
      echo "Valor da variável \$cont é $cont<br>";
}
?>
```

No exemplo acima, definimos dentro do for uma variável $cont e atribuímos o valor 0 a ela. Em seguida, dizemos ao for para executar o bloco de comandos enquanto o valor de $cont for menor que 10. Por último, dizemos que cada vez que a estrutura for executada, a variável $cont receberá + 1 (em virtude do operador "++"). Após a atribuição, a condição é testada novamente (como no while) até que a condição para o término do for seja satisfeita.

Estrutura foreach

Esta estrutura oferece uma maneira mais simples de percorrer os elementos de um array. Possui duas sintaxes:

```
<?php
foreach($array as $elemento){
      comandos;
}
?>
```

ou

```
<?php
foreach($array as $chave => $valor){
      comandos;
}
?>
```

No primeiro exemplo, a estrutura vai do primeiro ao último elemento do array, e a cada iteração o valor corrente do array é atribuido a variável $elemento, e o ponteiro interno do array é avançado. Dessa forma, podemos manipular os valores de $array através da variável $elemento.Ex:

```
<?php
$valor = array(1, 2, 3, 4);
foreach($valor as $v){
      echo "Valor é: $v <br>";
}
?>
```

A estrutura foreach atribui a $v o valor da primeira posição de $valor e o imprime na tela. Em seguida, vai à segunda posição de $valor, atribui seu conteúdo a $v e o imprime novamente. Por isso o comando echo mostra o conteúdo em $v, em vez do conteúdo em $valor. No caso de um array associativo, teríamos:

```
<?php
$lista = array("um"=>1, "dois"=>2, "tres"=>3, "quatro"=>4);
foreach($a as $chave => $valor){
      echo "\$a[$chave] => $valor<br>";
}
?>
```

O exemplo acima imprime a chave associativa e o valor de cada posição do array. O resultado será:

$a[um] => 1

$a[dois] => 2

$a[tres] => 3

$a[quatro] => 4

Comandos break e continue

O comando break é utilizado para parar a execução de um determinado script, bloco ou função. Ao encontrar o comando break, PHP irá direto ao final do bloco, deixando a estrutura. Ao contrário, o comando continue faz com que o PHP mantenha a execução do script. Ex:

```
<?php
for($i=0; $i < 10; $i++){
    if($i == 5){
      echo "\$i == 5! Parou de executar!";
      break;
    }
  echo "Executando: Valor de \$i => $i";
}
?>
```

O resultado será:

executando: Valor de $i => 0
executando: Valor de $i => 1
executando: Valor de $i => 2
executando: Valor de $i => 3
executando: Valor de $i => 4
$i == 5! Parou de executar!

Estrutura switch

Essa estrutura executa um determinado bloco de instruções de acordo com o valor obtido pela expressão avaliada. Sua sintaxe é:

```
<?php
switch(condicao){
      case valor1 : comando1
            break;
      case valor2 : comando2
            break;
      case valorn : comandon
            break;
      default: comando_default
            break;
}
?>
```

Se a condição retornar o valor1, executará o comando1, então não precisará conferir as demais condições, podendo deixar a estrutura. Para isso, utiliza-se o comando break entre cada condição. Caso a condição não retorne nenhum dos valores contidos na estrutura, será executada a opção *default.* Ex:

```
<?
switch($cor){
      case 'vermelho' : echo "cor vermelha<br>";
            break;
      case 'verde' : echo "cor verde<br>";
            break;
      case 'azul' : echo "cor azul<br>";
```

```
            break;
default : echo "escolha uma cor<br>";
      break;
}
?>
```

Caso o valor de $cor seja vermelho, a frase "cor vermelha" será escrita na tela, então o comando break termina a execução da estrutura, visto que a condição foi satisfeita. Em caso contrário, fará a próxima verificação. Se nenhuma das condições for satisfeita, executará a opção default, escrevendo na tela: "escolha uma cor". O uso do default não é obrigatório no switch, nem mesmo o uso de break, visto que é a última verificação a ser feita.

Funções

Funções são blocos de códigos destinados a executar uma determinada tarefa. A sintaxe de uma função é:

```
<?php
function nome_funcao(argumento1, argumento2, argumento_n){
comandos;
}
?>
```

Qualquer código PHP válido pode ser colocado dentro de uma função. Ex:

```
<?php
function mensagem(){
      echo "Escreve uma mensagem.<br>";
      echo "Este é um exemplo de função<br>";
      for($i=0;$i<5;$i++){
            echo "Usando for em uma função<br>";
            echo "valor de \$i => $i<br>";
      }
}
?>
```

Após criarmos uma função, podemos fazer com que o script a execute com uma simples chamada. Ex:

```
<?php
echo "página web em php<br>";
echo "Vamos chamar a função mensagem()<br>";
mensagem();
echo "função executada com sucesso!!!<br>";
?>
```

Quando o PHP encontra a chamada à função, interrompe a execução do script atual e inicia a execução da função chamada. Quando terminar de executá-la, retornará ao ponto onde a função foi chamada e continuará a execução do script.

Valor de retorno

Uma função pode ou não retornar um valor. Para isso é necessário usar o comando return no final da função, indicando o valor que será retornado.Ex:

```
<?php
function retorna_valor(){
  return 10;                          //função rvalor retorna o valor 10
```

```
}
$valor = 3;                              //variável é inicializada
echo "\$valor => $valor<br>";            //valor antes da funcao
$valor = retorna_valor();                //$valor recebe o retorno da função
echo "\$valor => $valor<br>";            //valor depois da funcao
?>
```

Argumentos ou parâmetros

São valores passados à função para serem manipulados pela mesma. A passagem de parâmetros não é obrigatória. Os parâmetros devem ser declarados entre parênteses, após o nome da função. Os parâmetros serão recebidos de acordo com a ordem de passagem e transformados em variáveis locais para que a função possa utilizá-los. Ex:

```
<?php
function imprime($parametro){
      echo "$parametro<br>";
}
$argumento = "argumento torna-se parâmetro da função.";
imprime($argumento);
?>
```

No exemplo acima, a função imprime() recebe como argumento a variável $argumento para que seja manipulada. Dentro da função, o valor do argumento agora torna-se a variável $parametro, que é impressa na tela através do comando echo. O argumento passado à função não precisa necessariamente estar dentro de uma variável. Ex: a chamada *imprime("frase pra imprimir")* também é válida para o exemplo acima.

Passagem de parâmetros por referência

Os exemplos anteriores mostram a passagem de parâmetros por valor, ou seja, o valor da variável é passado à função, porém, quando a função termina de executar, o valor da variável passada à função como parâmetro permanece inalterado. Para que uma função possa alterar o valor de uma variável (ou mais de uma), é necessário fazer a passagem de parâmetros por referência, ou seja, a função irá receber a própria variável como parâmetro e poderá assim alterar seu valor. Ex:

```
<?php
function soma10($num){
  $num += 10;
}
$a = 5;
soma10($a);
echo $a;
?>
```

O resultado do script acima será 5, pois houve uma passagem de parâmetro por valor para a função soma10(). Neste caso, o valor de $a foi "copiado" e enviado à função, porém, o conteúdo de $a foi mantido pelo PHP. Para que a função possa alterar o valor de $a, a variável deve ser passada por referência, ou seja, o endereço de memória da variável deve ser passado para a função (afinal, variáveis são endereços de memória que armazenam determinado valor). A passagem por referência é feita utilizando o sinal "&". Veja o exemplo acima utilizando a passagem por referência:

```
<?php
function soma10(&$num){ //O & antes de $num indica passagem por referência
  $num += 10;
}
$a = 5;
soma10($a);
echo $a;
?>
```

O resultado agora será 15.

Variáveis globais e locais

O escopo de uma variável é a porção do script em que ela pode ser utilizada. Em geral, as variáveis tem escopo global, ou seja, podem ser utilizadas em qualquer porção do script, porém, quando declaramos funções, é criado um escopo local e a função passa a utilizar apenas as variáveis declaradas dentro dela. Uma variável global não pode ser utilizada dentro de uma função sem que seja declarada. Ex:

```
<?php
$frase = "uma frase qualquer...";
function mostra_frase(){
  echo $frase;
}
mostra_frase();
?>
```

A variável $frase dentro da função mostra_frase() é considerada uma variável diferente da que foi declarada anteriormente. Para que a função possa utilizar a variável $frase declarada fora dela, devemos declará-la dentro da função como uma variável global, para que a função possa reconhecê-la. O exemplo acima ficaria assim:

```
<?php
$frase = "uma frase qualquer...";
function mostra_frase(){
  global $frase;
  echo $frase;
}
mostra_frase();
?>
```

Outra forma de acessar uma variável global no PHP é recorrendo ao array $GLOBALS. Este array é parte do próprio PHP (uma constante) e registra todas as variáveis globais declaradas no script. Podemos utilizar o array $GLOBALS da seguinte forma:

```
<?php
$frase = "uma frase qualquer...";
function mostra_frase(){
  echo $GLOBALS["frase"];
}
mostra_frase();
?>
```

O valor dentro de ["] e ["] é o nome da variável global a ser utilizada.

Processando Formulários com PHP

A linguagem PHP é muito utilizada para o tratamento de formulários. Como visto

anteriormente, formulários são utilizados para que o usuário possa enviar informações à página, como seu endereço de e-mail, ou um nome a pesquisar. As informações são enviadas pelo formulário através dos métodos GET ou POST, processadas pelo servidor e armazenadas em arquivos ou bancos de dados. Muitas vezes, esse processamento retorna um resultado (Ex: uma pesquisa) que é enviado ao usuário. A linguagem HTML não é capaz de manipular essas informações, exigindo para isso o uso de scripts, como o PHP. Para compreendermos melhor o funcionamento do PHP no tratamento de formulários, vamos construir um exemplo simples. Faremos uma página chamada info.html, onde colocaremos um formulário que enviará informações do visitante para o servidor. Faremos também um script PHP que irá receber as informações enviadas pelo formulário e fará um processamento simples, retornando uma mensagem ao usuário. O código para o formulário é mostrado abaixo:

```
<html>
 <head>
  <title>informações do usuário</title>
 </head>
<body>
  <form action="info.php" method="GET">
   <p align="center">Nome: <input type="text" name="nome" size="30"></p>
   <p align="center">E-mail: <input type="text" name="mail" size="30"></p>
   <p align="center"><input type="submit" value="enviar"></p>
  </form>
</body>
</html>
```

A página info.html contém um formulário que pede ao usuário para que digite seu nome e seu endereço de e-mail. Note que especificamos na tag <form> o atributo action como info.php. Este é o nome do script que irá processar as informações enviadas pelo formulário. Também atruibuímos à tag method o valor GET, indicando que este será o método utilizado para o envido das informações ao servidor. Se nenhum valor for atribuído a essa tag, o método GET será utilizado.

Como vimos anteriormente, o método GET envia informações ao servidor através de uma cadeia de variáveis, indicada logo após o endereço de destino. Isso impõe um limite no tipo e na quantidade de informações que podem ser enviadas ao servidor (Ex: não é possível enviar fotos via método GET), além de tornar visíveis as informações enviadas, sendo assim um método pouco seguro de envido de dados.

Array superglobal $_GET

O PHP disponibiliza o array $_GET para tratar as informações enviadas pelo método GET. As informações enviadas são mantidas dentro destes arrays e o nome dos campos é utilizado como chave associativa para que o script possa acessar os valores. No nosso exemplo, seria $_GET['nome'] e $_GET['mail'].

Vamos agora escrever o script info.php que irá tratar as informações enviadas pelo formulário. O script de info.php é mostrado a seguir:

```
<?php
//captura as informações enviadas pelo formulário
$nome = $_GET['nome'];
$mail = $_GET['mail'];
```

```
//mostra as informações na tela
echo "<p align='center'>Meu nome é $nome</p>";
echo "<p align='center'>Meu e-mail é $mail</p>";
?>
```

O script acima captura as informações enviadas pelo formulário e as armazena dentro de variáveis, para facilitar seu processamento. Em seguida, os valores das variáveis são mostrados ao usuário. Caso os campos do formulário sejam preenchidos com os valores Umuarama e umuarama@senac, a URL será:

http://localhost/info.php?nome=umuarama&mail=umuarama%40senac

Onde:

- http://localhost/: é o endereço do site;
- info.php é o script que processa as informações do formulário;
- ? indica o início das variáveis do formulário;
- nome=umuarama : variável nome contendo o valor 'umuarama' ;
- & : usado para separar uma variável da outra;
- mail=umuarama%40senac : variável mail contendo o valor umuarama%40senac. Note que o símbolo @ foi substituído pelo símbolo %40, para que possa ser transmitido através do método GET.

Array Superglobal $_POST

Semelhante ao GET, o array superglobal $_POST armazena informações enviadas através do método POST. Para utilizar o método POST no exemplo anterior, devemos fazer pequenas alterações no formulário e no script. Ex:

```
<form action="info.php" method="POST">
```

Isso fará com que o formulário envie as informações através do método POST, em vez do GET. Agora, basta alterar o script info.php para que possa capturar as informações do array $_POST:

```
<?php
$nome = $_POST['nome'];
$mail = $_POST['mail'];
...
?>
```

Formatação de dados

Função htmlspecialchars()

Os dados enviados por formulários não possuem nenhum tipo de controle ou formatação, permitindo que ocorram diversos erros e/ou situações indesejadas. Ex: Você cria um web site para seu comércio e disponibiliza um espaço para que os visitantes comentem sobre seu trabalho. Um belo dia você encontra entre os recados, um banner da concorrência. Isso aconteceu porque não foi implantado nenhum tipo de controle ou formatação das mensagens, permitindo assim que o visitante adicionasse HTML à

mensagem. Para evitar esse tipo de transtorno, utilizamos a função htmlspecialchars, que converte HTML para texto simples, exibindo-o em forma de texto comum. No caso, o banner do concorrente apareceria como <img src="*http://www.concorrente.com/banner.jpg*"> . Ex:

```
<?php
$mensagem = <a href="www.senac.pr.gov.br">SENAC</a>;
echo $mensagem;
//mostra link
$mensagem = htmlspecialchars($mensagem);
echo $mensagem;
//mostra código html
?>
```

O primeiro comando echo mostrará o link que levará à página correspondente. o segundo mostrará o código html contido na variável.

funções addslashes() e stripslashes()

Em alguns casos, o usuário pode digitar caracteres especiais em um campo onde eles não devam ser utilizados. Ex: Um usuário pode preencher o campo nome do formulário como *José da Silva, popular "Zezinho".* As aspas contidas no campo podem causar erros na gravação e/ou interpretação das informações. Para evitar isso, utilizamos a função addslashes, que adiciona uma \ antes de cada ocorrência de caracteres especiais. O campo ficaria: *José da Silva, popular \"Zezinho\".* A presença da \ faz com que as aspas sejam interpretadas como caracteres especiais, e não sejam executadas como parte do script (Ex: poderia causar problemas com o comando echo). Utilizamos addslashes da seguinte forma:

```
$nome = addslashes($nome);
```

Para reverter o efeito da função addslashes, utilizamso a função stripslashes, que remove os caracteres de controle adicionados pela função addslashes, fazendo com que a string volte à sua composição original. Ex:

```
$nome = stripslashes($nome);
```

Dessa forma, o nome volta a ser José da Silva, popular "Zezinho".

funções urlencode() e urldecode()

Ao utilizar o método GET, caracteres especiais são substituídos pelo seu código hexadecimal correspondente, precedido de % (Ex: espaços em branco são substituídos por %40). Para que os códigos sejam convertidos para os caracteres especiais que representam, usamos urldecode. Para realizar a operação inversa, utilizamos a função urlencode. Ex:

```
<?php
$site = http://localhost/?nome=Jose%40Silva
$site = urldecode($site);
echo $site;
$site = urlencode($site);
?>
```

Isso fará com que os códigos hexadecimais sejam removidos, mostrando na tela: *http://localhost/?nome=Jose Silva.* Em seguida, a função urlencode é aplicada e o link volta ao seu estado normal. Isso pode ser útil caso o script precise retransmitir o link.

funções intval() e doubleval()

Muitas vezes precisamos que o usuário forneça um valor numérico para o formulário, porém os campos aceitam apenas texto. Para a devida conversão, podemos utilizar as funções intval() e doubleval() que, respectivamente convertem o valor da variável em inteiro e em double. Ex: No formulário html temos:

```
<input type="text" name="idade">
<input type="text" name="peso">
```

O script PHP deve tratar os dois valores como inteiro e double, respectivamente. Para isso podemso ter o seguinte código:

```
<?php
$idade = intval($_GET['idade']);
$peso = doubleval($_GET['peso']);
?>
```

funções trim(), ltrim() e chop()

Freqüentemente, usuários preenchem formulários colocando espaços em branco antes e depois dos valores. Para o formulário, isso não significa muito, mas para o script (ex: no momento de gravar as informações em um banco de dados) isso pode se tornar um problema, pois espaços em branco são interpretados como valor. Para evitar problemas, podemos remover os espaços em excesso que o usuário envia utilizando uma das funções citadas acima. Apesar de possuírem o mesmo objetivo, essas funções possuem pequenas diferenças: A função chop() remove espaços apenas no final da string. A função ltrim() remove espaços no início da string e a função trim() remove espaços no início e no final da string. A utilização depende do resultado desejado. Ex:

```
<?php
$nome = trim($nome);
$mail = chop($mail);
?>
```

Validação de formulários com javascript

Até agora, nossos formulários não receberam nenhum tipo de validação, nada que impedisse o envio de informações incorretas ou "campos em branco". Criar funções PHP para fazer esse tipo de validação é perfeitamente possível, no entanto, exige muito processamento do servidor, tornando o acesso lento e o retorno demorado. Para evitar esse tipo de transtorno, faremos com que uma parte das validações necessárias seja feita no navegador do cliente, usando Javascript.

Javascript é uma linguagem de programação que funciona diretamente no navegador do cliente (por isso chamado de tecnologia cliente). Um script javascript pode ser escrito em meio ao código html dentro da tag <script> ou fora, em documentos de texto separados. Javascript só precisa de um editor de texto para ser escrito e um navegador para ser executado. Apesar dos grandes esforços para que ele seja executado de forma padrão, apenas o navegador netscape executa-o com precisão.

Não nos aprofundaremos em javascript, pois não é o foco deste documento. Conheceremos apenas o básico para a validação de formulários.

Vamos criar a função que irá validar os campos do formulário. Essa função deve ser escrita no cabeçalho do código html da página do formulário. Todo o código deve estar dentro da tag <script>, que indicará que o código é javascript dessa forma:

```
<head>
<script language="Javascript">
function valida(nomeform){
      ...
}
</script>
...
</head>
```

Na tag <form>, devemos indicar que o formulário deverá passar pela função valida antes de ser enviada para o servidor. Essa condição é definida configurando o atributo onsubmit da seguinte forma:

```
<form method="POST" action="form.php" onsubmit="return valida(this)">
```

O formulário utilizará o método POST para envio das informações. o script que irá tratar as informações do formulário é o form.php. O atributo onsubmit (ao enviar) desviará a saída do formulário para a função valida. A palavra this indica que será este formulário que será tratado pela função. A palavra return indica que a função retornará um valor (no caso, booleano). Se a função retornar falso, o formulário não será enviado.

Vamos implementar a função valida com as devidas validações a serem feitas. Vamos começar fazendo com que o formulário não seja enviado caso o campo nome não tenha sido preenchido, ou seja, seu valor é nulo:

```
if(formulario.nome.value=="")
      alert("Nome não foi preenchido");
```

Se o valor no campo nome do formulário não possuir nada, uma janela de alerta será mostrada na tela com a mensagem "Nome não foi preenchido". O formulário não será enviado.

Para descobrir se o usuário digitou o @ no campo de e-mail:

```
if(formulario.email.value.indexOf('@',0) == -1)
      alert("E-mail inválido");
```

Para limitar o número de caracteres mínimo e máximo para o campo de senha:

```
if(formulario.senha.value.lenght<5 || formulario.senha.value.lenght >15)
      alert("senha deve ter entre 5 e 15 caracteres");
```

Agora vamos unir as condições de teste dentro da função valida:

```
<script language="javascript">
function valida(formulario){
  if(formulario.nome.value==""){
      alert("Nome não foi preenchido");
      return false;
  }
  if(formulario.email.value.indexOf('@',0)== -1){
      alert("E-mail incorreto!");
      return false;
  }
  if(formulario.senha.value.lenght<5 || formulario.senha.value.lenght>15){
      alert("Senha deve conter entre 5 e 15 caracteres");
      return false;
```

```
  }
  return true;
{
```

Antes de ser enviado ao servidor, os valores do formulário são submetidos à função valida. Caso alguma irregularidade seja encontrada, a função retorna false e o formulário não é enviado ao servidor. Caso contrário, a função retorna true e o formulário é enviado.

Validação com PHP

Muitas vezes, a validação feita com javascript não é suficiente (ex: o código foi modificado pelo cliente), exigindo assim uma segunda validação através do PHP. Vamos adaptar o script form.php para fazer a validação do formulário. Vejamos as validações mais importantes a serem feitas:

Espaços em branco

Em muitos casos, devemos evitar que o usuário envie espaços em branco dentro do campo (ex: campo de e-mail ou senha). Para remover os espaços em branco, utilizamos a função strstr(), que busca a ocorrência de um caracter dentro de uma string. Ex:

```
<?php
if(strstr($senha,' ')==TRUE)
      echo "A senha não pode conter espaços em branco";
?>
```

A função strstr buscou por espaços em branco na variável $senha. Em caso de haver espaços, uma mensagem de alerta é enviada ao usuário (O código também pode ser adaptado para que mostre uma janela de alerta, usando javascript).

Quantidade mínima de caracteres

Para verificar se um número mínimo de caracteres foi digitado, usamos a função strlen(), que retorna o número de caracteres de uma string. Ex:

```
<?php
if(strlen($senha)<5)
      echo "Senha deve ter, no mínimo 5 caracteres";
?>
```

Correção automática

Em muitos casos, especialmente no campo de e-mail, alguns caracteres são digitados incorretamente, como vírgula no lugar de ponto, espaços antes e depois do @, etc. O script PHP pode corrigir automaticamente estes erros com a função str_replace(). Sua sintaxe é:

```
str_replace("caracter_antigo","caracter_novo",string).
```

Ex: para substituirmos os hífens por underlines, temos:

```
<?php
$mail = str_replace( "-" , "_" , $mail);
?>
```

Isso fará com que toda ocorrência de hífen dentro da variável $mail seja substituída por underline. Se quisermos remover um caracter (ex: espaço em branco), basta indicar que o novo caracter será 'nada'. Ex:

```
<?php
$mail = str_replace( " ", "",$mail);
?>
```

Valores numéricos

Para saber se um campo recebeu apenas valores numéricos (ex: idade, telefone, CEP), podemos utilizar a função is_numeric(). Ela retorna 'verdadeiro' se o campo for numérico. Ex:

```
<?php
if(!is_numeric($cep))
      echo "cep deve conter apenas números";
?>
```

Após as devidas validações, as informações podem ser tratadas (ex: gravadas em um banco de dados). O código completo do formulário de cadastro (cadastro.html) e seu script PHP (cadastro.php) são mostrados a seguir:

```
<html>
<head>
<title>Cadastro</title>
<script language="javascript">
function valida(form){
  if(form.nome.value==""){
      alert("Nome não foi preenchido");
      return false;
  }
  if(form.mail.value=="" || form.mail.value.indexOf('@',0) == -1 ||
form.value.indexOf('.',0) == -1){
      alert("Campo de e-mail é inválido");
      return false;
  }
  if(form.estado.selectedIndex == 0){
      alert("Selecione um estado");
      return false;
  }
  if(form.login.value.lenght<5 || form.login.value.lenght>15){
      alert ("Login deve conter entre 5 e 15 caracteres");
      return false;
  }
  if(form.senha.value.lenght<5 || form.senha.value.lenght>15){
      alert("Senha deve conter entre 5 e 15 caracteres");
      return false;
  }
  if(form.senha.value.indexOf(' ',0)!= -1){
      alert("Senha não deve conter espaços em branco");
      return false;
  }
  if(form.senha.value != form.confirmasenha.value){
      alert("senhas não conferem");
      return false;
  }
  return true;
}
</script>
</head>
<body>
<form method="POST" action="cadastro.php" onsubmit="return valida(this)">
```

```
  <p>Nome: <input type=”text” name=”nome” size=”20”></p>
  <p>E-mail: <input type=”text” name=”email” size=”20”></p>
  <p>Estado:
  <select name=”estado”>
   <option value=”AC”>AC</option>
   <option value=”AL”>AL</option>
   <option value=”AM”>AM</option>
   <option value=”AP”>AP</option>
   <option value=”BA”>BA</option>
   <option value=”CE”>CE</option>
   <option value=”DF”>DF</option>
   <option value=”ES”>ES</option>
   <option value=”GO”>GO</option>
   <option value=”MA”>MA</option>
   <option value=”MG”>MG</option>
   <option value=”MS”>MS</option>
   <option value=”MT”>MT</option>
   <option value=”PA”>PA</option>
   <option value=”PB”>PB</option>
   <option value=”PE”>PE</option>
   <option value=”PI”>PI</option>
   <option value=”PR”>PR</option>
   <option value=”RJ”>RJ</option>
   <option value=”RN”>RN</option>
   <option value=”RO”>RO</option>
   <option value=”RR”>RR</option>
   <option value=”RS”>RS</option>
   <option value=”SC”>SC</option>
   <option value=”SE”>SE</option>
   <option value=”SP”>SP</option>
   <option value=”TO”>TO</option>
  </select></p>
  <p>Login: <input type=”text” name=”login” size=”20”></p>
  <p>Senha: <input type=”password” name=”senha1” size=”20”></p>
  <p>Confirma Senha: <input type=”password” name=”senha2” size=”20”></p>
  <p><input type=”submit” value=”enviar” name=”enviar”></p>
</form>
</body>
</html>
```

Quando o usuário clicar no botão enviar, as informações do formulário serão enviadas para a função valida(), que verifica o preenchimento correto dos campos e retorna um valor booleano. Caso seja encontrada uma irregularidade, a função retorna false (falso) e os dados não serão enviados ao servidor. Caso todas as informações estejam corretas, a função retorna true (verdadeiro) e o formulário é enviado para o servidor.

Vamos agora ao script cadastro.php:

```
<?php
//captura as informações do formulário através do array POST
$nome = $_POST['nome'];
$email = $_POST['email'];
$estado = $_POST['estado'];
$login = $_POST['login'];
$senha1 = $_POST['senha1'];
$senha2 = $_POST['senha2'];
//elimina erros na digitação de e-mails
$email = str_replace(“ ”,“”,$email);
```

```
$email = str_replace("/","",$email);
$email = str_replace("@.","@",$email);
$email = str_replace(".@","@",$email);
$email = str_replace(",",".",$email);
$email = str_replace(";",".",$email);
//variável que informará a ocorrência de erros
$erro = 0;
//verifica se foi digitado o nome
if(empty(nome)){
 $erro = 1;
 $msg = "Informe seu nome";
}
//verifica tamanho mínimo do e-mail e se existe "@" e ponto.
elseif(strlen($email)<8 || substr_count($email, "@")!=1 ||
substr_count($email,".")==0){
 $erro = 1;
 $msg = "E-mail  não foi digitado corretamente";
}
//verifica se o estado foi selecionado
elseif(strlen($estado)!=2){
 $erro = 1;
 $msg = "Informe o estado";
}
//verifica o tamanho do login
elseif(strlen(login)<5 || strlen(login)>15){
 $erro = 1;
 $msg = "Login deve ter entre 5 e 15 caracteres";
}
//verifica se a senha contém espaços em branco
elseif(strstr($senha,' ')!=false){
 $erro = 1;
 $msg = "A senha não deve conter espaços em branco";
}
//compara senha1 e senha2
elseif($senha1 != $senha2){
 $erro = 1;
 $msg = "Senhas digitadas não conferem";
}

//se há erro, exibe mensagem:
if($erro){
 echo "<html><body>";
 echo "<p align='center'>$msg</p>";
 //cria um link 'voltar' usando javascript
 echo "<p align='center'><a href='javascript:history.back()'>Voltar</a></p>";
 echo "</body></html>";
}else{
 //aqui podemos realizar o tratamento das informações. Ex: gravando em arquivo
 //ou banco de dados
 echo "<html><body>";
 echo "<p align='center'>Cadastro Realizado com sucesso!</p>";
 echo "</body></html>";
}
?>
```

O script acima verifica o preenchimento do formulário. Caso algum erro seja encontrado, a variável *$erro* recebe o valor 1, e a variável *$msg* recebe uma mensagem relacionada à irregularidade encontrada, que é exibida junto com um link, que utiliza uma função Javascript para levar o usuário à última página visitada (no caso, o formulário). Caso

o preenchimento esteja correto, o script apenas escreve a mensagem "Cadastro realizado com sucesso" e finaliza. Até então, este script tem apenas a função de validação do formulário, mas podemos acrescentar funções para o tratamento de formulários, como a gravação em arquivos de texto ou bancos de dados.

Arquivos de texto

A solução mais simples para o tratamento de informações é gravá-las em arquivos de texto. PHP disponibiliza funções para a criação e manipulação de arquivos de texto. A utilização de arquivos de texto é recomendada quando o volume de informações é pequeno e não exige um tratamento mais complexo. Ex: podemos utilizar arquivos de texto para criar contadores de acesso para o site em geral ou para páginas específicas, criação de livros de visitas, listas de discussão, anúncios on-line ou qualquer outra aplicação simples, que não exija um tratamento mais refinado, como oferecido por sistemas de gerenciamento de banco de dados (SGBD), como o MySQL por exemplo, que deixaria o acesso ao site mais lento. Entretanto, não é aconselhável utilizar arquivos de texto para tratar de informações sigilosas, como senhas de usuários, devido a limitações de segurança (arquivos texto podem ser lidos por qualquer pessoa). Nesses casos aconselhamos o uso de bancos de dados.

Manipulando Arquivos

As quatro operações básicas que podemos realizar em um arquivos são abertura, leitura, escrita e fechamento. PHP oferece diversas funções para o tratamento de arquivos. Vejamos as principais:

Abertura

Utilizamos a função fopen() para realizar a abertura de um arquivo. Sua sintaxe é:

```
fopen(nome_arquivo, modo);
```

O parâmetro *nome_arquivo* é o nome do arquivo que será criado. Se for informado uma URL iniciando em *http://,* será criada uma conexão http com o servidor informado antes da abertura do arquivo. Caso seja informado uma URL iniciada em *ftp://,* uma conexão ftp será criada antes da abertura do arquivo. Isso possibilita a manipulação de arquivos em servidores remotos. Caso a função seja executada com sucesso, ela retornará um ponteiro para o arquivo recém aberto, para que possamos manipular o arquivo em si.

O segundo parâmetro da função é o modo de abertura do arquivo. Uma relação dos modos de abertura é mostrada a seguir:

Modo	Descrição
'r'	Somente leitura, posiciona ponteiro no início do arquivo
'r+'	Leitura e escrita, posiciona ponteiro no início do arquivo
'w'	Somente escrita, posiciona ponteiro no início do arquivo e deixa-o com tamanho zero. Se o arquivo não existir, tenta criá-lo
'w+'	Leitura e escrita, posiciona ponteiro no início do arquivo e deixa-o com tamanho zero. Se o arquivo não existir, tenta criá-lo
'a'	Somente escrita, posiciona ponteiro no final do arquivo. Se o arquivo não existir, tenta criá-lo
'a+'	Leitura e escrita, posiciona ponteiro no final do arquivo. Se o arquivo não existir, tenta criá-lo
'x'	Cria e abre arquivo somente para escrita, posiciona ponteiro no início do arquivo. Se arquivo existir, retorna falso e gera um erro. Usado apenas em arquivos locais
'x+'	Cria e abre arquivo para leitura e escrita, posiciona ponteiro no início do arquivo. Se arquivo existir, retorna falso e gera um erro. Usado apenas em arquivos locais

Existem também os parâmetros 't' e 'b', informados como último caracter do parâmetro modo. O parâmetro 't' é usada em sistemas Windows, para "traduzir" as quebras de linha (\n) para \r\n. O parâmetro 'b' força o uso do modo binário, usado em sistemas que diferenciam o modo binário do modo texto. Ex:

```
$ponteiro = fopen("/pasta/teste.txt", "r");
$ponteiro = fopen("/var/www/texto.txt", "wb");
$ponteiro = fopen("http://www.server.com", "r+");
```

Em caso de problemas na abertura de arquivos em sistemas Unix (Linux, BSD, etc) verifique as permissões do diretório onde o arquivo está sendo criado. Em sistemas Windows, deve-se usar caracteres de escape para a utilização das barras invertidas. Ex:

```
$ponteiro = fopen("c:\\pasta\\arquivo.txt", "rt");
```

Fechamento

Para fechar um arquivo, devemos usar a função fclose() da seguinte forma:

```
fclose(ponteiro);
```

O parâmetro *ponteiro* é a variável que armazena o ponteiro do arquivo criado com a função fopen. Caso o arquivo não possa ser fechado, a função retornará o valor falso. Ex:

```
<?php
$ponteiro = fopen("testes.txt", "r");
fclose($ponteiro);
$arquivo = fopen("contador.txt", 'r+');
fclose($arquivo);
?>
```

Leitura

A leitura de um arquivo pode ser feita utilizando-se a função fread(), que possui a seguinte sintaxe:

```
fread(arquivo,tamanho);
```

onde arquivo é o nome do arquivo que deve ser lido e tamanho é o tamanho em bytes que

deve ser lido. Ex:

```
<?php
$arq = fopen('texto.txt', 'r');
$conteudo = fread($arq,4096);
echo $conteudo;
fclose($arq);
?>
```

No exemplo acima, o arquivo texto.txt é aberto no modo leitura (r). Em seguida, a função fread faz a leitura de 4096 bytes do arquivo e armazena na variável $conteudo, que é exibida na tela com o comando echo. Depois, o arquivo é fechado.

Se o parâmetro 'tamanho' não for informado para a função fread(), ela utilizará o valor padrão de 1024 bytes. A leitura será encerrada quando o número de bytes informado for lido ou quando o final do arquivo for alcançado.

Outra alternativa para a leitura de arquivos de texto é o uso da função file_get_contents(). Essa função dispensa o uso de fopen(), fread() e fclose(), já que faz a abertura, leitura e fechamento do arquivo de uma única vez. Ex:

```
<?php
$texto = file_get_contents("comentario.txt");
echo $texto;
?>
```

O arquivo 'comentario.txt' foi aberto pela função, que armazenou seu conteúdo na variável $texto e logo depois o fechou.

Outra alternativa seria o uso da função readfile(). Ela abre o arquivo, exibe seu conteúdo e o fecha, dispensando o uso de qualquer outra função extra. Ex:

```
<?php
$texto = readfile("comentario.txt");
?>
```

Esse simples exemplo abre o arquivo "comentario.txt", exibe seu conteúdo e o fecha.

Para ler apenas uma linha do arquivo, usamos a função fgets(). Essa função também recebe um tamanho como parâmetro, porém ela irá ler o arquivo até que o tamanho informado seja alcançado ou até alcançar o final da linha. Ex:

```
<?php
$arq = fopen("comentario.txt", "r");
$linha = fgets($arq,2048);
echo $linha;
fclose($arq);
?>
```

No exemplo acima, o arquivo "comentario.txt" é aberto em modo leitura e sua primeira linha é lida com a função fgets() e armazenada na variável $linha. Se a função fosse chamada novamente, retornaria a segunda linha do arquivo.

Para ler apenas um caracter por vez, usa-se a função fgetc(). Ex:

```
<?php
$arq = fopen("comentario.txt","r");
while(($char = fgetc($arq))!= false)
      echo $char;
fclose($arq);
?>
```

No exemplo acima, uma atribuição é feita dentro do while, onde a função fgetc lê um caracter do arquivo e armazena na variável $char, que é exibida com echo. Em seguida o próximo caracter é lido e assim até que não haja mais caracteres a serem lidos (retorna false). Então, o arquivo é fechado.

Existem casos em que o arquivo possui tag's html e php em seu conteúdo e elas devem ser removidas para que o arquivo possa ser exibido ao usuário. Para eliminar essas tag's, usamos a função fgetss(), que possui a mesma sintaxe de fgets(), com a diferença de eliminar tags do conteúdo. Ex: O arquivo "texto.txt" possui o seguinte conteúdo:

```
exemplo de <b>negrito</b> e </i> itálico</i>
```

Seria visualizado como:

exemplo de **negrito** e *itálico*

Vamos realizar a leitura do arquivo usando a função fgetss(), no exemplo abaixo:

```
<?php
$arq = fopen("texto.txt", "r");
$conteudo = fgetss($arq, 2048);
echo $conteudo;
fclose($arq);
?>
```

O resultado seria:

```
exemplo de negrito e itálico
```

Podemos também atribuir cada linha de um arquivo a uma posição de um array, para que seu conteúdo seja exibido através de um laço for ou foreach. Para isso, utilizamos a função file(), que armazena cada linha do arquivo em uma posição do vetor. Ex:

```
<?php
$vetor = file("comentario.txt");
foreach( $vetor as $v)
      echo "$v<br>";
?>
```

O exemplo acima coloca cada linha do arquivo "comentario.txt" em uma posição do vetor. Em seguida cada posição é exibida com o uso do foreach.

Escrita

Para escrever dados em um arquivo, utilizamos a função fwrite. Seus parâmetros são o ponteiro do arquivo que deve receber os dados e a string que será gravada no arquivo:

fwrite(ponteiro,string);

a string informada será gravada no arquivo especificado pelo seu ponteiro. Ex:

<?php

```
$conteudo = "Um pequeno exemplo de conteúdo que pode ser gravado em um arquivo";
$arq = fopen("exemplo.txt","w");
fwrite($arq,$conteudo);
fclose($arq);
?>
```

No exemplo acima, a string contida na variável $conteúdo foi gravada no arquivo "exemplo.txt".

Como foi visto anteriormente, podemos utilizar arquivos de texto para guardar pequenas informações e criar pequenas aplicações, como contadores de acesso, livros de visitas, mural de recados, entre outras. Entretanto, algumas aplicações exigem um gerenciamento mais completo das informações, além de uma maior segurança e organização, como um cadastro de clientes.

Funções Require e Include

Escrever scripts em php pode ser mais simples do que nas demais linguagens CGI, porém, não é uma boa prática colocar todo script dentro de um único documento. O ideal é que se tenha documentos separados e que estes sejam chamados quando necessários (assim como as funções). Para que um script contido em um arquivo separado possa ser usado, devemos chamá lo utilizando as funções require() e include(). Essas funções agem como chamadas à funções, carregando o script php que recebem como parâmetro e executando-o dentro do script que o chamou. Ex: Temos um script chamado data.php, cuja função é imprimir na tela a data atual. Seu conteúdo é:

```
<?php
$data = date("d-m-Y");
echo " Hoje é $data <br>";
?>
```

Se quisermos que a data seja exibida em todas as páginas de nosso site, não precisaremos escrever este script novamente. Basta que o "chamemos", através da função include:

```
<?php
echo "Página Principal<br>";
include("data.php");
?>
```

A função include incluirá o script data.php na execução, desviando o processamento para o script e retornando em seguida.

Em alguns casos, precisamos chamar o script apenas uma vez, então usamos a função inlcude_once(), que realiza a chamada apenas uma vez. As funções require() e require_once() são idênticas, porém, é aconselhável utilizar include em vez de require(), por questões de desempenho e confiabilidade.

MySQL e PHP

Como dito anteriormente, informações que precisam ser manipuladas com mais segurança e flexibilidade exigem o uso de um banco de dados. Esse banco de dados exige um sistema de gerenciamento (SGBD – Sistema de Gerenciamento de Banco de Dados). Existem muitos SGBD's disponíveis no mercado, de forma gratuita ou através de pagamento de licenças. Entre eles, está o MySQL, um sistema desenvolvido pela Sun Microsystems que roda nativamente no PHP.

A linguagem PHP suporta diversos SGBD's como Oracle, Sybase, Interbase(Firebird), mSQL, Microsoft SQL Server, MSSQL, MySQL, PostgreSQL, entre outros. Entretanto, o MySQL, como SGBD nativo do PHP, não exige instalação de nenhum recurso adicional (driver, extensão, etc), bastando apenas que o sistema esteja funcionando no servidor.

Estrutura de um Banco de Dados

Um banco de dados é constituído de tabelas, também chamadas de entidade, que são constituídas de colunas, onde cada coluna representa um atributo da entidade. Ex: Em um banco de dados de uma loja virtual, podemos encontrar as tabelas cliente, produto, fornecedor e pedido. Na tabela cliente, podemos encontrar atributos, como Nome, endereço, número do CPF, etc. As tabelas representam entidades existentes no mundo real e são constituídas de colunas que representam atributos relevantes de cada entidade.

Uma vez definidas as tabelas, podemos "preenchê-las" com as informações necessárias, ex: podemos cadastrar clientes enviando à tabela cliente as informações necessárias, conforme pede a tabela cliente (nome, endereço, cpf,...).

Acessando MySQL

Podemos acessar o MySQL através do terminal utilizando o comando mysql. Sua sintaxe é:

```
prompt> mysql -u root -h localhost -p
```

A opção -u indica o usuário (no caso, root ou superusuário). A opção -h indica o servidor onde se encontra o MySQL (nesse exemplo o servidor local ou localhost) e a opção -p indica que o mysql deve exigir a senha de acesso para o usuário. Se o usuário informado possuir uma senha de acesso, a mesma deve ser informada e então o sistema fará o login no MySQL. A tela do terminal terá agora a seguinte aparência:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.0.32-Debian_7etch5-log Debian etch distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Isso indica que o login foi efetuado e que você se encontra agora dentro do MySQL.

Comandos sql

A linguagem sql (Struct Query Language) é uma linguagem padrão para a manipulação de dados dentro de um SGBD. Desenvolvida pela IBM, possibilita o armazenamento, organização, atualização e exclusão de informações dentro do banco de dados.

A linguagem sql é dividida em duas: Linguagem de Definição de Dados (DDL) e Linguagem de Manipulação de Dados (DML). As DDL's são utilizadas para "montar" o banco de dados e suas tabelas, enquanto as DML's são utilizadas para manipular os dados contidos no banco.

Comando create

Este comando é utilizado para criar novos bancos de dados ou tabelas. Sua sintaxe é:

```
create database nome_do_banco ; //cria um novo banco de dados
create table nome_da_tabela(atributos); //cria uma nova tabela
```

Ex: Vamos criar um novo banco de dados:

```
mysql> create database exemplo;
Query OK, 1 row affected (0.00 sec)
```

A linha abaixo do comando indica que a Query (comando SQL) executou com sucesso, afetando uma linha, levando 0,00 segundos para ser executado.

Agora vamos informar ao MySQL que pretendemos utilizar este banco de dados Isso é feito através do comando use:

```
mysql> use exemplo;
Database changed
```

A mensagem abaixo indica que o MySQL passará a utilizar o banco de dados 'exemplo' para executar os próximos comandos.

Se quisermos ver os bancos de dados existentes no servidor, podemos utilizar o comando show da seguinte forma:

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| exemplo            |
| mysql              |
| testes             |
+--------------------+
4 rows in set (0.00 sec)
```

O MySQL mostrará uma pequena tabela com a relação dos bancos de dados existentes no servidor.

Para uma melhor demonstração, vamos criar uma tabela chamada Aluno. Essa tabela irá armazenar os dados dos alunos de um curso on-line. Os atributos dessa tabela serão: Nome do aluno, E-mail, data de nascimento, e sua matrícula. Cada um dos atributos descrito será uma coluna da tabela Aluno. Assim como as variáveis, atributos de uma tabela também possuem um tipo de dados que poderá armazenar. Os tipos de cada atributo devem ser informados no momento da criação da tabela, mas podem ser alterados posteriormente (como veremos mais adiante). Os tipos de dados mais importantes são descritos na tabela abaixo:

atributo	**tipo**
char	caracteres (string) de tamanho fixo
varchar	caracteres (string) de tamanho variável
int	números inteiros
float, double	números fracionários (ponto flutuante)
date	data no formato do mysql (ano-mes-dia)
not null	indica que o atributo não pode ser nulo (deve possuir um valor)
auto_increment	indica que valor do campo será incrementado automaticamente

primary key	indica que o atributo é uma chave primária (identificador único)
foreign key	indica que o atributo é uma chave estrangeira (chave primária de outra tabela)
text	armazena texto (não deve ser do tipo not null)

Atributos definidos como varchar podem ocupar menos espaço físico no servidor, visto que seu tamanho pode variar de acordo com o valor informado. Ex: um campo varchar de 30 caracteres pode ter o tamanho máximo de 30 caracteres, mas se for informado um valor menor, o campo se adaptará ao valor indicado, desde que não exceda o valor máximo indicado. Ao contrário, atributos do tipo char tem o tamanho fixo. O uso do atributo varchar torna a tabela mais lenta em suas operações.

Campos definidos como not null obrigatoriamente devem receber um valor. Em geral, é definido um valor padrão através do parâmetro 'default', que define um valor padrão para o campo.

Chave primária e chave estrangeira

As chaves primárias (primary key) são utilizadas para identificar um registro dentro de uma tabela. Ex: A matrícula de cada aluno pode ser utilizada para identificá-lo dentro da classe. Colunas definidas como primary key devem obrigatoriamente ser do tipo not null e não podem ter seu valor repetido, afinal, trata-se de um identificador único para cada elemento cadastrado. Preferencialmente utilizam-se campos numéricos para chaves primárias (ex: id, matrícula) ou outro atributo que respeite os argumentos citados.

As chaves estrangeiras são usadas quando queremos relacionar tabelas entre si. Ex: Cada aluno cadastrado será colocado em uma determinada turma. Podemos ter uma tabela chamada turma, com informações das turmas existentes e uma chave primária para identificar cada turma, assim como na tabela aluno. Então, só precisamos criar uma terceira tabela (ex: matriculado), contendo apenas a chave primária que identifica o aluno e a turma em que será matriculado. Para a tabela matriculado, esses atributos são chaves estrangeiras, pois não são identificadores da própria tabela, e sim identificadores pertencentes à outras tabelas (daí o nome de chaves estrangeiras). Essa técnica faz com que as pesquisas tornem-se mais eficientes, agiliza o funcionamento do banco de dados, organiza melhor as informações e possibilita melhor gerenciamento das informações cadastradas. Veremos o uso de chaves estrangeiras mais adiante.

Criando tabelas

Para a tabela aluno de nosso exemplo, teremos os atributos matricula, nome, email e data de nascimento. Os atributos nome e e-mail serão do tipo varchar de 40 caracteres. Data de nascimento será do tipo date e a matrícula será uma chave primária do tipo inteiro. O uso de chaves primárias não é obrigatório no MySQL, mas é muito útil para aplicações, como o exemplo anterior. Para montar essa estrutura, o comando create ficará dessa forma:

```
mysql> create table aluno(
    -> matricula int not null auto_increment primary key,
    -> nome varchar(40) not null,
    -> email varchar(40),
```

```
    -> dataNasc date
    -> );
Query OK, 0 rows affected (0.02 sec)
```

Note que cada linha termina com uma vírgula, exceto a última linha que antecede o sinal ");". Ele indica o fim dos parâmetros do comando create. Tudo o que estiver entre estes parênteses fará parte da estrutura da tabela aluno.

O comando show pode mostrar as tabelas dentro do banco de dados:

```
mysql> show tables;
+------------------+
| Tables_in_exemplo |
+------------------+
| aluno            |
+------------------+
1 row in set (0.00 sec)
```

Para averiguar a estrutura da tabela, podemos utilizar o comando desc, que fornecerá uma descrição da tabela:

```
mysql> desc aluno;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| matricula | int(11)     | NO   | PRI | NULL    | auto_increment |
| nome      | varchar(40) | NO   |     |         |                |
| email     | varchar(40) | YES  |     | NULL    |                |
| dataNasc  | date        | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Comando insert

Utilizamos esse comando para inserir dados em uma tabela. Para cadastrarmos um aluno, utilizamos insert e informamos os valores na ordem em que as colunas foram definidas. Para cadastrar um aluno, devemos informar a matrícula, nome, e-mail e data de nascimento nessa ordem, pois é como se encontra na tabela. Ex:

```
mysql> insert into aluno values(null,'Zé da feira','feira@localhost','1974-12-
17'); Query OK, 1 row affected (0.00 sec)
```

O comando acima insere um novo registro na tabela aluno, informando seu nome, e-mail e data de nascimento. Note que o parâmetro relativo à sua chave primária foi definido como null. Isso fará com que o MySQL numere automaticamente o campo. Por ser o primeiro registro da tabela, terá a matrícula número 1.

Existem casos em que a inserção de dados em uma tabela ocorre apenas em alguns campos específicos. Podemos informar os campos que serão preenchidos, lembrando que todos os campos que são do tipo NOT NULL devem receber um valor obrigatoriamente.

Em nosso exemplo, a tabela aluno deve receber obrigatoriamente os valores *matricula* e *nome:*

```
mysql> insert into aluno(matricula,nome) values (null,'maria bonita');
Query OK, 1 row affected (0.01 sec)
```

Os registros em nossa tabela ficarão dessa forma:

```
+-----------+--------------+-----------------+------------+
| matricula | nome         | email           | dataNasc   |
+-----------+--------------+-----------------+------------+
|         1 | Zé da feira  | feira@localhost | 1974-12-17 |
|         2 | maria bonita | NULL            | NULL       |
+-----------+--------------+-----------------+------------+
```

Note que os campos em que não foram especificados valores receberam o valor padrão NULL.

Cláusula where

A cláusula where é adicionada quando queremos que o comando seja executado dentro de uma condição. Essa condição pode ser uma comparação entre valores, uma ocorrência especial, etc. Podemos utilizar o where dentro dos comandos insert, update, drop, etc. Veremos o where sendo utilizado no comando update:

Comando update

O comando update é utilizado para atualizar um registro da tabela. Caso a cláusula where não seja informada, todos os registros da tabela serão atualizados com o valor informado:

```
mysql> update aluno set email='bonita@localhost' where matricula = 2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

No exemplo acima, o comando update ajustou o e-mail para 'bonita@localhost'. A cláusula where determina que a atualização somente deve ocorrer quando o número da matrícula for igual a 2. Se essa condição não fosse informada, todos os registros da tabela teriam o e-mail ajustado para 'bonita@localhost':

```
+-----------+--------------+------------------+------------+
| matricula | nome         | email            | dataNasc   |
+-----------+--------------+------------------+------------+
|         1 | Zé da feira  | feira@localhost  | 1974-12-17 |
|         2 | maria bonita | bonita@localhost | NULL       |
+-----------+--------------+------------------+------------+
```

Comando alter table

Utilzamos para alterar a estrutura de uma tabela. Podemos adicionar (add), ajustar (set), excluir (drop) uma coluna, etc:

```
mysql> alter table aluno add cidade varchar(100) after dataNasc;
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

O comando acima altera a tabela aluno, adicionando a coluna endereço, do tipo varchar(100) depois da coluna dataNasc. A estrutura da tabela pode ser vista com o comando *desc aluno;* :

```
mysql> desc aluno;
```

```
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| matricula | int(11)      | NO   | PRI | NULL    | auto_increment |
| nome      | varchar(40)  | NO   |     |         |                |
| email     | varchar(40)  | YES  |     | NULL    |                |
| dataNasc  | date         | YES  |     | NULL    |                |
| cidade    | varchar(100) | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
5 rows in set (0.01 sec)
```

Vamos agora excluir a coluna cidade:

```
mysql> alter table aluno drop cidade;
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

A tabela aluno volta a ter a estrutura anterior:

```
mysql> desc aluno;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| matricula | int(11)     | NO   | PRI | NULL    | auto_increment |
| nome      | varchar(40) | NO   |     |         |                |
| email     | varchar(40) | YES  |     | NULL    |                |
| dataNasc  | date        | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Comando delete

Utilizado para excluir um registro da tabela. Se aplicado sem a cláusula where, elimina todos os registros da tabela:

```
mysql> delete from aluno where matricula=2;
Query OK, 1 row affected (0.00 sec)
```

Resultado:

```
+-----------+--------------+-----------------+------------+
| matricula | nome         | email           | dataNasc   |
+-----------+--------------+-----------------+------------+
|         1 | Zé da feira  | feira@localhost | 1974-12-17 |
+-----------+--------------+-----------------+------------+
```

Comando select

Utilizado para realizar pesquisas dentro de uma tabela. Considerado um comando complexo devido a grande gama de combinações possíveis para a realização da pesquisa. Para auxiliar na compreensão do comando select, alguns registros foram adicionados à tabela aluno. Para pesquisarmos todos os registros dentro de uma tabela, a sintaxe é:

```
mysql> select * from aluno;
+-----------+--------------------+------------------------+------------+
| matricula | nome               | email                  | dataNasc   |
+-----------+--------------------+------------------------+------------+
|         1 | Zé da Feira        | feira@localhost        | 1973-12-17 |
|         2 | Maria Bonita       | bonita@localhost       | 1945-05-24 |
|         3 | Virgulino Ferreira | lampiao@localhost      | 1947-06-19 |
|         4 | Getúlio Vargas     | vargas@localhost       | 1910-04-12 |
|         5 | Raul Seixas        | malucobeleza@localhost | 1957-06-28 |
```

```
+----------+-------------------+----------------------+-----------+
5 rows in set (0.00 sec)
```

O sinal ' * ' diz para o comando select buscar todos os valores da tabela. Podemos especificar uma ou mais colunas para que a pesquisa seja realizada:

mysql> select nome,email from aluno;

```
+-------------------+-----------------------+
| nome              | email                 |
+-------------------+-----------------------+
| Zé da Feira       | feira@localhost       |
| Maria Bonita      | bonita@localhost      |
| Virgulino Ferreira | lampiao@localhost     |
| Getúlio Vargas    | vargas@localhost      |
| Raul Seixas       | malucobeleza@localhost |
+-------------------+-----------------------+
5 rows in set (0.00 sec)
```

No exemplo acima, foram pesquisados apenas os campos nome e email da tabela aluno, que retornou o nome e e-mail de todos os cadastrados. Podemos utilizar a cláusula where para realizar uma busca mais específica dentro da tabela:

mysql> select matricula,nome from aluno where matricula = 3;

```
+-----------+--------------------+
| matricula | nome               |
+-----------+--------------------+
|         3 | Virgulino Ferreira |
+-----------+--------------------+
1 row in set (0.00 sec)
```

Nesse caso, a busca foi feita nas colunas matricula e nome, nos registros cujo número de matrícula é igual a 3. Podemos também ordenar os valores da pesquisa usando a cláusula order by:

```
mysql> select matricula,nome,dataNasc from aluno order by nome;
+-----------+--------------------+------------+
| matricula | nome               | dataNasc   |
+-----------+--------------------+------------+
|         4 | Getúlio Vargas     | 1910-04-12 |
|         2 | Maria Bonita       | 1945-05-24 |
|         5 | Raul Seixas        | 1957-06-28 |
|         3 | Virgulino Ferreira | 1947-06-19 |
|         1 | Zé da Feira        | 1973-12-17 |
+-----------+--------------------+------------+
5 rows in set (0.00 sec)
```

Note que os registros pesquisados foram ordenados pelo nome. Podemos também ordenar pela data. Independente da coluna utilizada, a ordenação pode ser ascendente (asc) ou descendente (desc):

```
mysql> select matricula,nome,dataNasc from aluno order by dataNasc asc;
```

```
+-----------+--------------------+------------+
| matricula | nome               | dataNasc   |
+-----------+--------------------+------------+
|         4 | Getúlio Vargas     | 1910-04-12 |
|         2 | Maria Bonita       | 1945-05-24 |
|         3 | Virgulino Ferreira | 1947-06-19 |
|         5 | Raul Seixas        | 1957-06-28 |
|         1 | Zé da Feira        | 1973-12-17 |
+-----------+--------------------+------------+
5 rows in set (0.00 sec)
```

O exemplo acima foi ordenado pela data de nascimento, em ordem ascendente.

```
mysql> select matricula,nome,dataNasc from aluno order by dataNasc desc;
+-----------+--------------------+------------+
| matricula | nome               | dataNasc   |
+-----------+--------------------+------------+
|         1 | Zé da Feira        | 1973-12-17 |
|         5 | Raul Seixas        | 1957-06-28 |
|         3 | Virgulino Ferreira | 1947-06-19 |
|         2 | Maria Bonita       | 1945-05-24 |
|         4 | Getúlio Vargas     | 1910-04-12 |
+-----------+--------------------+------------+
5 rows in set (0.00 sec)
```

Mesmo exemplo, agora com a data em ordem descendente.

Em alguns casos, a busca não é feita por valores exatos (ex: busca por nome). Existe uma cláusula a ser adicionada ao comando select que busca valores idênticos ao fornecido: a cláusula like. O valor buscado pode encontrar-se entre o sinal '%' para que a busca não se importe com valores que existam antes ou depois do valor buscado. Ex: Vamos adicionar três registros com nomes idênticos:

```
mysql> insert into aluno values(null,'Paulo Ricardo','rpm@localhost','1960-04-
09');
Query OK, 1 row affected (0.00 sec)

mysql> insert into aluno values(null,'Paulo Roberto','proberto@localhost','1982-
07-17');
Query OK, 1 row affected (0.00 sec)

mysql> insert into aluno values(null,'Pedro Paulo
Diniz','diniz@localhost','1979-06-10');
Query OK, 1 row affected (0.00 sec)
```

Os três novos registros têm em comum o nome 'Paulo'. Vamos agora realizar uma pesquisa entre os alunos que possuem 'Paulo' no nome:

```
mysql> select matricula, nome, dataNasc from aluno where nome like '%paulo%';
+-----------+------------------+------------+
| matricula | nome             | dataNasc   |
+-----------+------------------+------------+
|         6 | Paulo Ricardo    | 1960-04-09 |
|         7 | Paulo Roberto    | 1982-07-17 |
|         8 | Pedro Paulo Diniz | 1979-06-10 |
+-----------+------------------+------------+
3 rows in set (0.00 sec)
```

Como o sinal ' % ' foi indicado antes e depois do nome 'paulo', qualquer valor antes ou depois do nome foi ignorado. Se for informado apenas depois do nome, apenas os dois primeiros registros serão listados:

mysql> select matricula, nome, dataNasc from aluno where nome like 'paulo%';

```
mysql> select matricula, nome, dataNasc from aluno where nome like 'paulo%';
+-----------+---------------+------------+
| matricula | nome          | dataNasc   |
+-----------+---------------+------------+
|         6 | Paulo Ricardo | 1960-04-09 |
|         7 | Paulo Roberto | 1982-07-17 |
+-----------+---------------+------------+
2 rows in set (0.00 sec)
```

Chaves estrangeiras

O uso de chaves estrangeiras é útil para melhor organização dos dados, evitando redundância e facilitando buscas. Ex: Agora, cada aluno deve informar o estado em que mora. Vamos então criar uma tabela, onde cadastraremos os estados e relacionaremos cada estado com um aluno, utilizando chaves estrangeiras:

```
 mysql> create table estado(
    -> uf char(2) not null primary key,
    -> estado varchar(40) not null
    -> );
Query OK, 0 rows affected (0.01 sec)
```

O comando acima cria uma tabela estado com os campos UF e estado. Note que UF é uma chave primária, visto que cada estado possui sua própria sigla. Vamos cadastrar apenas alguns estados nessa tabela:

```
mysql> insert into estado values('MG','Minas Gerais');
Query OK, 1 row affected (0.00 sec)

mysql> insert into estado values('BA','Bahia');
Query OK, 1 row affected (0.00 sec)

mysql> insert into estado values('PR','Paraná');
Query OK, 1 row affected (0.00 sec)

mysql> insert into estado values('CE','Ceará');
Query OK, 1 row affected (0.00 sec)

mysql> insert into estado values('RS','Rio Grande do Sul');
Query OK, 1 row affected (0.00 sec)
```

Tipos de tabelas

As tabelas do MySQL possuem tipos. O tipo padrão de tabela é MyISAM. Todas as tabelas são criadas dessa forma. É um tipo de tabela mais rápido, mas que não permite relacionamentos através de chaves estrangeiras. Para que este relacionamento possa ser utilizado, i tipo da tabela deve ser innoDB, que é um tipo mais lento, mas que trabalha com chaves estrangeiras. O tipo da tabela pode ser informado no momento da criação da tabela, utilizando a cláusula engine no final dos atributos:

```
 mysql> create table estado(
    -> uf char(2) not null primary key,
    -> estado varchar(40) not null
    -> )engine=innodb;
```

Caso a tabela já exista, podemos alterá-la:

```
mysql> alter table aluno engine=innodb;
Query OK, 8 rows affected (0.15 sec)
Records: 8  Duplicates: 0  Warnings: 0

mysql> alter table estado engine=innodb;
Query OK, 5 rows affected (0.05 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Agora, vamos alterar a tabela aluno, adicionando o campo estado, onde colocaremos apenas a chave primária que identifica cada estado (nesse caso, uf). Para isso, a nova coluna deve ser ajustada com a cláusula *foreign key.*

Primeiramente, vamos criar a coluna estado na tabela aluno:

```
mysql> alter table aluno add estado char(2);
Query OK, 8 rows affected (0.09 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

Agora, vamos defini-la como chave estrangeira que referencia a coluna uf da tabela estado:

```
mysql> alter table aluno add foreign key(estado) references estado(uf);
Query OK, 8 rows affected (0.14 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

Vejamos como ficou a estrutura da tabela aluno:

```
mysql> desc aluno;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| matricula | int(11)     | NO   | PRI | NULL    | auto_increment |
| nome      | varchar(40) | NO   |     |         |                |
| email     | varchar(40) | YES  |     | NULL    |                |
| dataNasc  | date        | YES  |     | NULL    |                |
| estado    | char(2)     | YES  | MUL | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

Agora, vamos atualizar os registros da tabela aluno, informando o estado de cada um:

```
mysql> update aluno set estado='PR' where matricula=1;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

O aluno com o número de matrícula igual a 1 teve seu estado alterado para PR. Vamos atualizar os demais:

```
mysql> update aluno set estado='CE' where matricula=2;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update aluno set estado='CE' where matricula=3;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update aluno set estado='RS' where matricula=4;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update aluno set estado='BA' where matricula=5;
Query OK, 1 row affected (0.02 sec)
```

```
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update aluno set estado='PR' where matricula=6;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update aluno set estado='MG' where matricula=7;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update aluno set estado='MG' where matricula=8;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Vejamos como ficou o registro dos alunos:

```
mysql> select * from aluno;
+-----------+--------------------+------------------------+------------+-------+
| matricula | nome               | email                  | dataNasc   | estado|
+-----------+--------------------+------------------------+------------+-------+
|         1 | Zé da Feira        |feira@localhost         | 1973-12-17 | PR    |
|         2 | Maria Bonita       | bonita@localhost       | 1945-05-24 | CE    |
|         3 | Virgulino Ferreira | lampiao@localhost      | 1947-06-19 | CE    |
|         4 | Getúlio Vargas     | vargas@localhost       | 1910-04-12 | RS    |
|         5 | Raul Seixas        | malucobeleza@localhost | 1957-06-28 | BA    |
|         6 | Paulo Ricardo      | rpm@localhost          | 1960-04-09 | PR    |
|         7 | Paulo Roberto      | proberto@localhost     | 1982-07-17 | MG    |
|         8 | Pedro Paulo Diniz  | diniz@localhost        | 1979-06-10 | MG    |
+-----------+--------------------+------------------------+------------+-------+
8 rows in set (0.00 sec)
```

Relacionando tabelas

Agora que temos a chave estrangeira definida, podemos listar o nome do estado ao qual pertence cada aluno, referenciando a chave estrangeira da tabela. Ex: Vamos selecionar todos os alunos de Minas Gerais:

```
mysql> select aluno.nome, estado.estado from aluno, estado where estado.uf =
aluno.estado and estado.uf='MG';
+-------------------+--------------+
| nome              | estado       |
+-------------------+--------------+
| Paulo Roberto     | Minas Gerais |
| Pedro Paulo Diniz | Minas Gerais |
+-------------------+--------------+
2 rows in set (0.05 sec)
```

A pesquisa utiliza um select que busca informações nas duas tabelas (aluno e estado), obtendo valor da coluna nome da tabela aluno (aluno.nome) e o valor do campo estado da tabela estado (estado.estado). O segredo está na condição where: A pesquisa retornará os resultados em que o campo estado da tabela aluno seja igual ao campo estado da tabela estado (aluno.estado = estado.uf) e a UF do estado seja MG (estado.uf = MG).

Dessa forma não precisamos armazenar os nomes dos estados mais de uma vez. Caso haja uma mudança no nome dos estados, essa informação é facilmente atualizada.

Relacionamentos entre tabelas

Tabelas de um banco de dados podem ser comparadas a conjuntos. Esses conjuntos

(tabelas) podem ser relacionados de três formas:

1. Um para um: Ocorre quando cada elemento da primeira tabela está relacionado com apenas um elemento da segunda tabela. ex: Um livro somente pode ser publicado por uma editora, portanto, a relação entre livro e editora é de um para um;
2. Um para muitos (muitos para um): Ocorre quando um elemento da primeira tabela está relacionado a mais de um elemento da segunda tabela. Ex: Um funcionário trabalha apenas em uma empresa, porém, uma empresa pode ter vários funcionários. O relacionamento entre funcionário e empresa é de muitos para um (um para muitos);
3. Muitos para muitos: Ocorre quando muitos elementos da primeira tabela se relacionam com muitos elementos da segunda tabela. É o caso de nosso exemplo, pois um aluno pode realizar mais de um curso ao mesmo tempo, enquanto uma turma, obrigatoriamente possui muitos alunos. A relação entre aluno e turma é de muitos para muitos.

No banco de dados, as relações de um para muitos são resolvidas com chaves estrangeiras. A tabela que terá seus atributos repetidos recebe uma chave primária e a fornece como identificador para a outra tabela, como chave estrangeira.

Os casos de relacionamento de muitos para muitos não são tão facilmente representados, exigindo a criação de uma terceira tabela, que relacionará as duas tabelas onde ocorrem esse relacionamento. Ex: Poderíamos criar uma tabela, onde cadastraríamos os cursos oferecidos pela instituição. Um curso deve ter mais de um aluno e um aluno pode fazer mais de um curso. Neste caso, há um relacionamento de muitos para muitos, onde há a necessidade da criação de uma tabela "intermediária", para que possa haver um relacionamento entre os alunos matriculados e os respectivos cursos que freqüentam. Essa tabela poderia conter apenas as chaves primárias das tabelas aluno e curso, realizando assim o relacionamento entre ambos.

Configurando usuários no MySQL

Até agora, todas as operações no banco de dados foram realizadas utilizando o usuário root do mysql. Acessar o banco de dados como root é uma péssima idéia do ponto de vista da segurança. O usuário root tem acesso e permissão de leitura e escrita em qualquer arquivo dentro do servidor. Escrever scripts onde o acesso ao banco de dados é feito através do root é abrir as portas para usuários mal intencionados no servidor. Para evitar que pessoas não autorizadas tenham acesso ao seu banco de dados, devemos criar outro usuário para o mysql e garantir que ele não terá os mesmos privilégios do root.

Sistema de privilégios

Podemos dizer que, para que um usuário possa realizar qualquer operação dentro de um banco de dados, é necessário que ele tenha permissão (privilégio) para isso. Cada operação (insert, update, delete, drop,...) é uma operação diferente e o usuário precisa ter permissão para executá-las. O MySQL (assim como outros SGBD's) permite que gerenciemos esses privilégios, escolhendo assim quais operações poderão ser realizadas

pelo novo usuário. Podemos também determinar os bancos de dados, tabelas e até as colunas em que esse usuário poderá atuar.

Concedendo privilégios – comando GRANT

O comando grant é responsável pela criação do novo usuário, bem como a concessão de privilégios ao mesmo. Através deste comando, configuramos um novo usuário para o MySQL, descrevemos quais serão seus privilégios e quais bancos de dados, tabelas e colunas em que ele terá permissão de atuar. Sua sintaxe é:

```
grant privilégios on (banco.tabela | tabela.coluna) to usuario@servidor [
identified by 'senha' ];
```

Note que podemos configurar uma senha de acesso para o novo usuário utilizando esse comando. Ex: Vamos criar um usuário que terá todos os privilégios sobre o banco de dados *projeto:*

```
grant all privileges on projeto.* to testes@localhost identified by 'senha123';
```

O parâmetro *all privileges* indica que o usuário terá todos os privilégios concedidos. *projeto.** indica que estes privilégios serão concedidos para serem utilizados no banco de dados *projeto,* em qualquer tabela pertencente a este banco. testes@localhost indica que o nome do novo usuário é *testes,* e que o mesmo tem acesso apenas ao servidor *localhost.* Se informássemos o símbolo '%' no lugar de *localhost,* indicaríamos que o usuário testes teria acesso a qualquer servidor em que o MySQL atua. Em *identified by 'senha123',* configuramos a senha de acesso para esse usuário.

Caso queiramos revogar os privilégios do usuário, utilizaremos o comando revoke:

```
revoke all privileges to testes@localhost;
```

Note que não precisamos informar a senha do usuário que terá os privilégios revogados. Esse comando revoga todos os privilégios do usuário testes, concedidos anteriormente.

Para uma maior segurança, devemos conceder o mínimo possível de privilégios a um usuário. Vamos conceder ao usuário testes, os privilégios select, insert, update e delete:

```
grant select, insert, update, delete on projeto.* to testes@localhost identified
by 'senha123';
```

Isso faz com que o usuário testes possua apenas os privilégios citados. Além disso, nosso usuário terá acesso apenas ao banco de dados *projeto.*

Conectando-se ao MySQL com o PHP

Como foi mencionado anteriormente, o banco de dados MySQL foi desenvolvido para ser utilizado em conjunto com o PHP. O MySQL é um banco de dados nativo da linguagem. Apesar disso, o PHP também pode ser utilizado com outros bancos de dados, assim como o MySQL pode ser acessado por outras linguagens CGI (ex: C++, Perl, Python, Ruby, etc).

Para que o PHP possa acessar o banco de dados MySQL, devemos utilizar funções específicas para conexão ao banco de dados.

Conexão

Para conectarmos ao MySQL, utilizamos a função mysql_connect(), que possui a seguinte sintaxe:

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

servidor: endereço do servidor do banco de dados (caso seja sua própria máquina, o endereço é *localhost* ou *127.0.0.1);*

usuario: nome de usuário utilizado para acessar o banco de dados;

senha: senha do usuário que fará a conexão;

Esse comando retorna um identificador para a conexão (ponteiro) em caso de sucesso, ou **false** em caso de falha. Este ponteiro pode ser armazenado dentro de uma variável PHP para utilização posterior (ex: para fechar a conexão).

Ex: para nos conectarmos ao banco de dados *projeto* como usuário *testes:*

```
$con = mysql_connect('localhost','testes','senha123');
```

A variável *$con* receberá o identificador da conexão, que poderemos utilizar para identificá-la quando necessário. Feita a conexão, devemos selecionar o banco de dados no qual deveremos trabalhar. Você pode passar o nome do banco de dados como quarto parâmetro da conexão ou utilizar a função mysql_select_db() para isso:

```
mysql_select_db($projeto);
```

Executando query's SQL

Os comando SQL (query) podem ser executadas através da função mysql_query(). Ela recebe uma query como parâmetro e (opcionalmente) o identificador da conexão. Isso é muito útil em caso de aplicações que conectam-se a mais de um banco de dados ao mesmo tempo. Essa função retorna o resultado da execução da query. Ex: para executar um select na tabela cliente:

```
$res = mysql_query("select * from cliente;");
```

O resultado da busca (select) ficará armazenado na variável $res, e poderá ser manipulado mais adiante. Para melhor exemplificarmos, vamos utilizar a tabela cliente, com a seguinte estrutura:

```
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| id     | int(11)     | NO   | PRI | NULL    | auto_increment |
| nome   | varchar(80) | NO   |     |         |                |
| email  | varchar(50) | YES  |     | NULL    |                |
| estado | char(2)     | NO   |     |         |                |
+--------+-------------+------+-----+---------+----------------+
```

Você também pode armazenar a query em uma variável e usá-la como parâmetro para a função mysql_query(). Isso é muito útil em casos em que é necessário a criação de query's dinâmicas. Ex: Se o usuário pede para que o resultado da busca na tabela clientes seja ordenado pelo nome dos clientes:

```
$query = "select * from cliente";
```

```
if($ordenar==true)
      $query .= " order by nome";
$query .= ";";
$busca = mysql_query(query);
```

No exemplo acima, se a variável $ordenar for verdadeira, o parâmetro "order by nome", é adicionado à query, para que o resultado seja ordenado por nome.

Organizando os dados da consulta

Quando executamos um *select* no banco de dados, todas as informações ficam contidas dentro de uma variável (no exemplo acima, o resultado da pesquisa fica todo armazenado na variável $busca). Para que o resultado seja exibido, é necessário que ele seja organizado.

Retornando o número de linhas

Quando efetuamos uma pesquisa, uma determinada quantidade de resultados é obtida. No nosso exemplo, a busca "select * from cliente" retornará como resultado todas as informações de todos os clientes cadastrados. O MySQL (e outros SGBD's) retornam o número de linhas encontradas em uma pesquisa. Como vimos anteriormente, cada linha representa um registro, portanto, no exemplo acima, a quantidade de linhas retornadas na pesquisa será a quantidade de clientes cadastrados no banco. Para descobrirmos a quantidade de linhas retornadas, utilizamos a função mysql_num_rows(), que recebe a variável que contém o resultado da query como padrão. Essa função retorna um valor inteiro, que pode ser armazenado dentro de outra variável e utilizado para informar o número de clientes encontrados. Ex:

```
$num_clientes = mysql_num_rows($busca);
```

Se tivermos 7 clientes cadastrados, a variável *$num_clientes* receberá o valor 7. Este valor é utilizado na exibição de resultados, como valor de controle para uma estrutura de repetição. Assim, podemos fazer com que a estrutura de repetição (em geral, um *for)* mostre todos os resultados obtidos na pesquisa. Assim, para exibir todos os resultados, teríamos um código semelhante a este:

```
$busca = mysql_query("select * from cliente")or die("não faz pesquisa");
$num_busca = mysql_num_rows($busca)or die("não retorna num. de buscas feitas");
for($i=0;$i<$num_busca;$i++){
/*mostra os resultados*/
}
```

Em alguns casos, queremos executar alterações nos registros da tabela (ex: alterar o endereço de um cliente ou excluí-lo) e precisamos saber quantos registros foram afetados com a execução da query. Para isso, utilizamos a função mysql_affected_rows(), que retorna o número de registros afetados pela execução da query. Ex: Vamos atualizar o estado dos clientes para 'SC':

```
$query = " update cliente set estado='SC' ; ";
$alt = mysql_query($query);
$num_alt = mysql_affected_rows($con)or die("nao mostrou linhas afetadas");
echo "$num_alt linhas afetadas<br>";
```

Será exibida uma mensagem, informando o número de linhas afetadas, ou seja, o número de registros que foram alterados com a query.

Após obtermos o número de linhas retornadas, podemos formatar os dados para exibição. Para isso, devemos organizar os dados obtidos na pesquisa que estão dentro da variável *$busca.* Para realizar esta tarefa, podemos utilizar 3 funções diferentes:

A primeira é a função **mysql_fetch_row()**, que recebe a variável que contém o resultado da pesquisa e retorna o resultado em forma de linha. Para acessar os dados, devemos informar o número da linha correspondente. Ex:

```
$busca = mysql_query("select * from cliente")or die("não faz pesquisa");
$num_busca = mysql_num_rows($busca)or die("não retorna num. de buscas feitas");

for($i=0;$i<$num_busca;$i++){
  $resultado = mysql_fetch_row($busca);
  echo $resultado[0];
  echo $resultado[1];
  echo $resultado[2];
  echo $resultado[3].”<br>";
}
```

Os resultados da busca foram organizados em linhas. Cada linha representa uma coluna da tabela. A variável $resultado[0] contém o id do cliente, $resultado[1] o nome, $resultado[2] seu e-mail e resultado[4] seu estado.

Outra forma de organizarmos os resultados obtidos é utilizando a função **mysql_fetch_array(),** que retorna o resultado da pesquisa como um vetor. Para uma melhor organização, podemos acrescentar o parâmetro **MYSQL_ASSOC**, que faz com que os resultados sejam organizados em um array associativo, permitindo um melhor acesso e controle na exibição dos dados. Ex:

```
for($i=0;$i<$num_busca;$i++){
  $resultado = mysql_fetch_array($busca,MYSQL_ASSOC);
  echo $resultado['id'];
  echo $resultado['nome'];
  echo $resultado['email'];
  echo $resultado['estado']."<br>";
}
```

Isso permite que os dados sejam acessados conforme o nome das colunas da tabela.

A terceira forma de organização dos dados é utilizando a função mysql_fetch_object(), que retorna o resultado como um objeto (O conceito de orientação a objetos será visto mais adiante).:

```
for($i=0;$i<$num_busca;$i++){
  $resultado = mysql_fetch_object($busca);
  echo $resultado->id;
  echo $resultado->nome;
  echo $resultado->email;
  echo $resultado->estado."<br>";
}
```

Fechando a conexão

A função responsável pelo fechamento da conexão com o banco de dados é mysql_close(). Essa função recebe o ponteiro da conexão como parâmetro. Ex:

```
mysql_close($con);
```

Esse comando irá fechar a conexão iniciada anteriormente em nosso exemplo. Isso

reduz o número de conexões ativas no servidor. Bancos de dados em geral aceitam um número limitado de conexões simultâneas. A utilização da função mysql_close() libera recursos do servidor para novas conexões, permitindo maior eficiência da aplicação.

Aplicações em PHP e MySQL

Até agora, vimos como o banco de dados MySQL pode ser utilizado com o PHP em exemplos simples e isolados. Agora, vamos construir pequenas aplicações utilizando essas poderosas ferramentas, a fim de compreendermos melhor seu funcionamento. Para a construção de nossas aplicações, vamos utilizar a tabela cliente, que está contida no banco de dados projeto. Vamos também criar o usuário 'user', com a senha 'senha123' e conceder a ele os privilégios select, insert,update e delete para o banco. Os exemplos apresentados aqui são apenas scripts independentes. Os códigos estarão comentados para melhor compreensão. Para a construção de uma aplicação completa, os módulos deverão ser integrados, mas por questão didática, veremos o funcionamento de cada módulo separadamente. Vamos construir os módulos de cadastro, alteração, pesquisa e exclusão, para isso, Digite os seguintes comandos no terminal do MySQL:

```
//conectando-se ao banco de dados
mysql -u root -p

//criando o banco de dados projeto
create database projeto;

//selecionando o banco projeto
use projeto;

//criando a tabela cliente
create table cliente(
      id int not null auto_increment primary key,
      nome varchar(80) not null,
      email varchar(80) not null,
      estado char(2) not null
      )engine=innoDB;

//configurando o usuário do banco
grant select,insert,update,delete on projeto.* to user@localhost identified by
'senha123';
```

Precisaremos acessar o banco de dados constantemente e informarmos os parâmetros necessários para a conexão. Para facilitar nosso trabalho, vamos escrever o seguinte código:

```
<?php
$servidor = 'localhost';
$usuario = 'teste';
$senha = 'senha123';
$banco = 'projeto';
$con = mysql_connect($servidor,$usuario,$senha);
mysql_select_db($banco);
?>
```

Salve-o como mysqlconfig.inc (inc de include). Esse script será chamado toda vez que precisarmos nos conectar ao banco de dados.

Módulo de cadastro

Agora que temos o banco de dados devidamente montado e configurado, vamos criar a primeira parte da aplicação. Precisaremos de um formulário HTML para que o usuário possa digitar os dados:

```
<html>
  <head>
      <title>Cadastro</title>
  </head>
 <body>
<form method="POST" action="cadastrar.php">
<p align="center">Nome: <input name="nome" type="text"></p>
<p align="center">E-mail: <input name="email" type="text"></p>
<p align="center">Estado (UF): <input name="estado" size="2" type="text"></p>
<p align="center"><input value="salvar" type="submit"></p>
</form>
 </body>
</html>
```

Salve-o como cadastrar.html. Agora vamos escrever o script que irá gravar as informações no banco de dados (cadastrar.php):

```
<?php
//captura as informações enviadas pelo formulário
$nome = $_POST['nome'];
$email = $_POST['email'];
$estado = $_POST['estado'];

//conecta-se ao banco de dados
include("mysqlconfig.inc");

//monta a query que irá gravar as informações
$query = "insert into cliente values(null,'$nome','$email','$estado'); ";

//grava as informações
$grava = mysql_query($query);

//conta o número de colunas afetadas. Se for 1, a gravação foi efetuada
$num_linha = mysql_affected_rows($grava);
if($num_linha == 1)
  echo "Cadastro Efetuado com sucesso<br>";

  //link para a página anterior
  echo "< a href='javascript:history.back()'>voltar</a>";

//fecha a conexão
mysql_close($con);
?>
```

O script exibirá uma mensagem, caso o cadastro seja efetuado com sucesso.

Módulo de exclusão

Este módulo é muito simples. Criaremos um formulário, onde informaremos o código do cliente a ser excluído e um script que exclui este cliente. Vamos começar pelo formulário:

```
<html>
  <head>
```

```
    <title>Excluir</title>
  </head>
 <body>
  <form method="POST" action="excluir.php">
    <p align="center">Código: <input type="text" name="id" size="3"></p>
    <p align="center"><input type="submit" value="Excluir">
  </form>
 </body>
</html>
```

Salve-o como excluir.html. Agora, vamos criar os script que irá realizar a exclusão do usuário:

```
<?php
//captura id do formulário
$id = $_POST['id'];

//converte formato para inteiro
$id = intval($id);

//realiza a conexão com o banco de dados
include("mysqlconfig.inc");

//monta a query de exclusão
$query = "delete from cliente where id='$id'; ";

//executa a query
$res = mysql_query($query);

//conta o número de registros afetados
$num_exclui = mysql_affected_rows($res);

//caso um registro tenha sido excluído, exibe mensagem
if($num_exclui == 1)
  echo "Cliente excluído com sucesso<br>";

  //link para a página anterior
  echo "< a href='javascript:history.back()'>voltar</a>";

//fecha a conexão
mysql_close($con);
?>
```

Salve-o como excluir.php.

Módulo de pesquisa – parte 1

Este módulo será relativamente simples. Vamos fazer apenas com que ele liste os clientes cadastrados no banco de dados. Este script deve ser salvo como lista.php:

```
<?php
//conecta-se ao banco de dados
include("mysqlconfig.inc");

//monta a query de busca
```

```
$query = "select * from cliente; ";

//executa a query
$res = mysql_query($query);

//conta o número de registros encontrados na pesquisa
$num_reg = mysql_num_rows($res);

//monta uma tabela para organizar os dados
echo "<table border='0' cellpadding='1' cellspacing='1'> ";

//cria um for para a exibição dos dados
for($i=0;$i<$num_reg;$i++){

//formata resultado para exibição
$mostra = mysql_fetch_array($res);
echo "<tr>";

//exibe os resultados
echo "<td> $mostra['id'] </td>";
echo "<td> $mostra['nome'] </td>";
echo "<td> $mostra['email'] </td>";
echo "<td> $mostra['estado'] </td>";
echo "</tr>";
}

//encerra a tabela
echo "</table>";

//informa a quantidade de registros encontrados
echo "$num_reg encontrados<br>";

//encerra a conexão
mysql_close($con);
?>
```

Esse script monta uma tabela e exibe todos os registros encontrados na tabela cliente.

Módulo de pesquisa – parte 2

Agora, vamos criar um formulário, onde o usuário informará um nome a ser pesuisado na tabela cliente. Vamos começar com o formulário onde o usuário informará o nome:

```
<html>
  <head>
    <title>Buscar</title>
  </head>
 <body>
  <form method="POST" action="buscar.php">
    <p align="center">Nome: <input type="text" name="nome" size="30"></p>
    <p align="center"><input type="submit" value="Buscar">
  </form>
 </body>
</html>
```

Salve-o como buscar.html. O nome informado neste formulário será colocado na

query de pesquisa do script buscar.php, que irá listar todos os clientes registrados no banco com o nome igual ou idêntico ao informado:

```
<?php
//captura o nome informado no formulário
$nome = $_POST['nome'];

//conecta-se ao banco de dados
include("mysqlconfig.inc");

//monta a query de busca, inserindo o nome como parâmetro
$query = "select * from cliente where nome like'%'$nome'%'; ";

//executa a query
$res = mysql_query($query);

//conta o número de registros encontrados na pesquisa
$num_reg = mysql_num_rows($res);

//monta uma tabela para organizar os dados
echo "<table border='0' cellpadding='1' cellspacing='1'> ";

//cria um for para a exibição dos dados
for($i=0;$i<$num_reg;$i++){

//formata resultado para exibição
$mostra = mysql_fetch_array($res);
echo "<tr>";

//exibe os resultados
echo "<td> $mostra['id'] </td>";
echo "<td> $mostra['nome'] </td>";
echo "<td> $mostra['email'] </td>";
echo "<td> $mostra['estado'] </td>";
echo "</tr>";
}

//encerra a tabela
echo "</table>";

//encerra a conexão
mysql_close($con);
?>
```

Note que este script é semelhante ao anterior, porém, o nome informado no formulário de busca é colocado como parâmetro na query que é executada. O parâmetro like indica que a busca deve ser feita com valores idênticos, ou seja, se o usuário informar o nome Paulo, todos os clientes com o nome de Paulo serão listados.

Módulo de alteração

O módulo de alteração exige que seja dividido em 3 partes: Na primeira parte, um formulário html, onde informaremos o código (id) do cliente que será alterado. Na segunda parte, um script PHP que exibe as informações do cliente e um botão de confirmação. Na terceira parte, o script que executa a alteração dos dados do cliente. Vamos começar com o formulário html, onde o usuário informará o id do cliente a ser alterado:

```
<html>
  <head>
    <title>Alterar</title>
  </head>
 <body>
  <form method="POST" action="exibir.php">
    <p align="center">Nome: <input type="text" name="id" size="3"></p>
    <p align="center"><input type="submit" value="Exibir">
  </form>
 </body>
</html>
```

O código informado neste formulário será enviado para o script "exibir.php", que irá mostrar os dados do cliente. Ele conterá um formulário idêntico ao de cadastro, porém, o atributo value das tag's fará com que os valores já existentes sejam mostrados dentro dos campos:

```
<?php
//captura o código informado no formulário
$id = $_POST['id'];

//conecta-se ao banco de dados
include("mysqlconfig.inc");

//monta a query de busca, inserindo o id como parâmetro
$query = "select * from cliente where id='$id'; ";


//executa a query
$res = mysql_query($query);

//formata resultado para exibição
$mostra = mysql_fetch_array($res);

//monta o formulário para exibição, que leva os dados
//para o script de gravação
echo "<form method='POST' action='salvar.php'>";

//exibe os resultados. O html será exibido fora do PHP
?>

<p align='center'>Código:
  <?php echo $mostra['id'] ?>
</p>

<p align='center'>Nome:
  <input type='text' name='nome' value='<?php echo $mostra['nome']; ?> '>
</p>

<p align='center'>E-mail:
  <input type='text' name='email' value='<?php echo $mostra['email']; ?> '>
</p>

<p align='center'>Estado:
```

```
  <input type='text' name='estado' value='<?php echo $mostra['estado']; ?>'
size='2'>
</p>

<p align='center'>
  <input type='submit' value='Salvar'>
</p>

</form>
<?php

//encerra a conexão
mysql_close($con);
?>
```

Os dados do cliente são exibidos dentro das tag's do formulário (com exceção do id, que deve permanecer o mesmo). Este mesmo formulário que está exibindo as informações do cliente está "direcionado" para um script chamado "salvar.php". Podemos fazer as alterações necessárias nos dados que estão sendo exibidos neste script e então, quando clicarmos no botão "Salvar", os novos dados (e os antigos, isto é, que não foram alterados) serão enviados para o script "salvar.php", que irá realizar a efetiva gravação dos dados:

```
<?php
//captura o código informado no formulário
$id = $_POST['id'];
$nome = $_POST['nome'];
$email = $_POST['email'];
$estado = $_POST['estado'];

//converte para inteiro
$id = (integer) $id;

//conecta-se ao banco de dados
include("mysqlconfig.inc");

//monta a query de alteração
$query = "update cliente set nome='$nome', email='$email', estado='$estado'
where id='$id'; ";

//executa a query
$res = mysql_query($query);

//conta o número de colunas alteradas. Se for 1, a alteração foi efetuada com
sucesso
$num_alt = mysql_affected_rows($res);
if($num_alt == 1)
  echo "Alteração efetuada com sucesso!!!<br>";

//encerra a conexão
mysql_close($con);
?>
```

Note que, conforme foi especificado na query, os dados serão alterados enquanto o id do cliente for igual ao id fornecido, ou seja, isso garante que apenas o cliente em questão terá seus dados alterados.

Cookies e Sessões - Autenticação

Cookies

Você já deve ter vivido essa situação: Você acessa seu perfil em um site de relacionamentos e navega por ele por algum tempo. Depois você fecha seu navegador para terminar um trabalho que deixou para trás. Logo depois, você entra naquele site de relacionamentos novamente e, qual a sua surpresa quando percebe que o site já abre diretamente em seu perfil, sem que você precise digitar seu nome de login e sua senha novamente. Então você se pergunta: "Como isso é possível?". A resposta está, primeiramente nos cookies. Eles são pequenos arquivos de texto que o próprio PHP grava em sua máquina, com informações relevantes para a aplicação (ex: seu login e sua senha, para que não tenha que logar-se novamente). Gravar um cookie é uma tarefa muito simples, é necessário apenas o uso da seguinte função:

```
setcookie("nomedocookie","valor a ser gravado");
```

Ex: vamos gravar um cookie contendo a data atual:

```
$datadehoje = date("d-m-Y");
setcookie("datadehoje",$datadehoje);
```

Este cookie agora está gravado na máquina do usuário, e contém uma string com a data atual. Todos os cookies podem ser acessados através do array superglobal $_COOKIE[] da seguinte forma:

```
$datadehoje = $_COOKIE['datadehoje'];
```

Para que um cookie seja excluído, basta utilizar novamente a função setcookie(), porém sem nenhum valor como conteúdo:

```
setcookie("datadehoje");
```

Isso fará com que o cookie seja apagado. O uso de cookies pode parecer uma boa prática a princípio, mas em muitos casos, os usuários configuram os navegadores para não aceitarem cookies, o que dificulta a implementação dos mesmos.

Sessões

Uma alternativa ao uso de cookies é o uso de sessões. Seu uso é idêntico ao dos cookies, porém, os dados não ficam gravados no computador do usuário. Qualquer navegador aceita sessão, por isso torna-se uma boa prática criar autenticação utilizando sesões. Para criar uma sessão, devemos adicionar em nossos scripts PHP a função session_start(). Recomenda-se que ela esteja nas primeiras linhas do script, antes de qualquer processamento:

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

Para registrar uma sessão, podemos inserir dados diretamente no array superglobal $_SESSION[], da seguinte forma:

```
<?php
session_start();
$_SESSION['login'] = $login;
?>
```

O exemplo acima configura a variável de sessão login com o valor contido na variável $login. Também podemos utilizar a função session_register() para registrar uma variável de sessão:

```
<?php
session_start();
session_register("administrador");
?>
```

Dessa forma, registramos uma variável de sessão administrador. Para verificar se uma variável de sessão está registrada, podemos utilizar a função session_is_registered(), que verifica se a variável (passada como parâmetro está registrada:

```
<?php
session_start();
if(session_is_registered("administrador") )
      echo "variável registrada<br>";
else
      echo "variável não está registrada<br>";
?>
```

Outra forma de verificarmos se uma variável de sessão está registrada é adicionando seu conteúdo a uma variável do PHP e então conferindo seu conteúdo, através da função isset(), que verifica se uma variável está registrada:

```
<?php
session_start();
$login = $_SESSION['administrador'];
if(isset($login))
      echo "variável registrada<br>";
else
      echo "variável não registrada<br>";
?>
```

Para excluir o conteúdo de uma variável de sessão, utilizamos a função unset():

```
<?php
session_start();
unset($_SESSION['login']);
?>
```

Isso faz com que o valor da variável de sessão *login* seja removido. Quando o usuário deixa o site ou decide fazer logout e continuar navegando, deve-se então desregistrar as variáveis de sessão do usuário e destruir a sessão iniciada por ele. Isso pode ser feito utilizando a função session_destroy(), que elimina todas as variáveis de sessão registradas. Dessa forma, o script para logout ficaria da seguinte forma:

```
<?php
session_start();
$_SESSION = array();
session_destroy();
?>
```

Note que inserimos dentro do array $_SESSION[] um array completamente vazio. Isso faz com que todas as variáveis de sessão fiquem "em branco". Em seguida, a sessão é destruída com a função session_destroy().

Vamos criar um formulário de login para nosso site. Para isso, vamos utilizar uma nova tabela, chamada usuario. Esta tabela conterá um nome para login e uma senha. Vamos criar a tabela no banco de dados:

```
mysql -u root -p
use projeto
create table usuario(
      login char(30) not null primary key,
      senha char(20) not null
);
```

Agora vamos criar a página de cadastro para os novos usuários:

```
<html>
  <head>
      <title>Cadastro de Usuários</title>
 </head>
<body>
<form method=”POST” action=”usuario.php”>
 <p align='center'>Login: <input type=”text” name=”login” size=”20”></p>
 <p align='center'>Senha: <input type=”text” name=”senha” size=”20”></p>
 <p align='center'><input type=”submit” value=”Cadastrar”></p>
</form>
</body></html>
```

Montado o formulário de cadastro, vamos criar o script PHP que irá gravar as informações:

```
<?php
//recebe as variáveis do formulário
$login = $_POST['login'];
$senha = $_POST['senha'];

//conecta-se ao banco de dados
include “mysqlconfig.inc”;

//monta a query de gravação
$query = “insert into usuario values('$login','$senha'); ”;

//grava os dados
mysql_query($query);

/*
aqui, coloque um link que direcione o usuário para a página principal do site,
ou utilize a função header para levá-lo diretamente para a página principal
vamos considerar que a página principal do nosso site seja meusite.php:
*/

//direciona o usuário para a página de login
header(“Location: login.html”);

//fecha a conexão
mysql_close($con);
?>
```

Salve-o como usuario.php. Ele irá gravar as informações dos novos usuários na tabela usuario, em nosso banco de dados. Vamos agora criar a página e o script de login, onde o usuário informará os dados e, se o usuário existir, as variáveis de sessão serão criadas. Vamos começar com o formulário de login:

```
<html>
  <head>
      <title>Login</title>
 </head>
```

```
<body>
<form method=”POST” action=”login.php”>
 <p align='center'>Login: <input type=”text” name=”login” size=”20”></p>
 <p align='center'>Senha: <input type=”text” name=”senha” size=”20”></p>
 <p align='center'><input type=”submit” value=”Entrar”></p>
</form>
</body></html>
```

Agora, vamos criar o script login.php que irá efeturar o registro das variáveis de sessão, que irão determinar que o usuário está registrado no site:

```
<?php
//recebe as variáveis do formulário
$login = $_POST['login'];
$senha = $_POST['senha'];

//conecta-se ao banco de dados
include “mysqlconfig.inc”;

//query queseleciona o usuário correspondente ao login e senha informados
$query = “select login,senha from usuario where login='$login' and
senha='$senha'; ”;

//executa query
$res = mysql_query($query);

//verifica se query retornou resultados
$num_linha = mysql_num_rows($res);

//se retornou resultado, usuário existe, então, registra-o
if($num_linha > 0){
session_start();
$_SESSION['usuario'] = $login;

//usuário é direcionado para página principal
header(“Location: meusite.php”);
}else{
//caso login seja inválido
echo “usuário não existe<br>”;
//link redireciona para a página login.html
echo “<a href='login.html'>Voltar</a>”;
}
?>
```

Caso o usuário esteja cadastrado no banco de dados, seu login é armazenado numa variável de sessão chamada 'usuario'. Agora, podemos determinar se o usuário está logado no site apenas verificando a variável de sessão. Vamos criar um script que verifique a variável:

```
<?php
session_start();
if( !session_is_registered(“usuario”) ){
      echo “Efetue o login no site:<br>”;
      ehco “<a href='login.html'>Login</a>”;
}
?>
```

Vamos salvá-lo como verifica_sessao.php. Esse script simples verifica se a variável de sessão “usuario” está registrada. Se não estiver, mostra o link para a página de login do site. Dessa forma, se quisermos proteger páginas de nosso site, basta adicionar uma chamada

ao script no início da página. Se o usuário não estiver registrado, não poderá ver o conteúdo da página:

```
<?php
include "verifica_sessao.php";
...
?>
```

Para que a nossa aplicação torne-se completa, devemos oferecer um link para que o usuário possa efetuar logout no site. Basta criar um link direcionado para o script logout.php, mostrado anteriormente.

Para aumentar a segurança, podemos utilizar criptografia para a senha. A criptografia é um recurso que transforma a string em um código difícil de ser interpretado pelo ser humano. Uma das funções que podemos utilizar para isso é a função crypt(), do próprio PHP. Essa função recebe como parâmetro, a string que será criptografada e um conjunto de dois caracteres para serem utilizados como base para a criação da string criptografada. Sua sintaxe é:

crypt(*caracteres, string_para_criptografar);*

Ex: para criptografar a senha informada pelo usuário:

$senha = $_POST['senha'];

$senha = crypt(wh,$senha);

A função crypt usará as letras wh como base e irá criptografar a senha. Caso o usuário tenha informado a senha como "abacate", o resultado seria: abUpSjlq7OP6c . Depois de criptografada, o processo não pode ser revertido. Para conferir se a senha do usuário está correta, informada pelo usuário no momento do login também deve ser criptografada e então as duas são conferidas. A palavra abacate sempre terá o valor abUpSjlq7OP6c se as letras wh forem informadas no momento da criptografia.

Para garantir que o processo não apresentará erros, a string que será criptografada não deverá conter espaços em branco, pois a função não os criptografa.

PHP Orientado a Objetos

O conceito de orientação a objetos tem sido adotado por muitos programadores e linguagens de programação em virtude de sua flexibilidade, capacidade de reaproveitamento do código e facilidade de manutenção. Este novo paradigma tem como princípio a representação de objetos do mundo real em forma de linguagem de programação. Cada componente do programa é tratado como um objeto, com suas características e funcionalidades, dentro de um conceito muito próximo ao mundo real.

Principais componentes

Classes, atributos e métodos

São como "moldes" utilizados para a criação de objetos. Uma classe contém um conjunto de atributos (variáveis) e métodos (funções) determinados peb programador para

compor um devido objeto. As classes devem ser criadas conforme os objetos do mundo real que necessitamos representar. Ex: Vamos implementar a classe automóvel: Para representar um pássaro em linguagem de programação, devemos primeiramente observar seus atributos e métodos. Alguns atributos de um automóvel são: marca, modelo, ano de fabricação, combustível, número de marchas, etc.

Perceba que os atributos são características comuns a todos os automóveis. As classes devem, preferencialmente possuir características genéricas em relação ao objeto que será representado.

A classe automóvel também deve possuir métodos. Métodos são funções que permitem que uma ação seja realizada pelo objeto. Alguns métodos de um automóvel (o que um automóvel pode fazer): ligar, desligar, acelerar, freiar, trocar de marcha, etc.

Agora que temos uma pequena descrição de um automóvel, podemos implementá-lo como um objeto no PHP. Para isso, devemos seguir a sintaxe da programação orientada a objetos. Toda classe é determinada pela palavra reservada class. Seus atributos e métodos ficam dentro de chaves.

```
<?php
class automovel{
//atributos e métodos
}
?>
```

Para representar os atributos da classe automóvel, vamos criar apenas os atributos marca, modelo e ligado, que usaremos para dizer se o automóvel está ligado ou desligado

```
<?php
class automovel{
  //declaração de atributos (variáveis)
  var $marca;
  var $modelo;
  var $ligado;
}
?>
```

Note que utilizamos uma sintaxe diferente nos nomes de variáveis. Essa sintaxe é muito comum em orientação a objetos. Quando um nome é formado pela união de duas palavras distintas, o primeiro nome fica inteiramente em letras minúsculas. Os demais nomes iniciam-se com letra maiúscula. Isso não é uma regra oficial e seu script não vai deixar de funcionar por isso, mas iremos adotar essa sintaxe para uma melhor assimilação.

Vamos agora, criar alguns métodos para a classe automóvel. A criação de métodos é idêntica à criação de funções. Ex: Vamos criar o método liga():

```
<?php
class automovel{
  //declaração de atributos
  var $marca, $modelo, $ligado;

  //criação do método liga()
  function liga(){
    $this->ligado = true;
  }
}
?>
```

Nosso método liga() apenas altera o estado da variável ligado para true, representando um automóvel ligado. Utilizamos o formato $this->atributo para informar que estamos modificando uma variável que pertence à própria classe. Podemos criar um método desliga(), para que o automóvel possa ser desligado:

```
<?php
class automovel{
  //declaração de atributos
  var $marca, $modelo, $ligado;

  //criação do método liga()
  function liga(){
    $this->ligado = true;
  }

  //método desliga
  function desliga(){
    $this->ligado = false;
  }
}
?>
```

Temos uma classe automóvel que liga e desliga, mas como saber se o automóvel está ligado? Vamos criar um método que informa seu estado. Chamaremos esse método de estado():

```
<?php
class automovel{
  //declaração de atributos
  var $marca, $modelo, $ligado;

  //criação do método liga()
  function liga(){
    $this->ligado = true;
  }

  //método desliga
  function desliga(){
    $this->ligado = false;
  }

  //método que informa o estado do automóvel
  function estado(){
    if($this->ligado == true)
      echo "Automóvel ligado<br>";
    else
      echo "Automóvel desligado<br>";
  }
}
?>
```

Agora temos um método que informa se o automóvel está ligado ou desligado. Vamos então realizar o que chamamos de instanciar um objeto. Como vimos no início deste documento, as variáveis em PHP também podem ser do tipo objeto (object). Isso significa que elas podem receber, como valor, toda uma classe, ou seja, podemos determinar que uma variável conterá a classe automóvel. Automaticamente, todos os atributos e métodos que existem na classe automóvel passam a pertencer à variável. Inicializar uma variável como o valor de uma classe é o que chamamos de instanciação de um objeto, pois cada

variável instanciada é tratada como um método diferente; como se cada variável fosse um automóvel diferente. Automóveis possuem as mesmas características (descritas dentro da classe), mas cada automóvel é único. Pense na instanciação como a "linha de montagem" de um objeto (neste exemplo, de um automóvel):

Vamos instanciar um automóvel:

```
<?php
class automovel{
  //declaração de atributos
  var $marca, $modelo, $ligado;

  //criação do método liga()
  function liga(){
    $this->ligado = true;
  }

  //método desliga
  function desliga(){
    $this->ligado = false;
  }

  //método que informa o estado do automóvel
  function estado(){
    if($this->ligado == true)
      echo "Automóvel ligado<br>";
    else
      echo "Automóvel desligado<br>";
  }
}

//instanciando o objeto
$carro = new automovel();

//chamando a função estado
$carro->estado();
?>
```

Note que, para instanciarmos um objeto, utilizamos a palavra reservada new, seguida do nome da classe. A partir deste momento, a variável $carro torna-se um automóvel, e contém todas as características definidas na classe. Se quisermos que o objeto execute um determinado método, basta realizar uma chamada à função, através da sintaxe:

```
$objeto->método
```

Isso faz com que o método contido dentro do objeto seja executado. Em nosso exemplo, queríamos que o método estado() fosse executado, então, fizemos uma chamada ao método:

```
 $carro->estado();
```

Assim, o método estado() contido no objeto $carro foi executado.

Note que nenhum outro atributo foi inicializado neste objeto. Vamos informar os atributos de nosso objeto $carro:

```
$carro->marca = "VolksWagen";
$carro->modelo = "Fusca";
```

Podemos utilizar o comando echo para imprimir diretamente o valor das variáveis:

```
echo "eu tenho um $carro->marca $carro->modelo !!!";
```

Essa linha irá retornar a mensagem: "eu tenho um VolksWagen Fusca !!!";

Podemos, facilmente, criar diversos automóveis, sem que seja necessário reescrever a classe automóvel:

```
<?php
  $carro = new automovel();
  $esportivo = new automovel();
  $jeep = new automovel();
  $carroPopular = new automovel();
?>
```

Temos então $carro, $esportivo, $jeep e $carroPopular como quatro objetos distintos, pertencentes à classe automóvel. Todos eles possuem os mesmos atributos e métodos. Se quisermos ligar apenas o jeep...:

```
<?php
$jeep->liga();
?>
```

Apenas o objeto $jeep estará ligado. Os demais não terão seu estado alterado. Note que o aproveitamento do código é muito maior. Sua manutenção também é facilitada. Ex: Se quisermos modificar o método estado, para que ele também informe a marca e o modelo do automóvel, basta modificar o método estado dentro da classe automóvel e pronto. Todos os objetos estarão alterados:

```
<?php
function estado(){
    if($this->ligado == true)
      echo "auomóvel ligado<br>";
    else
      echo "automóvel desligado<br>";
  echo "$this->marca<br>";
  echo "this->modelo<br>";
  }
?>
```

Quando realizamos a chamada ao método estado do objeto carro:

```
$carro->estado();
```

O resulado obtido é:

```
automóvel ligado
VolksWagen
Fusca
```

Métodos construtores e destrutores

Métodos construtores são utilizados para inicializar uma classe. o método construtor é o primeiro método a ser executado dentro de uma classe, durante a instanciação de um objeto. O método construtor não é obrigatório, mas pode ajudar muito. Métodos construtores em PHP possuem a seguinte sintaxe:

```
<?php
class automovel{

//método constutor recebe $marca e $modelo como parâmetros
  function __construct($marca,$modelo){
```

```
//inicializa os atributos
    $this->marca = $marca;
    $this->modelo = $modelo;
  }
}
?>
```

Ex: Se quisermos determinar marca e modelo de um automóvel no momento da instanciação:

```
$conversivel = new automovel("Ford","Escort");
```

Os parâmetros informados no momento da instanciação são passados para o método construtor, que instancia as variáveis. Vamos executar uma chamada ao método estado:

```
$conversivel->estado();
```

O resultado será:

```
automóvel desligado
Ford
Escort
```

O método __destruct é executado quando o objeto é destruído. Não se pode determinar a forma como o objeto será destruído, ou seja, removido da memória (em geral, acontece no final da execução do script), mas pode-se configurar ações a serem executadas durante a destruição do objeto:

```
function __destruct{
      echo "$this->marca $this->modelo destruído<br>";
}
```

Acrescentando este método destrutor na classe automóvel, o resultado será:

```
automóvel desligado
Ford
Escort
Ford Escort destruído
```

Modificadores de Acesso

São palavras reservadas que determinam a forma como um atributo ou método será acessado pela aplicação. Podemos dizer que eles alteram permissões de acesso aos mesmos. Os principais são:

Modificador public

Um método ou atributo determinado como public pode ser acessado por qualquer método da aplicação. Ex: Os atributos definidos em nossa aplicação até o momento são atributos públicos, pois podemos modificá-los diretamente dentro da aplicação, simplesmente determinando um valor, como uma variável pública:

```
$carro->modelo = "Santana";
```

Modificador private

É o método mais restritivo. Indica que um atributo ou método somente poderá ser acessado de dentro da classe a qual pertence, impedindo o acesso externo. Se declararmos os atributos da classe automóvel como private, apenas métodos pertencentes à classe poderão alterá-los, o que impede que atribuições de valores como a mostrada acima. Para

que um atributo private possa ser alterado, é necessário que seja criado um método capaz de alterá-lo. Vamos declarar os atributos da classe automóvel como private:

```
<?php
class automovel{

//atributos declarados como private
private $marca, $modelo, $ligado;

...

}
?>
```

Em nosso exemplo, o método que modifica os atributos $marca e $modelo é o próprio método construtor, enquanto que o método que modifica o estado do carro (ligado ou desligado) é o método estado. Repare que não há mais como alterar esses atributos sem que uma chamada a um desses métodos seja realizada.

Modificador protected

Este modificador é de um "nível intermediário". Atributos ou métodos declarados como protected tornam-se públicos dentro da classe em que foram criados e das classes que herdam a classe em que eles foram criados. Fora de sua classe e suas herdeiras (veremos isso em Herança, logo adiante), atributos e métodos definidos como protected possuem a mesma restrição obtida com o modificador private.

Herança

Herança é uma forma de reutilizar uma classe. Classes podem herdar características de outras classes, passando a possuir todas as suas características (atributos e métodos). Uma classe herda características de outra classe com o uso da palavra reservada extends. Ex: Vamos criar a classe carroDeCorrida, que irá herdar as características da classe automóvel:

```
class carroDeCorrida extends automovel{
...
}
```

Pronto, agora temos uma nova classe carroDeCorrida, com os atributos $marca, $modelo e $ligado e os métodos ligar(), desligar() e estado(), sem que fosse necessário reescrever toda a classe. Qualquer modificação na classe automovel modifica também a calsse carroDeCorrida. Dizemos então que a classe automovel é a "classe pai" e a classe carroDeCorrida é a "classe filha", pois herda as características da classe pai. Podemos agora criar novos atributos e métodos para a classe carroDeCorrida. Vamos criar o atributo pneu e o método trocaPneu para a classe carroDeCorrida:

```
class carroDeCorrida{
 private $pneu;
  function trocaPneu($tipoPneu){
    $this->pneu = $tipoPneu;
  }
}
```

O método trocaPneu recebe como parâmetro o tipo de pneu que o carro de corrida irá receber na troca e altera o atributo $pneu para o tipo especificado.

Sobrescrever métodos

Uma técnica muito utilizada na programação orientada a objetos. Permite que alteremos todo um método da classe pai, reescrevendo-o. Ex: Na classe pai, o método estado() mostra apenas as características básicas do carro. Vamos alterar a classe carroDeCorrida para que também mostre o tipo do pneu:

```
<?php
class carroDeCorrida extends automovel{
//novo atributo pneu
private $pneu;
//método trocaPneu, definida anteriormente
 function trocaPneu($tipoPneu){
   $this->pneu = $tipoPneu;
 }
//método estado sendo sobreescrita
 function estado(){
//chamada ao método estado da classe pai
  parent::estado();
//nova linha do método estado
 echo "Pneus $this->pneu<br>";
 }
}
?>
```

Note que dentro do novo método estado() há uma chamada ao método estado da classe automóvel (parent::estado()). Essa chamada é necessária para que o PHP identifique o método que está sendo sobrescrito.

Quando utilizamos herança em PHP, devemos nos lembrar sempre de que o PHP ainda possui limitações, se comparado a outras linguagens orientadas a objeto. Ex: As classes que possuem herança devem ser definidas dentro do mesmo arquivo da classe pai, ou seja, em nosso exemplo, a classe carroDeCorrida deve estar no mesmo arquivo da classe automovel, pois o PHP não pode definir qual é a classe pai e qual a classe filha. Porém, isso pode ser resolvido com o uso da função require ou include.

Outra limitação encontrada na linguagem é que o PHP não suporta herança múltipla, ou seja, uma classe filha pode ter apenas uma classe pai, diferente de outras linguagens como java, que suporta essa característica.

Vamos instanciar um carro de corrida:

```
$formula = new carroDeCorrida("Puma","GTB");
$formula->trocaPneu("Slick");
$formula->estado();
```

O resultado do trecho acima será:

```
automóvel desligado
Puma
GTB
Pneus Slick
Puma GTB destruído
```

Glossário de funções do PHP

Algumas das principais funções disponibilizadas pelo PHP:

Datas

Date()

Retorna uma data no formato especificado

String date(string formato,int[datahora]);

Argumento	Descrição
Data	Data/hora a ser formatada. Se não for especificada utilizará a data/hora corrente
formato	String com caracteres de formatação de datas

Caractere	Significado
a	"am" ou "pm"
A	"AM" ou "PM"
d	Dia em formato numérico. De "01" a "31".
D	Dia da semana, textual, 3 letras. Ex.: "Fri"
F	Mês, textual, formato longo. Ex.: "January".
h	Hora no formato de 12 horas. De "01" a "12".
H	Hora no formato de 24 horas. De "00" a "23".
g	Hora no formato de 12 horas, sem os zeros à esquerda. De "1" a "12".
G	Hora no formato de 24 horas, sem os zeros à esquerda. De "0" a "23".
i	Minutos. De "00" a "59".
j	Dia do mês sem os zeros à esquerda. De "1" a "31".
l	Dia da semana, textual, longo. Ex.: "Friday".
L	Booleano que indica se o ano é bissexto."0" ou "1".
m	Mês. De "01" a "12".
n	Mês sem zeros à esquerda. De "1" a "12".
M	Mês, textual, 3 letras. Ex.: "Jan".
s	Segundos. De "00" a "59".
S	Sufixo ordinal em inglês, textual, 2 caracteres. Ex.: "th","nd".
t	Número de dias do mês determinado. Ex.:"28" a "31".

U Número de segundos desde o "epoch".
w Dia da semana, numérico. De "0"(domingo) a "6" (sábado) .
Y Ano com 4 dígitos. Ex.: "2002".
y Ano com 2 dígitos. Ex.: "02".
z Dia do ano. De "0" a "365".
Z Fuso horário em segundos. De "-43200" a "43200".

Getdate

Retorna um array associativo contendo todas as informações de uma data/hora específica.
Array getdate(int data/hora);
Índices da matriz retornada.

"seconds"	Segundos
"minutes"	Minutos
"hours"	Horas
"mday"	Dia do mês
"wday"	Dia da semana no formato decimal
"mon"	Mês no formato decimal
"year"	Ano no formato decimal
"yday"	Dia do ano no formato decimal
"weekday"	Dia da semana no formato decimal
"month"	Mês no formato texto

Gmmktime()

Retorna a data/hora GMT no formato UNIX. É idêntica à função mktime(), exceto que a data passada no parâmetro representa uma data GMT.
Int gmmktime(int hora, int minuto, int segundo, int mês, int dia, int ano, int [dsf]);

Gmstrftime()

Retorna uma data/hora GMT/CUT no formato especificado. É semelhante à função strftime(), exceto que a hora retornada é a hora de Greenwich.
String gmstrftime(string formato, int datahora);

Microtime()

Retorna uma string contendo a data/hora atual no formato "msec sec" do UNIX, que

significa o número de milisegundos e depois os segundos desde o UNIX Epoch(01/01/1970 00:00:00 GMT).

String microtime(void);

Mktime()

Retorna uma data/hora no formato UNIX

Int mktime(int hora, int minuto, int segundo, int mês, ine dia, int ano, int [dsf]);

Strftime()

Retorna uma data no formato especificado. Os nomes por extenso dos mesese dos dias da semana dependem da configuração feita com a função setlocale().

String strftime(string formato, int datahora);

Argumento Descrição

Data Data/hora a ser formatada. Se não especificada, utilizará a data/hora corrente

Caractere Significado

%a Dia da semana abreviado

%A Dia da semana completo

%b Nome abreviado do mês

%B Nome completo do mês

%c Representação preferida de data e hora

%d Dia do mês no formato decimal (00-31)

%H Hora no formato decimal de 24 horas (00-23)

%l Hora no formato decimal de 12 horas (00-12)

%j Dia do ano no formato decimal (001-366)

%m Mês em formato decimal (1-12)

%M Minutos no formato decimal

%p 'am' ou 'pm', dependendo da hora especificada

%S Segundos no formato decimal

%U Número da semana do ano atual no formato decimal, começando no primeiro domingo como o primeiro dia da primeira semana

%W Número da semana do ano atual no formato decimal, começando na primeira segunda-feira como o primeiro dia da primeira semana

%w Dia da semana no formato decimal, sendo o domingo igual a 0.

%x Representação preferida da data, sem a hora
%X Representação preferida da hora, sem a data
%y Ano no formato decimal, sem o século (00-99)
%Y Ano no formato decimal, com o século.
%Z Zona de fuso horário, ou nome, ou abreviação.
%% Uma caractere literal "%".

Time()

Retorna a data/hora no formato UNIX, que corresponde ao número de segundos desde o Unix Epoch(01\01\1970 00:00:00 GMT)
Int time(void);

Diretórios

Chdir()

Altera o diretório correspondente do PHP para o diretório especificado. Retorna true se tiver sucesso; caso contrário retorna false
Int chdir(string diretório);

Closedir()

Encerra a associação de um identificador(handle) com um diretório.
Void closedir(int handle_dir) ;

Opendir()

Retorna um handle de diretório para ser usado nas funções closedir(), readdir() e

Readdir()

Retorna o próximo nome de arquivo do diretório, na ordem em que estiverem armazenadas
String readdir(int handle_dir);

Execução de Programas

Escapeshellcmd()

Retira quaisquer caracteres de um string que poderiam ser utilizados para enganar um comando Shell para executar comandos arbitrários. Esta função normalmente é usada para

verificar os dados fornecidos pelo usuário antes de serem passados para as funções exec() ou system().

String escapeshellcmd(string comando);

Exec()

Executa um comando externo e mostra a última linha do resultado do comando.

String exec(string comando, string [array], int [variável_ref]) ;

Argumento	Descrição
Comando	Comando externo a ser executado
Array	Array contendo as linhas do resultado
Variável_ref	Variável que conterá o código de retorno do comando executado

Passthru()

Executa um comando externo e mostra todos os resultados.

String passthru(string comando, int [variável_ref]);

Argumento	Descrição
Comando	Comando externo a ser executado
Variável_ref	Variável que conterá o código de retorno do comando executado

System()

Executa um comando externo e mostra os resultados.

String system(string comando, int [variável_ref]);

Argumento	Descrição
Comando	Comando externo a ser executado
Variável_ref	Variável que conterá o código de retorno do comando executado

Sistema de arquivos do servidor

Basename()

Retorna o nome do arquivo em um path. No Windows, ambos os caracteres, / e \, são usados como separadores de diretórios. Em outros ambientes é utilizado somente o caractere /.

String basename(string path);

Chgrp()

Muda o grupo ao qual um arquivo pertence em ambientes UNIX. Em ambientes Windows não faz nada, retornando sempre true.

chgrp(string nome_do_arquivo, mixed grupo) ;

Chmod()

Altera as permissões de um arquivo em ambientes UNIX. Retorna true em caso de sucesso; caso contrário, retorna false. Em ambientes Windows esta função não faz nada, sempre retornando true

Int chmod(string nome_do_arquivo, int modo) ;

Chown()

Altera o proprietário de um arquivo em ambientes UNIX. Somente o root pode alterar o proprietário de um arquivo. Retorna true em caso de sucesso; caso contrário, retorna false. No Windows esta função na faz nada sempre retornando true.

Int chown(string nome_arquivo, mixed proprietário);

Copy()

Copia um arquivo. Retorna true se tiver sucesso; caso contrário, retorna false.

Int copy(string nome_arquivo, string arquivo_destino);

Delete()

Apaga um arquivo do servidor

Void delete(string nome_arquivo) ;

Dirname()

Retorna o nome de um diretório em um path. Veja a função basename().

String dirname(string path);

Fclose()

Fecha um arquivo. Retorna true se tiver sucesso; caso contrário, retorna false.

Int fclose(int fp);

Feof()

Retorna true caso o ponteiro de arquivo esteja no fim do arquivo ou ocorra um erro; caso

contrário, retorna false.

Int feof(fp);

File()

Lê um arquivo inteiro e coloca seu conteúdo em um array. Cada elemento do array corresponderá a cada linha do arquivo lido, sem tirar qualquer informação.

Array file(string nome_arquivo);

File_exists()

Retorna true se um determinado arquivo existe; caso contrário, retorna false. Os resultados desta função são armazenados e reutilizados. Para atualizar esta informação utilize a função clearstatcache().

Int file_exists(string nome_arquivo);

Filesize()

Retorna o tamanho de um arquivo, ou false em caso de erro. Os resultados desta função são

armazenadas e reutilizadas. Para utilizar esta informação utilize a função clearstatcache().

Int filesize(string nome_arquivo);

Filetype()

Retorna o tipo de um arquivo. Os valores possíveis são fifo, char, dir, block, link, file e unknown. Retorna false em caso de erro. Os resultados desta função são armazenadas e reutilizadas. Para utilizar esta informação utilize a função clearstatcache().

String filetype(string nome_arquivo);

Fopen()

Abre um arquivo ou uma URL. A função retornará false se ocorrer erro.

Int fopen(string nome_arquivo, string modo);

Argumento Descrição

Nome_arquivo Nome do arquivo. A ação a ser tomada dependerá de como o nome é iniciado. Veja a baixo as opções.

"http://" Abre um conexão HTTP 1.0 com o servidor especificado e um ponteiro de arquivo é retornado ao início do texto da resposta

"ftp://" Abre um conexão FTP com o servidor especificado e um ponteiro de

arquivo é retornado. O servidor deve suportar o modo passivo de FTP. É possível abrir os arquivos para leitura ou para escrita, mas não ambos ao mesmo tempo

outro Se o nome começar com qualquer outra coisa, o arquivo será aberto no sistema de arquivos e um ponteiro de arquivo será retornado.

Modo Modo de abertura. Adicionalmente pode-se colocar a letra "b" no parâmetro modo, informando que a ser processado é um arquivo binário

"r" Somente leitura, a partir do inicio do arquivo.

"r+" Leitura e escrita, a partir do inicio do arquivo.

"w" Somente escrita. A partir do inicio do arquivo e apagando todo o conteúdo do arquivo. Se o arquivo não existir, a função tentará criá-lo.

"w+" Para leitura e escrita. A partir do inicio do arquivo e apagando todo o conteúdo do arquivo. Se o arquivo não existir, a função tentará criá-lo.

"a" Somente escrita. A partir do inicio do arquivo. Se o arquivo não existir, a função tentará criá-lo.

"a+" Para leitura e escrita. A partir do inicio do arquivo. Se o arquivo não existir, a função tentará criá-lo.

Fputs()

Escreve o conteúdo de uma string em um arquivo

Int fputs(int fp, string str, int [tamanho]);

Fread()

Lê bytes de um arquivo

String fread(int fp, int [tamanho]);

Fwrite()

Escreve um número especifico de bytes de uma string em um arquivo

Int fwrite(int fp, string str, int [numbytes]);

Readfile()

Lê um arquivo e o envia para a saída padrão do sistema; retorna o número de bytes lidos

Int readfile(string nome_arquivo);

Rename()

Altera o nome de um arquivo. Retorna true se tiver sucesso; caso contrário false

Int rename(string nome_anterior, string nome_novo);

Matemática

Abs()

Retorna o valor absoluto de um número.

Mixed abs(mixed número);

Base_convert()

Retorna uma string com um número convertido para outra base numérica.

String base_convert(string número, int base_ant, int nova_base);

Max()

Retorna o maior dentre os especificados

Mixed max(mixed arg1, mixed arg2, mixed argn);

Min()

Retorna o menor dentre os especificados

Mixed min(mixed arg1, mixed arg2, mixed argn);

Mt_rand()

Gera um número aleatório mais confiável no intervalo especificado

Int mt_rand(int [limite_inf], int [limite_sup]);

Pi()

Retorna o valor da constante pi

Double pi(void);

Pow()

Retorna o resultado de uma base elevada a um expoente

Float pow(float base, float expoente);

Rand()

Retorna um número aleatório dentro de um intervalo especificado

Int rand(int [limit_inf], int [limi_sup]);

Round()

Retorna o valor de um número arredondado para o número inteiro mais próximo. Se o número estiver exatamente entre dois números inteiros, o resultado será sempre o número inteiro par

Double rand(double número);

Sin()

Retorna o seno de um ângulo em radianos

Float sin(float arg);

Sqrt()

Retorna a raiz quadrada

Float sqrt(float arg);

Tan()

Retorna a tangente de um ângulo em radianos

Float tan(float arg);

Funções diversas

Eval()

Processa uma string como código PHP

Void eval(string string_código);

Die()

Imprime uma mensagem e termina o script PHP

Void die(string mensagem);

Exit()

Termina o script PHP atual

Void exit(void);

Mail()

Envia um email para um ou mais destinatários

Bool mail(string dest, string assunto, string mensagem, string [headers_adic]);

Argumento	Descrição
Dest	Endereço de email dos destinatários(separados por ",")
Assunto	Assunto do email
Mensagem	Conteúdo do email
Headers_adic	Especifica headers adicionais que devem ser inseridas no fim do header padrão. Múltiplos headers podem ser especificados e devem ser separados por newline("\n")

Exemplo:

```
$destino = "NomeDestino <endereço@dominio.com.br>";
$remetente = "NomeRemetente <remetente@dominio.com.br>";
$assunto = "assunto do email";
mail($destino,"$assunto", $mensagem, "From: $remetente \n" );
```

Pack()

compacta dados em uma string binária

string pack(string formato, mixed [args]...) ;

Sleep()

Atrasa a execução por um tempo determinado (em segundos)

Void sleep(int num_segundos);

Tratamento de sessões

Session_decode()

Decodifica os dados de uma sessão em uma string

Bool session_decode(string dados);

Session_destroy()

Destrói os dados registrados associados à sessão atual

Bool session_destroy(void);

Session_encode()

Codifica os dados de uma sessão

Bool session_encode(void) ;

Session_start()

Inicializa os dados de uma sessão

Bool session_start(void);

Session_is_registered()

Descobre se uma variável foi registrada na sessão

Bool session_is_registered(string nome);

Session_module_name()

Retorna ou muda o nome do módulo da sessão atual

String session_module_name(string [módulo]);

Session_name()

Retorna ou muda o nome da sessão atual

String session_name(string [nome]);

Session_register()

Registra uma variável com a sessão atual

Bool session_register(string nome);

Session_unregister()

Descarta uma variável da sessão atual

Bool session_unregister(string nome);

Strings

Addslashes()

Coloca barras invertidas antes dos caracteres especiais: apostrofo(‘), aspas(“), barra invertida (\) e NUL

String addslashes(string str);

Chop()

Remove espaços em branco em seqüência

String chop(string str);

Crypt()

Retorna uma string criptografada através do modo DES.

String crypt(string str, string [salf]);

Echo()

Imprime uma ou mais strings

Echo (string arg1, string argn....);

Explode()

Retorna um array contendo as partes da string com valores separados por um separador

Array explode(string separador, string str) ;

Htmlentities()

Converte todos os caracteres aplicáveis em tags HTML

String htmlentities(string str);

Htmlspecialchars()

Converte caracteres especiais em tags HTML

String htmlspecialchars(string str);

Caracter Descrição

'&' '&'

'"' '"'

'<' '<'

'>' '>'

Implode()

Retorna uma string contendo a representação string de todos os elementos de um array separados pelo argumento glue

String implode(string glue, array fatias);

Ltrim()

Retorna uma string sem os espaçosiniciais

String ltrim(string str);

Md5()

Retorna o hash MD5 de uma string

String md5(string str);

Parse_str()

Converte uma string de parâmetros no formato das URLs vindas de formulários HTML em variáveis

Void parse_str(string str);

Exemplo:

$str = "form=firm&operation=insert&index=0";

parse_str($str);

echo $form; //imprime "firm"

echo $operation; //imprime "insert"

echo $index; //imprime "0"

Strcmp()

Compara duas strings, retornando um valor: < 0 se str1 for menor que str2; > 0 se str1 for maior que str2, e 0 se elas forem iguais

Int strcmp(string str1, string str2);

Strip_tags()

Retorna uma string sem as tags HTML e PHP

String strp_tags(string str);

Stripslashes()

Apaga as barras invertidas de caracteres específicos

String stripslashes(string str);

Strlen()

Retorna o comprimento de uma string

Int strlen(string str);

Strrev()

Inverte uma string

String strrev(string str);

Strtolower()

Transforma as letras em uma string para minúscula.

String strtolower(string str) ;

Strtoupper()

Transforma as letras em uma string para maiúscula.

String strtoupper(string str) ;

Str_replace()

Substitui todas as ocorrências de uma substring por outra

String str_replace(string str_ant, string novo_str, string str) ;

Argumento	Descrição
Str_ant	Substring a ser substituída
Novo_str	Substring que substituirá a anterior
Str	String original

Trim()

Apaga os espaços em branco do inicio e fim de uma string.

String trim(string str);

Ucfirst()

Transforma o primeiro caractere de uma string em maiúsculo

String ucfirst(string str) ;

Ucwords()

Transforma o primeiro caractere de cada palavra de uma string em maiúsculo

String ucwords(string str);

Variáveis

Doubleval()

Retorna o valor em ponto flutuante de uma variável

Double doubleval(mixed var) ;

Empty()

Retorna false se var estiver atribuída; caso contrário retorna true

Int empty(mixed var) ;

Gettype()

Retorna o tipo de uma variável

String gettype(mixed var);

Intval()

Retorna o valor em inteiros de uma variável, utilizando uma base especificada. O padrão da base é 10

Int intval(mixed var, int [base])

Is_array()

Retorna true se a variável for do tipo array

Int is_array(mixed var);

Is_double()

Retorna true se a variável for do tipo double

Int is_double(mixed var);

Is_float()

Retorna true se a variável for do tipo float

Int is_float(mixed var);

Is_int()

Retorna true se a variável for do tipo inteiro

Int is_int(mixed var);

Is_object()

Retorna true se a variável for um objeto

Int is_object(mixed var);

Is_real()

Retorna true se a variável for do tipo real

Int is_real(mixed var);

Is_string()

Retorna true se a variável for do tipo string

Int is_string(mixed var);

Isset()

Retorna true se uma variável existir

Int isset(mixed var);

Settype()

Altera o tipo de uma variável. Retorna true se tiver sucesso; caso contrário, retorna false

Int settype(string var, string tipo);

Tipos permitidos

Integer

Double

String

Array

Objeto

Strval()

Retorna o valor em string de uma variável

String strval(mixed var);

Unset

Exclui uma variável

Int unset(mixed var);

Referências Bibliográficas

- PHP para quem conhece PHP – Recursos avançados para a criação de websites dinâmicos. Autor: Juliano Niederauer Editora: Novatec Ltda www.novateceditora.com.br
- Apostila de PHP. Autor: Bruno Rodrigues Siqueira bruno@netfly.com.br
- Programação para Web utilizando PHP. Autores: Alexandre Arroyo e Fabio Santos. Divisão de serviços à comunidade. Centro de Computação – Unicamp.
- Programando para Web com PHP/MySQL. Autor: Fred Cox Junior
- Curso de PHP e MySQL. Autores: Flávio S. Gonzaga (bim@inf.ufsc.br) e Guilherme Birckan (birckan@inf.ufsc.br)
- PHP e MySQL – Desenvolvimento Web. Autores: Luke Welling e Laura Thomson. Segunda edição - 2003. Editora: Campus