MINICURSO

Aplicações WEB com PHP

Prof. Ms. Sérgio Carlos Portari Júnior

IX SASI – Semana Acadêmica de Sistemas de Informação

UEMG – Universidade do Estado de Minas Gerais

Campus de Frutal

11 / 2013

PARTE I

A linguagem **PHP** surgiu por volta de 1994, como um pacote de programas CGI criados por **Rasmus Lerdorf**, com o nome **P**ersonal **H**ome **P**age **T**ools, para substituir um conjunto de scripts em Perl que usava no desenvolvimento de sua página pessoal. Em 1997 foi lançado o novo pacote da linguagem com o nome **PHP/FI**, trazendo a ferramenta Forms Interpreter, um interpretador de comandos SQL. Mais tarde, **Zeev Suraski** desenvolveu o analisador do **PHP 3** que contava com o primeiro recurso de orientação a objetos, que dava poder de alcançar alguns pacotes, já tinha herança e dava aos desenvolvedores apenas a possibilidade de implementar propriedades e métodos. Logo após, Zeev e Andi Gutmans, escreveram o **PHP 4**, abandonando por completo o PHP 3, dando assim bem mais poder a linguagem com maior número de recursos a orientação a objetos. O grande problema do PHP 4 foi a criação de cópias de objetos, pois a linguagem ainda não trabalhava com apontadores ou handlers, o que não ocorre por exemplo com as linguagens **Java** e **Ruby**. Logo após o problema foi resolvido para atual versão do **PHP 5**, que passou a trabalhar com handlers, funcionando da seguinte forma caso copie um objeto, copiaremos seu apontador, pois, caso haja alguma mudança na versão original do objeto, todas as outras também sofrem a alteração, o que não acontecia anteriormente com o PHP 4.

Trata-se de uma linguagem extremamente modularizada, o que a torna ideal para instalação e uso em servidores web. Diversos módulos são criados no repositório de extensões PECL (PHP Extension Community Library) e alguns destes módulos são introduzidos como padrão em novas versões da linguagem. É muito parecida, em tipos de dados, sintaxe e mesmo funções, com a linguagem C e com a C++. Pode ser, dependendo da configuração do servidor, embarcada no código HTML. Existem versões do PHP disponíveis para os seguintes sistemas operacionais: Windows, Linux, FreeBSD, Mac OS, OS/2, AS/400, Novell Netware, RISC OS, AIX, IRIX e Solaris.

O grande trunfo da linguagem é sua popularidade e preço da manutenção, já que a grande maioria dos **Hosts** (Serviços de Hospedagem de Sites) oferecem planos de hospedagem a preços bem baixos ou até mesmo gratuita. Construir páginas dinâmicas em PHP é muito fácil, sem falar no grande número de bancos de dados compatíveis com o PHP: *Oracle, Sybase, PostgreSQL, InterBase, MySQL, SQLite, MSSQL, Firebird, etc.*, destaque para o **MySQL** que faz um casamento perfeito com o PHP a anos.

PHP oferece vasto suporte aos protocolos: IMAP, SNMP, NNTP, POP3, HTTP, LDAP, XML-RPC, SOAP. É possível abrir sockets e interagir com outros protocolos. E as bibliotecas de terceiros expandem ainda mais estas funcionalidades. Existem iniciativas para utilizar o PHP como linguagem de programação de sistemas fixos. A mais notável é a PHP-GTK (http://www.php-gtk.com.br/home). Trata-se de um conjunto do PHP com a biblioteca GTK, portada do C++, fazendo assim softwares inter-operacionais entre Windows e Linux. Infelizmente esta extensão tem sido pouco usada em projetos atuais.

Principais Características do PHP

PHP é uma linguagem de programação de domínio específico, ou seja, seu escopo se estende a um campo de atuação que é o desenvolvimento web, embora tenha variantes como o PHP-GTK. Seu propósito principal é de implementar soluções web velozes, simples e eficientes. Suas Características são:

- Velocidade e robustez
- Estruturado e orientação a objetos
- Portabilidade – independência de plataforma
- Tipagem dinâmica (não precisa declarar o tipo da variável)
- Sintaxe similar a C/C++ e o Perl
- Open-source

Preparando nosso ambiente

Atualmente o PHP encontra-se na versão 5. Para nosso curso, utilizaremos a versão 5.3.0 embutida no WAMPSERVER 2.x (http://www.wampserver.com/en/#download-wrapper) , que também instala para utilização o Apache Web Server 2, o MySQL 5 e o PHPMyAdmin 3.

O primeiro passo para iniciarmos a disciplina é o download, instalação e configuração do WampServer 2.0i, que é totalmente gratuito para estudos e pode ser adquirido no endereço http://www.wampserver.com/en/download.php. Na versão atual possui 16Mb.

Após baixar o arquivo, basta executá-lo e seguir as simples instruções na tela, apertando Next. Após finalizar a instalação, é possível trocar a linguagem padrão para Português para facilitar a utilização das ferramentas.

Durante a instalação, será possível escolher o browser padrão para exibição das páginas.

O Wamp ficará na bandeja, ao lado do relógio e com o botão direito do mouse pode ser acionado para configurações de Idioma (Language). Com o botão esquerdo é possível acessar cada aplicação separadamente, bem com iniciar ou interromper qualquer um dos serviços ou todos simultaneamente. Para iniciar o WampServer corretamente, clique com o botão esquerdo do mouse e depois em Iniciar todos os serviços.

Para testar se a aplicação está corretamente, abra um navegador de internet e digite HTTP://localhost no endereço e clique enter. Se tudo estiver correto, você verá a tela de informações do WampServer, bem como a exibição das ferramentas e versões ativas, em os atalhos para o phpinfo() e phpmyadmin.

O diretório padrão do WampServer fica na raiz da unidade c:\ da máquina. Suas páginas ocuparão a pasta www dentro da pasta wamp. Você pode criar novas pastas dentro de www para separar seus projetos, e elas serão incluídas automaticamente na página de boas vindas de HTTP://localhost.

É possível criar alias (apelidos / atalhos) para que seus projetos fiquem em outras pastas que não a www. Com o botão esquerdo do mouse, clique no ícone do Wamp na bandeja, selecione no menu Apache a opção Diretórios de Alias e Adicionar e siga as instruções na tela. A pasta deve primeiramente ter sido criada para criarmos o alias, que poderá estar em qualquer local.

Quando você configurar toda e estrutura poderá colocar seu servidor on-line para a rede ou internet. Selecione com o esquerdo do mouse a opção Colocar On-line. Caso esteje em uma rede, basta divulgar agora seu endereço IP para que todos possam acessar suas páginas. Lembre que caso faça isso na Internet você deve remover ou renomear o arquivo índex.php e colocar sua página inicial com esse nome.

Para edição das páginas, você pode utilizar qualquer editor de texto (padrão ANSI), como o wordpad, bloco de notas, etc. Remcomendo utilizar o Notepad++ (http://notepad-plus-plus.org/), ferramenta gratuita, leve e que realiza diversas funções (como colorir o texto de acordo com comandos/tipos de dados, endentação automática, etc).

1 – Extensão de arquivos PHP

.php – Arquivo PHP contendo um programa.
.class.php – Arquivo PHP contendo uma classe.
.inc.php – Arquivo PHP contendo constantes ou configurações.

2 – Delimitadores do código PHP

Todo código escrito em PHP dever estar contido dentro dos seguintes delimitadores: **<?php** e **?>** ou**<?** e **?>** (se o o arquivo php.ini estiver configurado para isto. Não é o padrão).

Exs.:

```
<?php

echo “Sérgio”;

?>
```

saída: **Sérgio**

_____-

```
<?php

print (‘123456’);

?>
```

saída: **123456**

_____

```
<?php
$vetor = array( 'uva', 'pera' );
print_r($vetor);
?>
```

saída: **Array(**

[0]=>Uva

[1]=>Pera

)

2.1. Criando o primeiro script

Primeiro Exemplo

Neste exemplo, criaremos um script com uma saída simples, que servirá para testar se a instalação foi feita corretamente:

```
<html>
<head><title>Aprendendo PHP</title></head>
<body>

<?php
echo "Primeiro Script";
?>

</body>
</html>
```

Salve o arquivo como "primeiro.php" na pasta de documentos do Apache (c:\wamp\www\). Abra uma janela do navegador e digite o endereço "http://localhost/primeiro.php". Se você abrir o arquivo dando dois cliques ao invés de abri-lo no navegador, o PHP não será interpretado.

Verificando o código fonte da página exibida, temos o seguinte:

```
<html>
<head><title>Aprendendo PHP</title></head>
<body>

Primeiro Script

</body>
</html>
```

Isso mostra como o PHP funciona. O script é executado no servidor, ficando disponível para o usuário apenas o resultado.

3 – Variáveis PHP

Nomes de variáveis

Variáveis são endereços de memória nos quais podemos armazenar dados ou informações. Usamos variáveis para manipular esses dados mais facilmente e também para não perdê-los no meio do processo.

Toda variável em PHP tem seu nome composto pelo caracter **$**. PHP é case sensitive, ou seja, as variáveis $vida e $VIDA são diferentes. Por isso é preciso ter muito cuidado ao definir os nomes das variáveis. É bom evitar os nomes em maiúsculas, pois como veremos mais adiante, o PHP já possui alguma variáveis pré-definidas cujos nomes são formados por letras maiúsculas.

As variáveis em **PHP** são peculiares por nos possibilitar **guardar qualquer tipo de dado** em uma variável, desde um simples caracter até um número decimal de grande precisão.

Ex.:
```
<?php

$nome = “Ana”;

$idade = “21”;

echo “$nome, $idade”;

?>
```

saída: **Ana, 21**

Agora vamos escrever um script que produza exatamente o mesmo resultado do primeiro script desenvolvido no item 2.1 com a utilização de uma variável:

```
<html>
<head><title>Aprendendo PHP</title></head>
<body>

<?php
$texto = "Primeiro Script";
echo $texto;
?>

</body>
```

```
</html>
```

3.1 – Tipos de variáveis

- String

Uma string é um conjunto de caracteres de qualquer tipo. É o "vale tudo" da programação. Qualquer coisa entra ali. Pode colocar letra, número, símbolo, enfim, aceita tudo. Por exemplo, no campo onde colocamos o tutorial para publicação temos como variável de recepção (aquela que recebe o conteúdo que nós mandamos) uma string.

```
$carro = "meu carro é fiat";
$dog = "meu cão é um pitbull";
```

- Inteiro

Um inteiro é essencialmente um número, pode ser positivo ou negativo. Inteiro é representado pelo tipo "integer" em php (e na grande maioria das linguagens). Um integer tem 8 bits, portanto tme um limite mínimo e um máximo. Mas, para as coisas mais triviais integer dá conta do recado.

```
$a = 11;
$b = 250;
```

- Float ou dooble

Este é um outro tipo de variável que só aceita número. Mas, diferentemente do tipo inteiro, aqui podemos colocar números com casas decimais. Como em integer, este tipo também tem limites, mas vai ser bem complicado de você chegar neles. Ele pode suportar até 14 casasdecimais. Um integer suporta 5 dígitos no máximo. Num site comum dificilmente usa-se float. Você só vai usar isso em coisas mais elaboradas como sistemas de controle de estoque por exemplo.

```
$x = 3.500;
$y = 23.900;
```

- Booleano

Um valor booleano é a síntese do sistema binário, onde baseamos a informática. O sistema binário consiste em representar tudo em apenas duas formas: 0=desligado e 1=ligado. Tudo em informática basea-se nisso. Até aquele vídeo que você assistiu no youtube agora pouco, internamente, é composto de sequências de zeros e uns. Pois bem, o booleano é precisamente essa representação. Ele serve para determinar se algo é verdadeiro(TRUE) ou falso(FALSE). 0 para falso e 1 para verdadeiro. Diversas funções do php retornam booleanos. Eles normalmente são usados nos verificadores de condição if/else.

```
<?php
$exibir = TRUE; // declara variável com valor TRUE
// testa se $exibir é TRUE
if ($exibir)
{
```

```
echo “É verdadeiro!”;
}
?>
```
saída: É verdadeiro!

- Array

Array é um tipo de variável largamente usado. Consiste basicamente num conjuntos de variáveis com um indexador e um valor. São pares, chaves ou indexadores e valor. Funciona como um índice de um livro: para cada página listada no índice temos um capítulo. Um array é desse jeito. Existem dois tipos de arrays: array unidimensional e array multidimensional.

Um array multidimensional é um array que contem outros arrays. Agora é como se você estivesse numa biblioteca. Imagine cada estante como sendo um array, daí cada prateleira como sendo um outro array dentro do array estante, e em cada prateleira os livros são arrays do array prateleira. Os capítulos do livro são os elementos do array livro. Para denominar as chaves ou indexadores de um array podemos usar basicamente strings. Números também são usados. Várias funções retornam arrays então é bom saber trabalhar com eles. Eles facilitam nossa vida consideravelmente em várias aplicações.

```
<?php
$carros = array(“BMW”,”Ferrari”,”Volvo”);
echo $carros[1];
?>
```
saída: Ferrari

- Objeto

Objeto é um tipo de variável exclusivo de programação orientada a objeto. É algo parecido com um array. Só que um objeto é composto de métodos e propriedades. Esses métodos e propriedades são determinados em classes. Um objeto é a instância de uma classe, ou seja, é a classe pronta para ser usada. Mas classes e objetos são assunto para outro tutorial. Várias funções de php retornam objetos contendo somente propriedades.

```
<?php
class Computador
{
var $cpu;
function ligar()
{
echo “Funcionou meu Computador a {$this->cpu}...”;
}
}
$obj = new Computador;
$obj -> cpu = “500Mhz”;
$obj ->ligar();
?>
```
saída: Funcionou meu Computador a 500Mhz...

4 – Operadores em PHP

Um operador é utilizado para realizar operações entre um ou mais valores (ou expressões, no jargão de programação) e retornar apenas um valor final. Vamos agora aos operadores.

Operadores aritméticos no PHP

Neste grupo de operadores estão os operadores de operações matemáticas básicas como adição, subtração, multiplicação e divisão. Veja a tabela com os operadores:

Operador	Função	Exemplo
`+`	Adição	`$a + $b`
`-`	Subtração	`$a - $b`
`*`	Multiplicação	`$a * $b`
`/`	Divisão	`$a / $b`
`%`	Módulo (resto da divisão)	`$a % $b`

Vejam alguns exemplos com os operadores aritméticos.

```
<?php
// Declarando os valores das variáveis
$a = 4;
$b = 2;
?>
<h2>Adição</h2>
<p>
<?php
echo $a + $b;
?>
</p>
<h2>Subtração</h2>
<p>
<?php
echo $a – $b;
?>
</p>
<h2>Multiplicação</h2>
<p>
<?php
echo $a * $b;
?>
</p>
<h2>Divisão</h2>
<p>
<?php
```

```
echo $a / $b;
?>
</p>
<h2>Módulo(resto da divisão)</h2>
<p>
<?php
echo $a % $b;
?> </p>
```

Operadores de Atribuição no PHP

O operador de atribuição é utilizado quando queremos atribuir o valor da expressão que esta a direita ao operando (normalmente uma variável) que esta a sua esquerda (quase como se fosse um "igual a", como por exemplo x é igual a 1 e y igual a 2).

O operador mais básico de atribuição você já deve estar imaginando é o sinal de igual (=) em que usamos para atribuir todas as nossas variáveis anteriormente. Veja a seguir o uso do operador de atribuição igual e um truque que podemos fazer utilizando **operadores aritméticos** e **expressões**.

```
<?php
// Declarando o valor de a com uma simples expressão
$a = 5 + 3;
echo $a; //imprime 8
?>
<br />
<?php
// Atribuimos o valor de 2 a variavel $c e somamos com
// 5 para ser o valor de b
$b = ( $c = 2 ) + 5;
echo $b; //imprime 7
?>
<br />
<?php
echo $c; //imprime 2
?>
```

Observe que conseguimos declarar o valor de $a uma simples expressão aritmética, lembrando que você pode utilizar todos os operadores aritméticos que aprendemos anteriormente em sua expressão.

Em seguida criamos a variável $b e $c, sendo $c criada dentro da atribuição de $b sendo assim $b equivale a $c, que equivale a 2, mais 5.

Além do formato de atribuição que aprendemos podemos combinar o operador de atribuição a todos os operadores aritméticos e de string observe a tabela a seguir:

Operador	Função	Exemplo	Equivalente a
`+=`	Atribuição e adição	`$a += $b`	`$a = $a + $b`

Operador	Função	Exemplo	Equivalente a
`-=`	Atribuição e Subtração	`$a -= $b`	`$a = $a - $b`
`*=`	Atribuição e Multiplicação	`$a *= $b`	`$a = $a * $b`
`/=`	Atribuição e Divisão	`$a /= $b`	`$a = $a / $b`
`%=`	Atribuição e Módulo	`$a %= $b`	`$a = $a % $b`
`=`	Atribuição e concatenação	`$a = $b`	`$a = $a . $b`

Veja agora um exemplo utilizando operadores de atribuição em conjunto com os operadores aritméticos.

```
<?php
// Declarando o valor de a
$a = 5;
// Atribuindo e somando 3 em $a
// ou seja $a = 5 + 3;
$a += 3;

echo $a //imprime 8

?>
```

Operadores de string no PHP

O único operador de string que possuímos no PHP é o operador de concatenação, além do que falamos anteriormente que é o de atribuição e concatenação, que é representado pelo . (ponto). O operador de concatenação tem por finalidade unir o conteúdo de duas strings, veja o exemplo:

```
<?php
// Declaro a variável $titulo
$titulo = "Operadores de string no PHP";
// Concateno $titulo a $texto
$texto = "Estou aprendendo sobre" . $titulo;

echo $texto;

echo "<br>" . PHP_EOL;

// Mesmo exemplo anterior porém utilizando atribuição
e concatenação
$texto = "Estou aprendendo sobre ";
$texto = $titulo;

echo $texto;
?>
```

Existe um conjunto de funções (assim como em C) para tratar strings já embutidos no PHP. Uma que será bastante utilizada é a nl2br().

nl2br()

string nl2br(string str);

Retorna a string fornecida substituindo todas as quebras de linha (“\n”) por quebras de linhas em html (“
”).

Exemplo:

echo nl2br(“Sérgio\nPortari\n”);

Imprime:

Sérgio
Portari

Operador de incremento/decremento no PHP

Os operadores de incremento e decremento são muito parecidos com os operadores aritméticos. Eles permitem que sejam feitas adições (incremento) e subtrações (decremento) direto na variável informada, mas sempre operações unitárias, isto é, soma se 1 ou subtrai se 1 da variável. Os operadores de incremento e decremento são respectivamente ++ e – - .

Existem duas formas de incremento/decremento: Pós e Pré.

Pós incremento/decremento no PHP

No pós incremento/decremento o PHP retorna o valor da variável para só depois então a incrementá-la/decrementá-la. Veja o exemplo:

```
<?php

$a = 10;
$b = $a++; // Primeiro $a é atribuido a $b para só então
ser incrementada
$a++;

echo “$a = ” . $a . ” | $b = ” . $b;

?>
```

Pré incremento/decremento no PHP

No pré incremento/decremento o PHP primeiro incrementa/decrementa a variável e depois retorna o seu valor. Observe alterando o exemplo anterior:

```
<?php

$a = 10;
$b = ++$a; // Primeiro $a é incrementada e só depois é
```

atribuido a $b
++$a;

echo "$a = " . $a . " | $b = " . $b;

?>

Veja a tabela para uma consulta caso tenha duvida nos operadores de **incremento** e **decremento**.

Operador	Nome	Função
`++$a`	Pré-incremento	Incrementa `$a` em um, e então retorna `$a`.
`$a++`	Pós-incremento	Retorna `$a`, e então incrementa `$a` em um.
`--$a`	Pré-decremento	Decrementa `$a` em um, e então retorna `$a`.
`$a--`	Pós-decremento	Retorna `$a`, e então decrementa `$a` em um.
`-$a`	Inverter o sinal	Inverte o sinal de `$a` e retorna `$a`.

Atribuição por referência (ponteiros)

Outro meio de atribuir valores a variáveis que o PHP nos fornece é a atribuição por referência. Ou seja, a nova variável será uma referência, um apelido, a variável original sendo assim qualquer alteração na variável original alterará a variável de referencia como qualquer alteração na variável de referência alterara a variável original. Observe:

```
<?php
// Criamos a variável $b
$b = "Estamos aprendendo sobre";
// Atribuimos por referência o valor de $b a $a,
// ou seja, ambas agora possuem o mesmo valor
$a = &$b;
// Altero o valor de $a concatenando o com uma string
// caso não tenha entendido a concatenação volte a operadores
de string
$a = "Atribuição por referência no PHP";
// Exibo $a e $b pulando linhas para melhor vizualização
echo $a;
echo "<br />" . PHP_EOL;
echo $b;
echo "<br />" . PHP_EOL;
// Alterei o valor de $b e como $b foi atribuido por referência a $a
// agora quando altero $b o valor de $a muda também
$b = "Alterei o valor de $b e sabia que o de $a muda também?";
echo $a;
echo "<br />" . PHP_EOL;
echo $b;
?>
```

5 – Classes no PHP

Conceito

Orientação a objetos é um termo que descreve uma série de técnicas para estruturar soluções para problemas computacionais. No nosso caso específico, vamos falar de programação OO, que é um paradigma de programação no qual um programa é estruturado em objetos, e que enfatiza os aspectos abstração, encapsulamento, polimorfismo e herança.

Objeto

Objetos são a unidade fundamental de qualquer sistema orientado a objetos. Tudo é um objeto — tipos, valores, classes, funções, métodos, pode ser uma pessoa, um lugar, um carro, um avião, e é claro, instâncias: todos possuem atributos e métodos associados. Características que definem um objeto são os atributos e comportamentos chamados de métodos.

Atributos seriam os aspectos de um objeto:

Objeto Carro:

· Marca
· Modelo
· Cor
· Ano

As ações de um objeto seriam os métodos:

Objeto Carro:

· Correr
· Freiar
· Bater
· Parar

Construtores
O Construtor é referenciado no PHP como __construct(), é uma função definida na classe e que é executada sempre que o objeto é criado, ou seja, sempre que a classe é instanciada.

Destrutores
São chamadas no momento em que o objeto está sendo destruído.Podem servir para fechar uma conexão com bando de dados, no PHP5 usamos a nomenclatura padrão __destruct() para implementarmos destrutores.

Visibilidade de Atributos e Métodos
Os métodos e atributos de uma classe no PHP podem ser definidos como: private, public, protected

Public
O atributo ou método definido como public torna-o acessível em qualquer lugar da classe, de suas subclasses ou em qualquer parte dos scritps que fazem parte da classe.

Protected

O atributo ou método definido como protected são visíveis pela classe que os criou e por suas subclasses, e não são acessíveis fora desse contexto.

Private

O atributo ou método definido como private são visíveis apenas na classe que os criou, ou seja, subclasses ou script que contêm a classe não podem acessar esses atributos ou métodos.

6 – Constantes

O PHP possui algumas constantes pré-definidas, indicando a versão do PHP, o Sistema Operacional do servidor, o arquivo em execução, e diversas outras informações. Para ter acesso a todas as constantes pré-definidas, pode-se utilizar a função phpinfo(), que exibe uma tabela contendo todas as constantes pré-definidas, assim como configurações da máquina, sistema operacional, servidor http e versão do PHP instalada.

Para definir constantes utiliza-se a função define. Uma vez definido, o valor de uma constante não poderá mais ser alterado. Uma constante só pode conter valores escalares, ou seja, não pode conter nem um array nem um objeto. A assinatura da função define é a seguinte:

int define(string nome_da_constante, mixed valor);

A função retorna true se for bem-sucedida. Veja um exemplo de sua utilização a seguir:

```
define ("pi", 3.1415926536);

$circunf = 2*pi*$raio;
```

7 – Comparações lógicas

As comparações são feitas entre os valores contidos nas variáveis, e não as referências.
Sempre retornam um valor booleano.

==	igual a
!=	diferente de
<	menor que
>	maior que
<=	menor ou igual a
>=	maior ou igual a

8 – Estruturas de Controle

Comandos de seleção

Também chamados de condicionais, os comandos de seleção permitem executar comandos ou blocos de comandos com base em testes feitos durante a execução.

if

O mais trivial dos comandos condicionais é o `if`. Ele testa a condição e executa o comando indicado se o resultado for `true` (valor diferente de zero). Ele possui duas sintaxes:

```
if (expressão)
   comando;
```

```
if (expressão):
   comando;
   . . .
   comando;
endif;
```

Para incluir mais de um comando no `if` da primeira sintaxe, é preciso utilizar um bloco, demarcado por chaves.

O `else` é um complemento opcional para o `if`. Se utilizado, o comando será executado se a expressão retornar o valor `false` (zero). Suas duas sintaxes são:

```
if (expressão)
   comando;
else
   comando;
```

```
if (expressão):
   comando;
   . . .
   comando;
else
   comando;
   . . .
   comando;
endif;
```

A seguir, temos um exemplo do comando `if` utilizado com `else`:

```
if ($a > $b)
  $maior = $a;
else
  $maior = $b;
```

O exemplo acima coloca em $maior o maior valor entre $a e $b

Em determinadas situações é necessário fazer mais de um teste, e executar condicionalmente diversos comandos ou blocos de comandos. Para facilitar o entendimento de uma estrutura do tipo:

```
if (expressao1)
   comando1;
else
  if (expressao2)
    comando2;
  else
    if (expressao3)
      comando3;
    else
      comando4;
```

foi criado o comando, também opcional `elseif`. Ele tem a mesma função de um `else` e um `if` usados sequencialmente, como no exemplo acima. Num mesmo `if` podem ser utilizados diversos `elseif`'s, ficando essa utilização a critério do programador, que deve zelar pela legibilidade de seu script.

O comando `elseif` também pode ser utilizado com dois tipos de sintaxe. Em resumo, a sintaxe geral do comando `if` fica das seguintes maneiras:

```
if (expressao1)
  comando;
[ elseif (expressao2)
   comando; ]
[ else
   comando; ]
if (expressao1) :
  comando;
   . . .
  comando;
[ elseif (expressao2)
  comando;
   . . .
  comando; ]
[ else
  comando;
   . . .
  comando; ]
endif;
```

switch

O comando `switch` atua de maneira semelhante a uma série de comandos `if` na mesma expressão. Frequentemente o programador pode querer comparar uma variável com diversos valores, e executar um código diferente a depender de qual valor é igual ao da variável. Quando isso for necessário, deve-se usar o comando `switch`. O exemplo seguinte mostra dois trechos de código que fazem a mesma coisa, sendo que o primeiro utiliza uma série de `if`'s e o segundo utiliza `switch`:

```
if ($i == 0)
   print "i é igual a zero";
elseif ($i == 1)
   print "i é igual a um";
elseif ($i == 2)
   print "i é igual a dois";

switch ($i) {
case 0:
     print "i é igual a zero";
     break;
case 1:
     print "i é igual a um";
     break;
case 2:
     print "i é igual a dois";
     break;
}
```

É importante compreender o funcionamento do `switch` para não cometer enganos. O comando `switch` testa linha a linha os cases encontrados, e a partir do momento que encontra um valor igual ao da variável testada, passa a executar todos os comandos seguintes, mesmo os que fazem parte de outro teste, até o fim do bloco. por isso usa-se o comando `break`, quebrando o fluxo e fazendo com que o código seja executado da maneira desejada. Veremos mais sobre o `break` mais adiante. Veja o exemplo:

```
switch ($i) {
case 0:
     print "i é igual a zero";
case 1:
     print "i é igual a um";
case 2:
     print "i é igual a dois";
}
```

No exemplo acima, se $i for igual a zero, os três comandos "print" serão executados. Se $i for igual a 1, os dois últimos "print" serão executados. O comando só funcionará da maneira desejada se $i for igual a 2.

Em outras linguagens que implementam o comando switch, ou similar, os valores a serem testados só podem ser do tipo inteiro. Em PHP é permitido usar valores do tipo string como elementos de teste do comando switch. O exemplo abaixo funciona perfeitamente:

```
        switch ($s) {
        case "casa":
             print "A casa é amarela";
        case "arvore":
             print "a árvore é bonita";
        case "lampada":
             print "joao apagou a lampada";
}
```

comandos de repetição

while

O `while` é o comando de repetição (laço) mais simples. Ele testa uma condição e executa um comando, ou um bloco de comandos, até que a condição testada seja falsa. Assim como o `if`, o `while` também possui duas sintaxes alternativas:

```
        while (<expressao>)
           <comando>;


        while (<expressao>):
           <comando>;
            . . .
           <comando>;
        endwhile;
```

A expressão só é testada a cada vez que o bloco de instruções termina, além do teste inicial. Se o valor da expressão passar a ser `false` no meio do bloco de instruções, a execução segue até o final do bloco. Se no teste inicial a condição for avaliada como `false`, o bloco de comandos não será executado.

O exemplo a seguir mostra o uso do `while` para imprimir os números de 1 a 10:

```
        $i = 1;
        while ($i <=10)
          print $i++;
```

do... while

O laço `do..while` funciona de maneira bastante semelhante ao `while`, com a simples diferença que a expressão é testada ao final do bloco de comandos. O laço `do..while` possui apenas uma sintaxe, que é a seguinte:

```
do {
   <comando>
   . . .
   <comando>
} while (<expressao>);
```

O exemplo utilizado para ilustrar o uso do `while` pode ser feito da seguinte maneira utilizando o `do.. while`:

```
$i = 0;
do {
  print ++$i;
} while ($i < 10);
```

for

O tipo de laço mais complexo é o `for`. Para os que programam em C, C++ ou Java, a assimilação do funcionamento do for é natural. Mas para aqueles que estão acostumados a linguagens como Pascal, há uma grande mudança para o uso do for. As duas sintaxes permitidas são:

```
for (<inicializacao>;<condicao>;<incremento>)
   <comando>;
```

```
for (<inicializacao>;<condicao>;<incremento>) :
   <comando>;
   . . .
   <comando>;
endfor;
```

As três expressões que ficam entre parênteses têm as seguintes finalidades:

Inicialização: comando ou sequencia de comandos a serem realizados antes do inicio do laço. Serve para inicializar variáveis.

Condição: Expressão booleana que define se os comandos que estão dentro do laço serão executados ou não. Enquanto a expressão for verdadeira (valor diferente de zero) os comandos serão executados.

Incremento: Comando executado ao final de cada execução do laço.

Um comando `for` funciona de maneira semelhante a um `while` escrito da seguinte forma:

```
<inicializacao>
while (<condicao>) {
comandos
...
<incremento>
}
```

Quebra de fluxo

Break

O comando `break` pode ser utilizado em laços de `do`, `for` e `while`, além do uso já visto no comando `switch`. Ao encontrar um `break` dentro de um desses laços, o interpretador PHP para imediatamente a execução do laço, seguindo normalmente o fluxo do script.

```
while ($x > 0) {
  ...
  if ($x == 20) {
    echo "erro! x = 20";
    break;
  ...
}
```

No trecho de código acima, o laço `while` tem uma condição para seu término normal ($x <= 0), mas foi utilizado o break para o caso de um término não previsto no início do laço. Assim o interpretador seguirá para o comando seguinte ao laço.

Continue

O comando `continue` também deve ser utilizado no interior de laços, e funciona de maneira semelhante ao `break`, com a diferença que o fluxo ao invés de sair do laço volta para o início dele. Vejamos o exemplo:

```
for ($i = 0; $i < 100; $i++) {
  if ($i % 2) continue;
  echo " $i ";
}
```

O exemplo acima é uma maneira ineficiente de imprimir os números pares entre 0 e 99. O que o laço faz é testar se o resto da divisão entre o número e 2 é 0. Se for diferente de zero (valor lógico `true`) o interpretador encontrará um `continue`, que faz com que os comandos seguintes do interior do laço sejam ignorados, seguindo para a próxima iteração.

PARTE II

9. Utilizando formulários HTML + PHP

Ao clicar num botão "Submit" em um formulário HTML as informações dos campos serão enviadas ao servidor especificado para que possa ser produzida uma resposta. O PHP trata esses valores como variáveis, cujo nome é o nome do campo definido no formulário. O exemplo a seguir mostra isso, e mostra também como o código PHP pode ser inserido em qualquer parte do código HTML:

```
<html>
<head><title>Aprendendo PHP</title></head>
<body>
<?php
if ($_POST['texto'] != "")
{
  $texto = $_POST['texto'];
  echo "Você digitou $texto <br><br>";
}
?>
<form method=post action="testeform.php">
<input type="text" name="texto" value="" size=50>
<br>
<input type="submit" name="sub" value="Enviar!">
</form>
</body>
</html>
```

Ao salvar o arquivo acima e carregá-lo no browser, o usuário verá apenas um formulário que contém um espaço para digitar o texto.. Ao digitar um texto qualquer e submeter o formulário, a resposta será uma frase mostrando o texto digitado e exibindo novamente o formulário para você refazer a operação.

Neste exemplo utilizamos a função $_POST, que é a função que recebe os dados vindo de um formulário do tipo POST. Deve ser especificado entre colchetes o nome do campo do formulário que você deseja, pois imagine um formulário de cadastro completo com nome, endereço, telefone, e-mail, etc, temos que especificar qual o campo do formulário desejamos.

Na primeira vez que a página é carregada, a variável texto do formulário não contém nenhum conteúdo, por isso a estrutura condicional a desvia e não exibe a frase. Quando o usuário apertar o Submit, ele enviará os dados para a variável. Nesse momento, quando o teste for feito, ela estará com o conteúdo digitado e exibe o conteúdo.

10. Funções

A sintaxe básica para definir uma função é:

```
function nome_da_função([arg1, arg2, arg3]) {
        Comandos;
        ... ;
        [return <valor de retorno>];
}
```

Qualquer código PHP válido pode estar contido no interior de uma função. Como a checagem de tipos em PHP é dinâmica, o tipo de retorno não deve ser declarado, sendo necessário que o programador esteja atento para que a função retorne o tipo desejado. É recomendável que esteja tudo bem documentado para facilitar a leitura e compreensão do código. Para efeito de documentação, utiliza-se o seguinte formato de declaração de função:

```
tipo function nome_da_funcao(tipo arg1, tipo arg2, ...);
```

Este formato só deve ser utilizado na documentação do script, pois o PHP não aceita a declaração de tipos. Isso significa que em muitos casos o programador deve estar atento ao tipos dos valores passados como parâmetros, pois se não for passado o tipo esperado não é emitido nenhum alerta pelo interpretador PHP, já que este não testa os tipos.

Valor de retorno

Toda função pode opcionalmente retornar um valor, ou simplesmente executar os comandos e não retornar valor algum.

Não é possível que uma função retorne mais de um valor, mas é permitido fazer com que uma função retorne um valor composto, como listas ou arrays.

Argumentos

É possível passar argumentos para uma função. Eles devem ser declarados logo após o nome da função, entre parênteses, e tornam-se variáveis pertencentes ao escopo local da função. A declaração do tipo de cada argumento também é utilizada apenas para efeito de documentação.

Exemplo:

```
function imprime($texto){
  echo $texto;
}
imprime(“teste de funções”);
```

Passagem de parâmetros por referência

Normalmente, a passagem de parâmetros em PHP é feita por valor, ou seja, se o conteúdo da variável for alterado, essa alteração não afeta a variável original.

Exemplo:

```
function mais5($numero) {
  $numero += 5;
}
$a = 3;
mais5($a); //$a continua valendo 3
```

No exemplo acima, como a passagem de parâmetros é por valor, a função mais5 é inútil, já que após a execução sair da função o valor anterior da variável é recuperado. Se a passagem de valor fosse feita por referência, a variável $a teria 8 como valor. O que ocorre normalmente é que ao ser chamada uma função, o interpretador salva todo o escopo atual, ou seja, os conteúdos das variáveis. Se uma dessas variáveis for passada como parâmetro, seu conteúdo fica preservado, pois a função irá trabalhar na verdade com uma cópia da variável. Porém, se a passagem de parâmetros for feita por referência, toda alteração que a função realizar no valor passado como parâmetro afetará a variável que o contém.

Há duas maneiras de fazer com que uma função tenha parâmetros passados por referência: indicando isso na declaração da função, o que faz com que a passagem de parâmetros sempre seja assim; e também na própria chamada da função. Nos dois casos utiliza-se o modificador “&”. Vejamos um exemplo que ilustra os dois casos:

```
function mais5(&$num1, $num2) {
  $num1 += 5;
  $num2 += 5;
```

```
}

$a = $b = 1;

mais5($a, $b); /* Neste caso, só $num1 terá seu valor alterado, pois a passagem por referência está definida na declaração da função. */

mais5($a, &$b); /* Aqui as duas variáveis terão seus valores alterados. */
```

Argumentos com valores pré-definidos (default)

Em PHP é possível ter valores default para argumentos de funções, ou seja, valores que serão assumidos em caso de nada ser passado no lugar do argumento. Quando algum parâmetro é declarado desta maneira, a passagem do mesmo na chamada da função torna-se opcional.

```
function teste($texto = “testando”) {
  echo $texto;
}

teste(); // imprime “testando”

teste(“outro teste”); // imprime “outro teste”
```

É bom lembrar que quando a função tem mais de um parâmetro, o que tem valor default deve ser declarado por último:

```
function teste($figura = circulo, $cor) {
 echo “a figura é um “, $figura, “ de cor “ $cor;
}

teste(azul);

/* A função não vai funcionar da maneira esperada, ocorrendo um erro no interpretador. A declaração correta é: */

function teste2($cor, $figura = circulo) {
 echo “a figura é um “, $figura, “ de cor “ $cor;
}

teste2(azul);
```

/* Aqui a função funciona da maneira esperada, ou seja, imprime o texto: “a figura é um círculo de cor azul” */

Contexto

O contexto é o conjunto de variáveis e seus respectivos valores num determinado ponto do programa. Na chamada de uma função, ao iniciar a execução do bloco que contém a implementação da mesma é criado um novo contexto, contendo as variáveis declaradas dentro do bloco, ou seja, todas as variáveis utilizadas dentro daquele bloco serão eliminadas ao término da execução da função.

Escopo

O escopo de uma variável em PHP define a porção do programa onde ela pode ser utilizada. Na maioria dos casos todas as variáveis têm escopo global. Entretanto, em funções definidas pelo usuário um escopo local é criado. Uma variável de escopo global não pode ser utilizada no interior de uma função sem que haja uma declaração.

Exemplo:

```
$texto = “Testando”;

function Teste() {

  echo $texto;

}


Teste();
```

O trecho acima não produzirá saída alguma, pois a variável $texto é de escopo global, e não pode ser referida num escopo local, mesmo que não haja outra com nome igual que cubra a sua visibilidade. Para que o script funcione da forma desejada, a variável global a ser utilizada deve ser declarada.

Exemplo:

```
$vivas = “Testando”;

function Teste() {

  global $vivas;

  echo $vivas;

}
```

```
Teste();
```

Uma declaração “global” pode conter várias variáveis, separadas por vírgulas. Outra maneira de acessar variáveis de escopo global dentro de uma função é utilizando um array pré-definido pelo PHP cujo nome é $GLOBALS. O índice para a variável referida é o próprio nome da variável, sem o caractere $. O exemplo acima e o abaixo produzem o mesmo resultado:

Exemplo:

```
$vivas = "Testando";

function Teste() {

  echo $GLOBALS["vivas"]; // imprime $vivas

  echo $vivas; // não imprime nada

}

Teste();
```

PARTE IV

PHP e MySQL – Introdução

Uma das maiores vantagens em utilizarmos a linguagem PHP em páginas de internet é a facilidade que ela oferece para acessar bancos de dados, em especial, o banco MySQL.

Utilizaremos aqui um banco de dados MySQL para simular um livro de visitas de uma página qualquer.

Conexão com o Banco de Dados

Antes de mais nada, precisaremos criar o banco e a tabela que iremos utilizar para que possamos fazer as conexões com o PHP e MySQL.

Se você não está familiarizado com o PHPMyAdmin, dê uma olhada nesse tutorial.

Crie a estrutura abaixo em um banco de dados. Em nosso modelo, criaremos um banco chamado Visitas com as seguintes tabelas:

Tabela Livro de assinaturas (Livro)

```
CREATE TABLE `livro` (
`cod` INT( 5 ) NOT NULL AUTO_INCREMENT PRIMARY KEY ,
`nome` VARCHAR( 255 ) NOT NULL ,
`localizacao` VARCHAR( 50 ) NOT NULL ,
`mensagem` TEXT NOT NULL ,
`data` DATETIME NOT NULL
);
```

Vamos inserir pelos menos cinco registros para fazermos nossos testes com o PHP.

Com o banco de dados e a tabela criados e alguns registros inseridos passaremos agora à conexão do banco com a página php.

Em primeiro lugar, criaremos uma página que irá fazer a conexão com o banco e exibir os dados existentes na tabela.

Iremos mostrar o código da página mostra.php e em seguida explicaremos passo a passo os comandos utilizados:

```
<?php
// Mensagens de Erro
$msg[0] = "Conexão com o banco falhou!";
$msg[1] = "Não foi possível selecionar o banco de dados!";

// Fazendo a conexão com o servidor MySQL
$conexao = mysql_pconnect("localhost","root","") or die($msg[0]);
mysql_select_db("visitas",$conexao) or die($msg[1]);

// Colocando o Início da tabela
?>
```

```
<table border="1"><tr>
   <td><b>Cód</b></td>
   <td><b>Nome</b></td>
   <td><b>Localização</b></td>
</tr>
<?php

// Fazendo uma consulta SQL e retornando os resultados em uma tabela
HTML
$query = "SELECT cod,nome,localizacao FROM livro ORDER BY nome";
$resultado = mysql_query($query,$conexao);
while ($linha = mysql_fetch_array($resultado)) {
   ?>
   <tr>
      <td><?php echo $linha['cod']; ?></td>
      <td><?php echo $linha['nome']; ?></td>
      <td><?php echo $linha['localizacao']; ?></td>
   </tr>
   <?php
}
?>
</table>
```

As primeiras linhas do código criam um vetor com duas strings que utilizaremos para dar mensagens de erro na hora de conexão ou seleção do banco de dados.

Em seguida, utilizamos a função **mysql_pconnect()** que é responsável pela abertura da conexão com o banco de dados mysql. Como parâmetros, enviamos o endereço do banco (nesse caso foi localhost, mas poderia ser o IP ou o host do servidor por exemplo), o nome do usuário (nesse caso o usuário foi o root), e enviamos a senha (que nesse caso é vazia, por isso “”). Na linha de comando terminamos com a chamada *or die,* que mostrará a mensagem que estiver em seguida, nesse caso usamos o vetor da primeira linha na posição zero.

Por que usar **mysql_pconnect()** e não usar **mysql_connect()**? Depende o que você irá fazer em sua página. A letra **p** antes do **connect** é abreviação de conexão persistente. O que é isso? Acesse o manual do PHP para termos uma noção clicando em http://www.php.net/manual/en/features.persistent-connections.php

Em seguida acionamos o comando **mysql_select_db()** que fará a seleção do banco de dados a ser utilizado. Passamos como parâmetros nesse caso o banco visitas que foi criado no mysql e indicamos em qual conexão. Mais uma vez encerramos com o comando *or die* que exibirá uma mensagem caso falhe, nesse segundo caso utilizamos o mesmo vetor com a posição um.

Em seguida vem a demonstração da flexibilidade do PHP, você pode sair e entrar da programação PHP a qualquer momento, bastando utilizar as tags <?PHP e ?>. Neste ponto, a gente fecha a programação PHP e começa a desenhar uma tabela html simples. Após o desenho do cabeçalho da tabela, acionamos mais uma vez o modo PHP para trazermos os dados do banco de dados para a página.

Vamos na seqüência usar a função **mysql_query()** que é responsável pela consulta no banco de dados. Basta inserir uma instrução SQL válida para termos o resultado. A função tem como parâmetros uma sql (ou variável string que contenha uma sql) e a conexão que a sql será executada.

Em seguida vem um pequeno truque para mostrarmos todos os dados retornados na consulta. Faremos um laço **While** ler linha a linha o resultado da mysql_query (que neste caso foi atribuída à variável $resultado) juntamente com a função **mysql_fecth_array()**, que transformará $resultado em um vetor. Como temos 5 linhas em nosso banco de dados, o laço repetirá 5 vezes apenas.

Observe que os dados de cada linha da tabela serão atribuídos ao vetor $linha criado na instrução, e acessaremos um a um com a chamada de seu nome como índice (**$linha[´coluna´]**), como podemos ver em $linha[´nome´], $linha[´cod´], etc.

Depois de passarmos as 5 linhas, novamente fechamos o modo PHP e acionamos o HTML para fechar a tabela.

Basicamente essas são as instruções necessárias para acessarmos os dados em uma tabela de um banco de dados mysql.

GRAVAR DADOS NO NOSSO LIVRO DE VISITAS

Para oferecermos a opção de salvar os dados no livro de visitas, utilizaremos um formulário html solicitando os dados a serem preenchidos e enviando-os para o php. Veja o código da página assinar.php

```
<?PHP
//vamos verificar se o método da página é POST
if (getenv("REQUEST_METHOD") == "POST") {
   // Configura as variáveis do método POST para virarem variáveis
   $nome = $_POST['nome'];
   $localizacao = $_POST['localizacao'];
   $mensagem = $_POST['mensagem'];

   // Caso todos os campos forem preenchidos, inclui a mensagem no
   // banco de dados. Caso isso não aconteça, gera uma mensagem de
   // erro que será impressa no browser mais a frente.
   if ($nome and $localizacao and $mensagem) {
      $conexao = mysql_pconnect("localhost","root","");
      mysql_select_db("visitas",$conexao);
$query = "INSERT INTO livro
VALUES('00000','$nome','$localizacao','$mensagem',NOW())";
      mysql_query($query,$conexao);
      header("Location: ler.php");
   } else {
      $err = "Preencha todos os campos!";
   }
}

?>
<html>
<head>
   <title>Livro de Visitas: Assinar</title>
</head>
<body bgcolor="white">

<h1>Assine o Livro de Visitas</h1>

<?PHP
// Se ocorreu algo de errado, então vai existir uma variável $err
// contendo a mensagem. Imprime-se então em fonte vermelha esta
// mensagem.
```

```
if ($err) {
   ?>
   <ul><font color="red"><?PHP echo $err; ?></font></ul>
   <?PHP
}
?>

<form method="post" action="assinar.php">
<table border="0"
<tr>
   <td>Nome: </td>
   <td><input type="text" size="15" name="nome" maxlength="250"></td>
</tr>
<tr>
   <td>Localização: </td>
   <td><input type="text" size="15" name="localizacao"
maxlength="45"></td>
</tr>
<tr>
   <td colspan="2">
   <textarea cols="60" rows="10" name="mensagem">Digite aqui sua
mensagem!</textarea>
   </td>
</tr>
</table>
<input type="submit" value="Assinar">
</form>

</body>
</html>
```

Aqui montamos um formulário padrão em HTML que receberá os dados do usuário. Quando pressionar o botão Assinar o formulário envia os dados com o método escolhido (no caso o POST para que os dados não sejam visíveis para o usuário) para a própria página, como se ela fosse chamada novamente. Nada impede que mandemos para outra página que fará apenas a validação e gravação dos dados.

Ao entrar na página, ela verifica se os dados estão chegando através de POST. Caso afirmativo, quer dizer que o usuário pressionou o botão de assinar. Porém não podemos garantir que os dados foram preenchidos. Por isso nossa primeira função é identificar isso.

Teremos então que criar variáveis com os dados que estão chegando via POST. Para isso, basta que criemos variáveis (no exemplo $nome) e igualemos aos dados que estão chegando, um a um (por exemplo $nome = $_POST[´nome´]). Os dados chegam no vetor $_POST, bastando identificarmos o ítem desejado.

Uma vez com os dados checamos se os mesmos estão preenchidos (if $nome and $mensagem and $localizacao). Nesta instrução, caso algum deles estiver vazio retornará falso, não deixando o programa abrir a conexão com o banco nem tentando gravar dados inválidos.

Uma vez que todos estão preenchidos, deveremos abrir a conexão com o banco novamente (como explicado acima) e proceder a gravação, com uma SQL de Insert (inserção).

Com a instrução preenchida, usamos mysql_select_db para executar nosso SQL.

Em seguida, encontramos a instrução header. Essa instrução é muito utilizada para inserirmos dados no cabeçalho (head) do html. Uma limitação do HTML é que nenhuma linha de página deve ter sido escrita antes de sua utilização. Nesse

caso, como não escrevemos nada, enviamos a instrução Location, que irá desviar a página para outro arquivo (no caso, ler.php)

EXIBINDO OS DADOS DO NOSSO LIVRO DE VISITAS

Agora mostraremos as assinaturas de nosso livro de visitas aos nossos visitantes. Para isso criaremos o arquivo chamado ler.php que está mostrado abaixo:

```
<html>
<head>
   <title>Livro de Visitas: Ler</title>
</head>
<body bgcolor="white">

<h1>Livro de Visitas</h1>

<?PHP
// Verifica se existe a variável $begin, que vai indicar a número
// da mensagem que vai aparecer no começo. Se não existir, assume-se
// que vai ser o começo, ou seja, o valor 0.
$begin = $_GET['begin'];
if (!$begin) { $begin = 0; }

// Conecta ao servidor e seleciona o banco de dados
$conexao = mysql_pconnect("localhost","root","");
mysql_select_db("visitas",$conexao);

// Coloca na variável $total o número total de mensagens no Guestbook
$query = "SELECT count(*) FROM livro";
$query = mysql_query($query,$conexao);
$query = mysql_fetch_array($query);
$total = $query[0];

?>
<p>
Total de mensagens postadas: <b><?PHP echo $total; ?></b>
(<a href="assinar.php">Assine você também!</a>)<br>
Exibindo <b>20</b> mensagens por página, mostrando mensagens de
<b><?PHP echo $begin+1; ?></b> a <b><?PHP echo $begin+20; ?></b>.
</p>

<?PHP
// Calcula os links para as próximas mensagens e as anteriores, de
// acordo com o número total de mensagens
if (($begin > 0) and ($begin <= 20)) {
   $anteriores = '<a href="ler.php?begin=0">Anteriores</a>';
} elseif (($begin > 0) and ($begin > 20)) {
   $anteriores   =   '<a   href="ler.php?begin='   .   ($begin-20)   .
'">Anteriores</a>';
} else {
   $anteriores = 'Anteriores';
}

if (($begin < $total) and (($begin+20) >= $total)) {
   $proximas = 'Próximas';
} else {
```

```
   $proximas = '<a href="ler.php?begin=' . ($begin+20) .
'">Próximas</a>';
}

echo $anteriores . " | " . $proximas;

// Faz uma consulta SQL trazendo as linhas das 20 ultimas mensagens
// que foram colocadas no livro de visitas.
$query = "SELECT * FROM livro ORDER BY data DESC LIMIT $begin,20";
$query = mysql_query($query,$conexao);

// Gera uma tabela para cada assinatura no livro de visitas (loop)
while ($linha = mysql_fetch_array($query)) {
   // Organiza a mostragem da data, já que no campo do MySQL, a data
   // se encontra em uma forma não tão legal.
   $var = $linha['data'];
   $var = explode(" ",$var);
   $dia = $var[0];
   $hora = $var[1];
   $dia = explode("-",$dia);
   $data = "$dia[2]/$dia[1]/$dia[0] às $hora";
   ?>

   <table border="0" width="70%">
   <tr><td bgcolor="navy" colspan="2"> </td></tr>
   <tr>
      <td><b>Data:</b></td>
      <td width="100%"><?PHP echo $data; ?></td>
   </tr>
   <tr>
      <td><b>Nome:</b></td>
      <td><?PHP echo $linha['nome']; ?></td>
   </tr>
   <tr>
      <td><b>Localização:</b></td>
      <td><?PHP echo $linha['localizacao']; ?></td>
   </tr>
   <tr>
      <td><b>Mensagem:</b></td>
      <td><?PHP echo $linha['mensagem']; ?></td>
   </tr>
   </table>
   <?PHP
}

?>
</body>
</html>
```

O código em si já possui diversos comentários, mas vamos destacar algumas funções e métodos utilizados aqui:

De início identificamos uma variável que está chegando (ou não) à página pelo método GET (que estará no vetor $_GET). Essa variável será utilizada para a paginação das mensagens. Esta página está programada para mostrar páginas de mensagens de 20 em 20. Quando o livro tiver mais de 20 assinaturas, elas serão separadas em páginas controladas pela variável $begin conseguida pelo método get ($begin=$_GET[´begin´]). Ela indicará o número da mensagem que iniciará a página.

Se ela não existe, então colocaremos o valor zero para indicar que está na primeira página.

Em seguida, acessamos o banco de dados (como já descrito anteriormente) e realizamos uma consulta (select count(*) from livro) para descobrirmos a quantidade de registros cadastrados na tabela. Atribuímos esse valor à variável $total, que é exibida em seguida.

Na seqüência utilizamos a variável $begin para que possamos realizar a paginação, descobrindo quais mensagens exibir, se ativa botões próximos e anteriores, etc.

Depois de paginarmos tudo, finalmente vem a seção onde iremos mostrar de fato as mensagens. A primeira coisa a reparar é a instrução DESC e LIMIT. A instrução DESC apenas indica que a ordem de exibição será inversa, ou seja, da maior para a menor (no caso, foi ordenado por data). A instrução LIMIT, como o próprio nome diz, "limita" a quantidade de linhas que será retornada, iniciando a partir do valor de $begin. Seria como se disséssemos começando de 0 até 0+20.

Em seguida, basta que mostremos os dados em seus devidos lugares, montando a formatação desejada.

Existe também um exemplo em como adaptar a exibição da data, já que no MySQL a data é armazenada em um formato diferente do que o que estamos acostumados a observar.

Alteração de registros

Para fazermos alterações nos registros, o primeiro passo é organizar uma forma de selecionarmos o registro que desejamos manipular. Existem diversas formas. Neste caso, optamos em realizar a busca com o auxílio de uma combo contendo os nomes das pessoas que assinaram nosso livro, mas vamos utilizar como parâmetro de pesquisa, o código do registro que contém aquele nome. Vejamos o código da página procurar.php que realiza a listagem dos nomes na combo:

```
<html>
      <head><title>Procuras</title></head>
<body><center>
   <form method=post action=exibe.php>
   <table border=1 bordercolor=blue><tr><td colspan=2><center><b><font
face=arial     size=4     color=blue>Alteração     ou     exclusão     de
registros</font></b></center></td></tr>
   <tr><td width=150><font face=arial>Selecione um nome: </td><td>
   <select name=procura size=1>
   <?PHP
            include("conecta.php");
            $query = "SELECT nome, cod FROM livro ORDER BY data DESC";
            $query = mysql_query($query,$conexao);
            while ($linha = mysql_fetch_array($query)) {
             echo                                                  '<option
value='.$linha['cod'].'>'.$linha['nome'].'</option>';

            }
   ?>
   </select></td></tr>
   <tr><td><center><input                                        type=reset
value=Limpar></td><td><center><input                             type=submit
value=Enviar></td></tr>
   </table>
   </form>
```

```
   <br><br><center><a href=index.php>Voltar ao menu
principal</a></center>
   </body>
</html>
```

O código começa com a declaração de um formulário, simples, em html. Utilizaremos o objeto select para criarmos a combo, que será preenchida dinamicamente com o PHP a seguir.

A primeira novidade aparece na função **include()**. Essa função tem a capacidade de inserir um arquivo inteiro dentro do arquivo que estamos trabalhando.

Nesse caso, optamos em criar um arquivo padrão com as instruções de conexão com o banco de dados e seleção do banco padrão. Criamos o arquivo conecta.php com as seguintes instruções:

```
<?PHP
// Mensagens de Erro
$msg[0] = "Conexão com o banco falhou!";
$msg[1] = "Não foi possível selecionar o banco de dados!";
// Fazendo a conexão com o servidor MySQL
$conexao = mysql_pconnect("localhost","root","") or
die($msg[0]);
mysql_select_db("visitas",$conexao) or die($msg[1]);
?>
```

E quando a página procurar.php insere a linha **include("conecta.php")** seria como se copiássemos todo o conteúdo do arquivo conecta.php para procurar.php naquela posição. Dessa forma, em todos os próximos arquivos aqui descritos, sempre que precisarmos abrir uma conexão com o banco de dados livros chamaremos a instrução include("conecta.php") ao invés de digitarmos todas essas instruções.

Feita a conexão preparamos uma consulta listando todos os nomes e códigos em ordem de entrada inversa e criamos a instrução para que todos os nomes sejam listados na select através do comando Option do HTML. Note que no option temos dois parâmetros. O primeiro identifica qual o dado que será enviado quando o select estiver selecionado e o segundo qual dado será exibido na página para que o usuário possa procurar a opção desejada.

Nesse caso colocamos em value do option a coluna cod que será enviada como post para a próxima página e colocamos a coluna nome para ser exibido na tela com essa instrução

```
echo '<option value='.$linha['cod'].'>'.$linha['nome'].'</option>';
```

Feito isso já temos todos os nomes listados prontos para enviar o código para a próxima página pelo método Post, bastando o usuário clicar no botão enviar (submit do form).

Será acionada a página exibe.php, com o seguinte código:

```
<?PHP
//vamos verificar se o método da página é POST
if (getenv("REQUEST_METHOD") == "POST") {
   // Configura as variáveis do método POST para virarem variáveis
   $cod = $_POST['procura'];
   //Faz a busca pelo código solicitado de acordo com o nome
pesquisado na página anterior
      include("conecta.php");
      $query = "Select * from livro where cod = ".$cod;
      $query = mysql_query($query,$conexao);
}
```

```
?>
<html>
<head>
   <title>Livro de Visitas: Alterar</title>
</head>
<body bgcolor="white">

<h1>Alterações no Livro de Visitas</h1>

<?PHP
// Se ocorreu algo de errado, então vai existir uma variável $err
// contendo a mensagem. Imprime-se então em fonte vermelha esta
// mensagem.
if ($err) {
   ?>
   <ul><font color="red"><?PHP echo $err; ?></font></ul>
   <?PHP
   } else {
$linha = mysql_fetch_array($query);
   }
?>
<form method="post" action="alterar.php">
<table border="0"
<tr>
   <td>Nome: </td>
   <td><input type="text" size="150" name="nome" value ="<?PHP echo
$linha['nome']; ?>" maxlength="250"></td>
</tr>
<tr>
   <td>Localização: </td>
   <td><input type="text" size="150" name="localizacao" value ="<?PHP
echo $linha['localizacao']; ?>" maxlength="45"></td>
</tr>
<tr>
   <td>Data: </td>
   <td><input type="text" size="20" name="data" value ="<?PHP echo
$linha['data']; ?>" maxlength="19"></td>
</tr>
<tr>
   <td colspan="2">
   <textarea cols="60" rows="10" name="mensagem"><?PHP echo
$linha['mensagem']; ?></textarea>
   </td>
</tr>
</table>
<input type=hidden name=cod value = "<?PHP echo $linha['cod']; ?>">
<input type=reset value="Recarregar"> <input type="submit"
value="Alterar">
</form>
<form method=post action=deletar.php>
<input type=hidden name=cod value = "<?PHP echo $linha['cod']; ?>">
<br><input type=submit value="Apagar"></form>
<br><br><center><a href=index.php>Voltar ao menu
principal</a></center>
</body>
</html>
```

Esta página receberá o código de acordo com o nome selecionado em procurar.php. Ela inicia checando se os dados estão chegando via POST para pode realizar a busca do registro. Mais uma vez é chamada a função include para trazer o arquivo conecta.php para que possamos abrir a conexão com a tabela livro.

Assim que é feita a consulta, utilizamos um formulário semelhante ao formulário utilizado na hora de assinar o livro de visitas (da página assinar.php), porém com a diferença de mostrarmos os dados nos campos, relativos ao nome selecionado na página anterior. Mais uma vez é criado um formulário com os dados preenchidos que aponta para outra página que, depois de fazer as alterações e pressionar o botão alterar, cuidará de realizar as alterações no banco de dados.

O final do formulário será utilizado na hora de apagar um registro. Será comentado adiante neste texto.

A página responsável pela alteração é a alterar.php, descrita abaixo:

```
<?PHP
//vamos verificar se o método da página é POST
if (getenv("REQUEST_METHOD") == "POST") {
   // Configura as variáveis do método POST para virarem variáveis
   $nome = $_POST['nome'];
   $localizacao = $_POST['localizacao'];
   $mensagem = $_POST['mensagem'];
   $data = $_POST['data'];
   $cod = $_POST['cod'];
   // Caso todos os campos forem preenchidos, inclui a mensagem no
   // banco de dados. Caso isso não aconteça, gera uma mensagem de
   // erro que será impressa no browser mais a frente.
   if ($nome and $localizacao and $mensagem and $data) {
      include("conecta.php");
  $query = "UPDATE `livro` SET `nome` = '".$nome."',`localizacao` =
'".$localizacao."',`mensagem` = '".$mensagem."',`data` = '".$data."'
WHERE `cod` =".$cod;
  mysql_query($query,$conexao);
      header("Location: procurar.php");
   } else {
      $err = "Preencha todos os campos!";
echo err."<br><a href=javascript:history.go(-1)>Voltar</a>";
   }
}
?>
```

Essa página praticamente não terá nenhum conteúdo, apenas se o usuário entrar direto nela sem enviar dados via POST mostrará uma mensagem para retornar ou se algum campo for deixado em branco, que no caso receberá uma opção para voltar e corrigir antes de salvar as alterações.

A página começa com a checagem se os dados vieram via POST. Se sim, ela trata de converter as variáveis do vetor POST para variáveis locais e em seguida efetua a SQL que irá fazer a alteração.

Logo depois a função header (“location: “) irá desviar a página para a página procurar.php novamente. Se o nome for alterado já estará visível a alteração.

Exclusão de registros

Na página procurar.php observamos a existência também de um formulário à parte no final, que será responsável pelo acionamento de uma página (deletar.php) específica para que possamos apagar o registro.

Uma vez pressionado o botão, ele acionará essa página (deletar.php) com o seguinte código:

```
<?PHP
//vamos verificar se o método da página é POST
if (getenv("REQUEST_METHOD") == "POST") {
   // Configura as variáveis do método POST para virarem variáveis
   $cod = $_POST['cod'];
      include("conecta.php");
        $query = "DELETE FROM `livro` WHERE `cod` =".$cod;
        mysql_query($query,$conexao);
      header("Location: procurar.php");
   } else {
      $err = "Selecione o registro antes de excluir!";
echo $err."<br><a href=javascript:history.go(-1)>Voltar</a>";
   }
?>
```

Assim como na página alterar.php essa página também não produz nenhum resultado em tela. Ela apenas abre a conexão com o banco e executa uma SQL com o código recebido pelo POST da página procurar.php e apaga o registro. Tendo sucesso ela redireciona pelo header para a página procurar.php novamente e se ela não for acionada corretamente dará uma mensagem de erro, solicitando que o usuário selecione o registro antes.

Procura de registros

Criamos aqui duas formas simples de pesquisa de registros pelo nome. Uma delas irá solicitar do usuário um nome para buscas e apenas exibirá os dados na tela. A outra solicitará o nome também, mas quando mostrar o resultado mostrará dois botões para que o usuário possa, à partir da consulta, alterar ou apagar os registros encontrados.

Vamos ao código da página que recebe os nomes (consultar.php):

```
<html>
      <head><title>Procuras</title></head>
<body><center>
   <form method=post action=resultados.php>
   <table border=1 bordercolor=blue><tr><td colspan=2><center><b><font
face=arial           size=4           color=blue>Consulta           de
registros</font></b></center></td></tr>
   <tr><td width=150><font face=arial>Digite um nome: </td><td>
   <input type=text name=nome width=150></td></tr>
   <tr><td><center><input                                    type=reset
value=Limpar></td><td><center><input                        type=submit
value=Procurar></td></tr>
   </table>
   </form>

   <hr color=red>
   <form method=post action=resultados2.php>
   <table border=1 bordercolor=blue><tr><td colspan=2><center><b><font
face=arial           size=4           color=blue>Consulta           com
opções</font></b></center></td></tr>
   <tr><td width=150><font face=arial>Digite um nome: </td><td>
   <input type=text name=nome width=150></td></tr>
```

```
   <tr><td><center><input                                    type=reset
value=Limpar></td><td><center><input                         type=submit
value=Procurar></td></tr>
   </table>
   </form>
   <br><br><center><a       href=index.php>Voltar       ao       menu
principal</a></center>
   </body>
</html>
```

Observe que temos praticamente duas vezes o mesmo código nesta página, porém uma delas o formulário envia os dados para a página resultados.php e na outra para resultados2.php.

Essa página é puramente html e não contém nenhum código php. Ela apenas é usada para criar os formulários e enviar os dados via POST

Em seguida vamos analisar o código da página resultados.php

```
<html><head><title>Resultados da pesquisa</title></head>
<body>
<table border="1"><tr>
   <td><b>Cód</b></td>
   <td><b>Nome</b></td>
   <td><b>Localização</b></td>
   <td><b>Data</b></td>
   <td><b>Mensagem</b></td>
</tr>
<?php
//chama a conexão com o banco de dados
include("conecta.php");
//pega a variável POST
$nome = $_POST['nome'];
// Fazendo uma consulta SQL e retornando os resultados em uma tabela
HTML
$query =  "SELECT * FROM livro WHERE nome like '%".$nome."%'";
$resultado = mysql_query($query,$conexao);
//mostrando os resultados
while ($linha = mysql_fetch_array($resultado)) {
   ?>
   <tr>
      <td><?php echo $linha['cod']; ?></td>
      <td><?php echo $linha['nome']; ?></td>
      <td><?php echo $linha['localizacao']; ?></td>
        <td><?php echo $linha['data']; ?></td>
        <td><?php echo $linha['mensagem']; ?></td>
   </tr>
   <?php
}
?>
      <br><br><center><a       href=index.php>Voltar       ao       menu
principal</a></center>
</table>
</body>
</html>
```

Aqui vamos receber os dados via POST e fazer uma consulta no banco de dados com o nome (ou pedaço dele) que foi digitado pelo usuário.

Note na consulta a utilização de LIKE na SQL e a adição de % antes e depois do nome recebido no POST. Isso é para que caso o usuário digite apenas

Silva, por exemplo, a página procure no banco nomes que contenham Silva em qualquer posição (início, meio ou fim do nome).

Uma vez localizados os nomes que contenham a procura do usuário, os dados são mostrados através de uma tabela semelhante à página mostrar.php, primeiro exemplo do manual.

Já a página resultados2.php descrita abaixo, realiza as mesmas funções, porém exemplifica como podemos criar dois botões para que, com os resultados encontrados, possamos alterar ou apagar os registros desejados.

```
<html><head><title>Resultados da pesquisa</title></head>
<body>
<table border="1"><tr>
   <td><b>Opção 1</b></td>
   <td><b>Opção 2</b></td>
   <td><b>Cód</b></td>
   <td><b>Nome</b></td>
   <td><b>Localização</b></td>
   <td><b>Data</b></td>
   <td><b>Mensagem</b></td>
</tr>
<?php
//chama a conexão com o banco de dados
include("conecta.php");
//pega a variável POST
$nome = $_POST['nome'];
// Fazendo uma consulta SQL e retornando os resultados em uma tabela
HTML
$query =  "SELECT * FROM livro WHERE nome like '%".$nome."%'";
$resultado = mysql_query($query,$conexao);
//mostrando os resultados
while ($linha = mysql_fetch_array($resultado)) {
   ?>
   <tr>
      <td><form   action="exibe.php"   method=post><input   type=hidden
name=procura   value=<?PHP   echo   $linha['cod'];?>><input   type=submit
value=Alterar></form></td>
      <td><form   action="deletar.php"   method=post><input   type=hidden
name=cod   value=<?PHP   echo   $linha['cod'];?>><input   type=submit
value=Apagar></form></td>
      <td><?php echo $linha['cod']; ?></td>
      <td><?php echo $linha['nome']; ?></td>
      <td><?php echo $linha['localizacao']; ?></td>
        <td><?php echo $linha['data']; ?></td>
        <td><?php echo $linha['mensagem']; ?></td>
   </tr>
   <?php
}
?>
</table>
<center><a href=index.php>Voltar ao menu principal</a></center>
</body>
</html>
```

Esses botões são criados em duas células da tabela e pertencem a um formulário padrão html, mas o único parâmetro visível é o submit. Observamos que os dados necessários estão ocultos (input type=hidden) e tem os mesmos nomes das páginas que irão recebê-los (no caso alterar.php e deletar.php).

ANEXO I – Funções diversas em PHP

Strings

Strings podem ser atribuídas de duas maneiras:

a) utilizando aspas simples (') – Desta maneira, o valor da variável será exatamente o texto contido entre as aspas (com exceção de \\ e \' – ver tabela abaixo)
b) utilizando aspas duplas (") – Desta maneira, qualquer variável ou caracter de escape será expandido antes de ser atribuído.

Exemplo:

```
<?
$teste = "Mauricio";
$vivas = '---$teste--\n';
echo "$vivas";
?>
```

A saída desse script será "---$teste--\n".

```
<?
$teste = "Mauricio";
$vivas = "---$teste---\n";
echo "$vivas";
?>
```

A saída desse script será "---Mauricio--" (com uma quebra de linha no final).

A tabela seguinte lista os caracteres de escape:

Sintaxe	Significado
\n	Nova linha
\r	Retorno de carro (semelhante a \n)
\t	Tabulação horizontal
\\	A própria barra (\ )
\$	O símbolo $
\’	Aspa simples
\”	Aspa dupla

Funções para tratamento de strings

htmlspecialchars

```
string htmlspecialchars(string str);
```

Retorna a string fornecida, substituindo os seguintes caracteres:

- & para '&'
- " para '"'
- < para '<'
- > para ‘>'

htmlentities

```
string htmlentities(string str);
```

Funciona de maneira semelhante ao comando anterior, mas de maneira mais completa, pois converte todos os caracteres da string que possuem uma representação especial em html, como por exemplo:

- º para 'º'
- ª para 'ª'
- á para 'á'
- ç para ‘ç'

nl2br

```
string nl2br(string str);
```

Retorna a string fornecida substituindo todas as quebras de linha (“\n”) por quebras de linhas em html (“
”).

```
Exemplo:
echo nl2br(“Mauricio\nVivas\n”);

Imprime:
Maurício<br>Vivas<br>
```

get_meta_tags

```
array get_meta_tags(string arquivo);
```

Abre um arquivo html e percorre o cabeçalho em busca de “meta” tags, retornando num array todos os valores encontrados.

```
Exemplo:
No arquivo teste.html temos:
...
<head>
<meta name="author" content="jose">
<meta name="tags" content="php3 documentation">
...
</head><!-- busca encerra aqui -->
...

a execução da função:

get_meta_tags(“teste.html”);

retorna o array:

array(“author”=>”jose”,”tags”=>"php3 documentation");
```

strip_tags

```
string strip_tags(string str);
```

Retorna a string fornecida, retirando todas as tags html e/ou PHP encontradas.

```
Exemplo:
strip_tags('<a href="teste1.php3">testando</a><br>');

Retorna a string “testando”
```

urlencode

```
string urlencode(string str);
```

Retorna a string fornecida, convertida para o formato urlencode. Esta função é útil para passar variáveis para uma próxima página.

urldecode

```
string urldecode(string str);
```

Funciona de maneira inversa a urlencode, desta vez decodificando a string fornecida do formato urlencode para texto normal.

Funções relacionadas a arrays

Implode e join

```
string implode(string separador, array partes);
string join(string separador, array partes);
```

As duas funções são idênticas. Retornam uma string contendo todos os elementos do array fornecido separados pela string também fornecida.

```
Exemplo:
$partes = array("a", "casa número", 13, "é azul");
$inteiro = join(" ",$partes);

$inteiro passa a conter a string:
"a casa número 13 é azul"
```

split

```
array split(string padrao, string str, int [limite]);
```

Retorna um array contendo partes da string fornecida separadas pelo padrão fornecido, podendo limitar o número de elementos do array.

```
Exemplo:
$data = "11/14/1975";
$data_array = split("/",$data);

O código acima faz com que a variável $data_array receba o valor:
array(11,14,1975);
```

explode

```
array explode(string padrao, string str);
```

Funciona de maneira bastante semelhante à função `split`, com a diferença que não é possível estabelecer um limite para o número de elementos do array.

Comparações entre strings

similar_text

```
int similar_text(string str1, string str2, double [porcentagem]);
```

Compara as duas strings fornecidas e retorna o número de caracteres coincidentes. Opcionalmente pode ser fornecida uma variável, passada por referência *(ver tópico sobre funções)*, que receberá o valor percentual de igualdade entre as strings. Esta função é *case sensitive*, ou seja, maiúsculas e minúsculas são tratadas como diferentes.

```
Exemplo:

$num = similar_text("teste", "testando",&$porc);

As variáveis passam a ter os seguintes valores:

$num == 4; $porc == 61.538461538462
```

strcasecmp

```
int strcasecmp(string str1, string str2);
```

Compara as duas strings e retorna 0 (zero) se forem iguais, um valor maior que zero se `str1 > str2`, e um valor menor que zero se `str1 < str2`. Esta função é *case insensitive*, ou seja, maiúsculas e minúsculas são tratadas como iguais.

strcmp

```
int strcasecmp(string str1, string str2);
```

Funciona de maneira semelhante à função `strcasecmp`, com a diferença que esta é *case sensitive*, ou seja, maiúsculas e minúsculas são tratadas como diferentes.

strstr

```
string strstr(string str1, string str2);
string strchr(string str1, string str2);
```

As duas funções são idênticas. Procura a primeira ocorrência de `str2` em `str1`. Se não encontrar, retorna uma string vazia, e se encontrar retorna todos os caracteres de `str1` a partir desse ponto.

```
Exemplo:

strstr("Mauricio Vivas", "Viv"); // retorna "Vivas"
```

stristr

```
string strstr(string str1, string str2);
```

Funciona de maneira semelhante à função `strstr`, com a diferença que esta é *case insensitive*, ou seja, maiúsculas e minúsculas são tratadas como iguais.

strpos

```
int strpos(string str1, string str2, int [offset] );
```

Retorna a posição da primeira ocorrência de `str2` em `str1`, ou zero se não houver. O parâmetro opcional `offset` determina a partir de qual caracter de str1 será efetuada a busca. Mesmo utilizando o offset, o valor de retorno é referente ao início de str1.

strrpos

```
int strrpos(string haystack, char needle);
```

Retorna a posição da última ocorrência de `str2` em `str1`, ou zero se não houver.

Funções para edição de strings

chop

```
string chop(string str);
```

Retira espaços e linhas em branco do final da string fornecida.

```
Exemplo:
chop("   Teste      \n  \n  "); // retorna "   Teste"
```

ltrim

```
string ltrim(string str);
```

Retira espaços e linhas em branco do final da string fornecida.

```
Exemplo:
ltrim("   Teste  \n  \n "); // retorna "Teste  \n  \n"
```

trim

```
string trim(string str);
```

Retira espaços e linhas em branco do início e do final da string fornecida.

```
Exemplo:
trim("   Teste  \n  \n "); // retorna "Teste"
```

strrev

```
string strrev(string str);
```

Retorna a string fornecida invertida.

```
Exemplo:

strrev("Teste"); // retorna "etseT"
```

strtolower

```
string strtolower(string str);
```

Retorna a string fornecida com todas as letras minúsculas.

```
Exemplo:

strtolower("Teste"); // retorna "teste"
```

strtoupper

```
string strtoupper(string str);
```

Retorna a string fornecida com todas as letras maiúsculas.

```
Exemplo:

strtolower("Teste"); // retorna "TESTE"
```

ucfirst

```
string ucfirst(string str);
```

Retorna a string fornecida com o primeiro caracter convertido para letra maiúscula.

```
Exemplo:

ucfirst("teste de funcao"); // retorna "Teste de funcao"
```

ucwords

```
string ucwords(string str);
```

Retorna a string fornecida com todas as palavras iniciadas por letras maiúsculas.

```
Exemplo:

ucwords("teste de funcao"); // retorna "Teste De Funcao"
```

str_replace

```
string str_replace(string str1, string str2, string str3);
```

Altera todas as ocorrências de `str1` em `str3` pela string `str2`.

Funções diversas

chr

```
string chr(int ascii);
```

Retorna o caracter correspondente ao código ASCII fornecido.

ord

```
int ord(string string);
```

Retorna o código ASCII correspontente ao caracter fornecido.

echo

```
echo(string arg1, string [argn]... );
```

Imprime os argumentos fornecidos.

print

```
print(string arg);
```

Imprime o argumento fornecido.

strlen

```
int strlen(string str);
```

Retorna o tamanho da string fornecida.

Arrays

Arrays em PHP podem ser observados como mapeamentos ou como vetores indexados. Mais precisamente, um valor do tipo array é um dicionário onde os índices são as chaves de acesso. Vale ressaltar que os índices podem ser valores de qualquer tipo e não somente inteiros. Inclusive, se os índices forem todos inteiros, estes não precisam formar um intervalo contínuo Como a checagem de tipos em PHP é dinâmica, valores de tipos diferentes podem ser usados como índices de array, assim como os valores mapeados também podem ser de diversos tipos.

```
Exemplo:
<?
$cor[1] = "vermelho";
$cor[2] = "verde";
$cor[3] = "azul";
$cor["teste"] = 1;
?>

Equivalentemente, pode-se escrever:

<?
$cor = array(1 => "vermelho, 2 => "verde, 3 => "azul",
"teste => 1);
?>
```

Listas

As listas são utilizadas em PHP para realizar atribuições múltiplas. Através de listas é possível atribuir valores que estão num array para variáveis. Vejamos o exemplo:

```
Exemplo:

 list($a, $b, $c) = array("a", "b", "c");
```

O comando acima atribui valores às três variáveis simultaneamente. É bom notar que só são atribuídos às variáveis da lista os elementos do array que possuem índices inteiros e não negativos. No exemplo acima as três atribuições foram bem sucedidas porque ao inicializar um array sem especificar os índices eles passam a ser inteiros, a partir do zero. Um fator importante é que cada variável da lista possui um índice inteiro e ordinal, iniciando com zero, que serve para determinar qual valor será atribuído. No exemplo anterior temos $a com índice 0, $b com índice 1 e $c com índice 2. Vejamos um outro exemplo:

```
$arr = array(1=>"um",3=>"tres","a"=>"letraA",2=>"dois);
list($a,$b,$c,$d) = $arr;
```

Após a execução do código acima temos os seguintes valores:

```
$a == null
$b == "um"
$c == "dois"
$d == "tres"
```

Devemos observar que à variável $a não foi atribuído valor, pois no array não existe elemento com índice 0 (zero). Outro detalhe importante é que o valor "tres" foi atribuído à variável $d, e

não a $b, pois seu índice é 3, o mesmo que $d na lista. Por fim, vemos que o valor "letraA" não foi atribuído a elemento algum da lista pois seu índice não é inteiro.
Os índices da lista servem apenas como referência ao interpretador PHP para realizar as atribuições, não podendo ser acessados de maneira alguma pelo programador. De maneira diferente do array, uma lista não pode ser atribuída a uma variável, servindo apenas para fazer múltiplas atribuições através de um array.

Funções para Array

Array

```
array array(...);
```

É a função que cria um array a partir dos parâmetros forncidos. É possível fornecer o índice de cada elemento. Esse índice pode ser um valor de qualquer tipo, e não apenas de inteiro. Se o índice não for fornecido o PHP atribui um valor inteiro sequencial, a partir do 0 ou do último índice inteiro explicitado. Vejamos alguns exemplos:

```
Exemplo 1
$teste = array("um", "dois","tr"=>"tres",5=>"quatro","cinco");

Temos o seguinte mapeamento:
0 => "um" (0 é o primeiro índice, se não houver um explicito)
1 => "dois" (o inteiro seguinte)
"tr" => "tres"
5 => "quatro" (valor explicitado)
6 => "cinco" (o inteiro seguinte ao último atribuído, e não o
próximo valor, que seria 2)

Exemplo 2
$teste                          =                          array("um",
6=>"dois","tr"=>"tres",5=>"quatro","cinco");

Temos o seguinte mapeamento:
0 => "um"
6 => "dois"
"tr" => tres
5 => "quatro" (seria 7, se não fosse explicitado)
7 => "cinco" (seria 6, se não estivesse ocupado)
```

Em geral, não é recomendável utilizar arrays com vários tipos de índices, já que isso pode confundir o programador. No caso de realmente haver a necessidade de utilizar esse recurso, deve-se ter bastante atenção ao manipular os índices do array.

range

```
array range(int minimo, int maximo);
```

A função range cria um array cujos elementos são os inteiros pertencentes ao intervalo fornecido, inclusive. Se o valor do primeiro parâmetro for maior do que o do segundo, a função retorna `false` (valor vazio).

shuffle

```
void shuffle(array &arr);
```

Esta função "embaralha" o array, ou seja, troca as posições dos elementos aleatoriamente e não retorna valor algum.

sizeof

```
int sizeof(array arr);
```

Retorna um valor inteiro contendo o número de elementos de um array. Se for utilizada com uma variável cujo valor não é do tipo array, retorna 1. Se a variável não estiver setada ou for um array vazio, retorna 0.

Funções de "navegação"

Toda variável do tipo array possui um ponteiro interno indicando o próximo elemento a ser acessado no caso de não ser especificado um índice. As funções seguintes servem para modificar esse ponteiro, permitindo assim percorrer um array para verificar seu conteúdo (chaves e elementos).

reset

```
mixed reset(array arr);
```

Seta o ponteiro interno para o primeiro elemento do array, e retorna o conteúdo desse elemento.

end

```
mixed end(array arr);
```

Seta o ponteiro interno para o último elemento do array, e retorna o conteúdo desse elemento.

next

```
mixed next(array arr);
```

Seta o ponteiro interno para o próximo elemento do array, e retorna o conteúdo desse elemento.
Obs.: esta não é uma boa função para determinar se um elemento é o último do array, pois pode retornar `false` tanto no final do array como no caso de haver um elemento vazio.

prev

```
mixed prev(array arr);
```

Seta o ponteiro interno para o elemento anterior do array, e retorna o conteúdo desse elemento. Funciona de maneira inversa a `next`.

pos

```
mixed pos(array arr);
```

Retorna o conteúdo do elemento atual do array, indicado pelo ponteiro interno.

key

```
mixed key(array arr);
```

Funciona de maneira bastante semelhante a `pos`, mas ao invés de retornar o elemento atual indicado pelo ponteiro interno do array, retorna seu índice.

each

```
array each(array arr);
```

Retorna um array contendo o índice e o elemento atual indicao pelo ponteiro interno do array. o valor de retorno é um array de quatro elementos, cujos índices são 0, 1, "key" e "value". Os elementos de índices 0 e "key" armazenam o índice do valor atual, e os elementos de índices 1 e "value" contém o valor do elemento atual indicado pelo ponteiro.
Esta função pode ser utilizada para percorrer todos os elementos de um array e determinar se já foi encontrado o último elemento, pois no caso de haver um elemento vazio, a função não retornará o valor `false`. A função `each` só retorna `false` depois q o último elemento do array foi encontrado.

Exemplo:

```
/*função que percorre todos os elementos de um array e imprime seus índices e valores */
function imprime_array($arr) {
  reset($arr);
  while (list($chave,$valor) = each($arr))
    echo "Chave: $chave. Valor: $valor";
}
```

Funções de ordenação

São funções que servem para arrumar os elementos de um array de acordo com determinados critérios. Estes critérios são: manutenção ou não da associação entre índices e elementos; ordenação por elementos ou por índices; função de comparação entre dois elementos.

sort

```
void sort(array &arr);
```

A função mais simples de ordenação de arrays. Ordena os elementos de um array em ordem crescente, sem manter os relacionamentos com os índices.

rsort

```
void rsort(array &arr);
```

Funciona de maneir ainversa à função `sort`. Ordena os elementos de um array em ordem decrescente, sem manter os relacionamentos com os índices.

asort

```
void asort(array &arr);
```

Tem o funcionamento bastante semelhante à função `sort`. Ordena os elementos de um array em ordem crescente, porém mantém os relacionamentos com os índices.

arsort

```
void arsort(array &arr);
```

Funciona de maneira inversa à função `asort`. Ordena os elementos de um array em ordem decrescente e mantém os relacionamentos dos elementos com os índices.

ksort

```
void ksort(array &arr);
```

Função de ordenação baseada nos índices. Ordena os elementos de um array de acordo com seus índices, em ordem crescente, mantendo os relacionamentos.

usort

```
void usort(array &arr, function compara);
```

Esta é uma função que utiliza outra função como parâmetro. Ordena os elementos de um array sem manter os relacionamentos com os índices, e utiliza para efeito de comparação uma função definida pelo usuário, que deve comparar dois elementos do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

uasort

```
void uasort(array &arr, function compara);
```

Esta função também utiliza outra função como parâmetro. Ordena os elementos de um array e mantém os relacionamentos com os índices, utilizando para efeito de comparação uma função definida pelo usuário, que deve comparar dois elementos do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

uksort

```
void uksort(array &arr, function compara);
```

Esta função ordena o array através dos índices, mantendo os relacionamentos com os elementos., e utiliza para efeito de comparação uma função definida pelo usuário, que deve comparar dois índices do array e retornar 0, 1 ou –1, de acordo com qualquer critério estabelecido pelo usuário.

Objetos

Um objeto pode ser inicializado utilizando o comando *new* para instanciar uma classe para uma variável.

```
Exemplo:
class teste {
          function nada() {
                echo "nada";
          }
}

$vivas = new teste;
$vivas -> nada();
```

A utilização de objetos será mais detalhada mais à frente.

Booleanos

PHP não possui um tipo booleano, mas é capaz de avaliar expressões e retornar *true* ou *false*, através do tipo `integer`: é usado o valor 0 (zero) para representar o estado *false*, e qualquer valor diferente de zero (geralmente 1) para representar o estado *true*.

Transformação de tipos

A transformação de tipos em PHP pode ser feita das seguintes maneiras:

Coerções

Quando ocorrem determinadas operações (“+”, por exemplo) entre dois valores de tipos diferentes, o PHP converte o valor de um deles automaticamente (coerção). É interessante notar que se o operando for uma variável, seu valor não será alterado.
O tipo para o qual os valores dos operandos serão convertidos é determinado da seguinte forma: Se um dos operandos for `float`, o outro será convertido para `float`, senão, se um deles for `integer`, o outro será convertido para `integer`.

```
Exemplo:
$vivas = “1”;        // $vivas é a string “1”
$vivas = $vivas + 1; // $vivas é o integer 2
$vivas = $vivas + 3.7;// $vivas é o double 5.7
$vivas = 1 + 1.5          // $vivas é o double 2.5
```

Como podemos notar, o PHP converte `string` para `integer` ou `double` mantendo o valor. O sistema utilizado pelo PHP para converter de *strings* para números é o seguinte:

- É analisado o início da `string`. Se contiver um número, ele será avaliado. Senão, o valor será 0 (zero);
- O número pode conter um sinal no início (“+” ou “-“);
- Se a `string` contiver um ponto em sua parte numérica a ser analisada, ele será considerado, e o valor obtido será `double`;
- Se a `string` contiver um “e” ou “E” em sua parte numérica a ser analisada, o valor seguinte será considerado como expoente da base 10, e o valor obtido será `double`;

```
Exemplos:
$vivas = 1 + “10.5”;      // $vivas == 11.5
$vivas = 1 + “-1.3e3”;    // $vivas == -1299
$vivas = 1 + “teste10.5”; // $vivas == 1
$vivas = 1 + “10testes”;  // $vivas == 11
$vivas = 1 + "  10testes";     // $vivas == 11
$vivas = 1 + "+  10testes";    // $vivas == 1
```

Transformação explícita de tipos

A sintaxe do *typecast* de PHP é semelhante ao C: basta escrever o tipo entre parenteses antes do valor

```
Exemplo:
$vivas = 15;              // $vivas é integer (15)
$vivas = (double) $vivas  // $vivas é double (15.0)
```

```
$vivas = 3.9                // $vivas é double (3.9)
$vivas = (int) $vivas       // $vivas é integer (3)
                            // o valor decimal é truncado
```

Os tipos de *cast* permitidos são:
(int), (integer) ⇒ muda para integer;
(real), (double), (float) ⇒ muda para float;
(string) ⇒ muda para string;
(array) ⇒ muda para array;
(object) ⇒ muda para objeto.

Com a função settype

A função `settype` converte uma variável para o tipo especificado, que pode ser "integer", "double", "string", "array" ou "object".

```
Exemplo:
$vivas = 15;                // $vivas é integer
settype($vivas,double)      // $vivas é Double
```

ANEXO II
Gravação de Imagens dentro do Banco de Dados

PASSO 1: Criação dos campos na tabela do MySQL

A seguir serão informados os campos básicos necessários para que possamos fazer a gravação dos dados binários e as informações da imagem necessárias para visualizá-la posteriormente.

Nome dos Campos (exemplo de nomes)	Tipo	Tamanho
binarioFoto	LONGBLOB	-
tipoFoto	VARCHAR	20

Preste atenção nos tipos binários diponíveis no MySQL, conforme tabela abaixo:

Tipo do Campo Binário	Tamanho	Tamanho Máximo Permitido para Imagem
TINYBLOB	2^8 bytes	256 bytes
BLOB	2^{16} bytes	64 Kbytes
MEDIUMBLOB	2^{24} bytes	16.384 Kbytes
LONGBLOB	2^{32} bytes	4.194.304 Kbytes

Você deverá utilizar o campo que satisfaça sua necessidade. Caso vá criar um site onde o cliente apenas utilizará num determinado campo logotipos que serão relativamente pequenos, verifique qual o melhor tipo a ser utilizado, assim, você estará aumentando a agilidade de seu banco de dados no momento da execução do SQL.

PASSO 2: Criação do formulário para envio da imagem

A imagem a ser gravada no banco de dados deve ser adquirida da seguinte forma:

```
<!-- **** Nome do arquivo: usuarios_edicao.php **** -->
<html>
<head>
<title>Entrada de Imagens</title>
</head>
<body>
<form enctype='multipart/form-data' action='usuarios_salvar.php'
method='POST'>
<input type='file' name='f_foto_usuarios' size='50'>
<input type='submit' value='Gravar'>
</form>
</body>
</html>
```

Lembre-se que no arquivo de configuração do PHP – php.ini – existe uma configuração que limita o tamanho máximo do upload de arquivos por POST, além de outras informações importantes sobre este assunto, conforme abaixo:

```
;;;;;;;;;;;;;;;;
; File Uploads ;
;;;;;;;;;;;;;;;;
; Whether to allow HTTP file uploads.
file_uploads = On
; Temporary directory for HTTP uploaded files (will use system default if not
; specified).
```

```
upload_tmp_dir = /tmp
; Maximum allowed size for uploaded files.
upload_max_filesize = 2M
```

No exemplo acima, o upload de arquivos por meio de POST é de 2 megabytes e o arquivo temporário será gravado em /tmp.

PASSO 3: Salvamento da imagem no banco de dados

No momento do salvamento da imagem, no exemplo abaixo, podemos colocar o tamanho máximo da imagem que queremos gravar, para que o usuário não faça um upload imenso e deixe o banco de dados “inchado”.

```
<?PHP
/*
**** Nome do arquivo: usuarios_salvar.php ****
*/
// VERIFICA TAMANHO DA IMAGEM (Este teste não é obrigatório)
if(is_uploaded_file($_FILES['f_foto_usuarios']['tmp_name'])) {
if($_FILES['f_foto_usuarios']['size']>(16000*1024)) { // se for maior de 16Mb
echo 'Arquivo de imagem deve ser menor que 16Mb!';
}
}
// INÍCIO: UPLOAD IMAGEM
if(is_uploaded_file($_FILES['f_foto_usuarios']['tmp_name'])) {
$imgData = file_get_contents($_FILES['f_foto_usuarios']['tmp_name']);
$sizeData = getimagesize($_FILES['f_foto_usuarios']['tmp_name']);
$foto_usuarios = $imgData;
$tipo_foto_usuarios = $sizeData['mime'];
/*
Neste momento você deverá fazer a gravação no banco
através do SQL respectivo, salvando nos campos
foto_usuarios com o conteúdo de $imgData
tipo_foto_usuarios com o conteúdo de $sizeData['mime']
*/

    $imagem = $imgData;
    $tipo = $sizeData['mime'];

//$fp = fopen($f_foto_usuarios,"rb");
//$imagem_temp = fread($fp,filesize($f_foto_usuarios));
//fclose($fp);

$imagem_temp = addslashes($imagem);

    $conexao = mysql_pconnect("localhost","root","");
    mysql_select_db("imagem",$conexao);
    $query = "INSERT INTO teste (tipoFoto,binarioFoto) VALUES ('$tipo','$imagem_temp')";
    //echo $query.'<br>';
        mysql_query($query,$conexao);
        echo "Imagem gravada com sucesso!";
}
// FIM: UPLOAD IMAGEM
?>
```

Ao fim deste procedimento – se tudo estiver ocorrido como esperado –, você terá a imagem gravada no banco de dados corretamente.

PASSO 4: Visualização da imagem gravada

A exibição da imagem é muito simples, como você poderá ver abaixo:

```
<?PHP
  $conexao = mysql_pconnect("localhost","root","");
  mysql_select_db("imagem",$conexao);
  $query = "SELECT * FROM teste limit 1 ";
$resultado = mysql_query($query,$conexao);

if($linha = mysql_num_rows($resultado)== 1)
{
$registro = mysql_fetch_array($resultado);
if($registro['tipoFoto']=="image/jpeg");
   $ext=’jpg’;
if($registro['tipoFoto']=="image/gif");
   $ext=’gif’;
if($registro['tipoFoto']=="image/png");
   $ext=’png’;
if($registro['tipoFoto']=="image/bmp);
   $ext=’bmp’;


$filename= 'temp.' . $ext;
$tempfile = fopen($filename, "w+");
fwrite($tempfile, mysql_result($resultado, 0, 'binarioFoto'));
fclose($tempfile);
echo '<img src='.$filename.'>';
}

?>
```