

ESCOLA TÉCNICA DE BRASÍLIA
COORDENAÇÃO DE CURSOS BÁSICOS



PROFESSOR: LUCIANO FERNANDES DA SILVA

AULA 1

Introdução

Aplicações WEB são empolgantes! Imagine você, como um programador WEB, criando sua aplicação para poder ser utilizada por milhares de usuários do planeta. Pense no lucro que você pode obter com isso. Mas, para aproveitar as vantagens e facilidades que a WEB oferece, você precisa dominar uma linguagem de programação WEB. Você precisa saber PHP. Porque somente com HTML você não faz nada além de sítios bonitinhos, mas sem graça e limitados.

OBJETIVOS

- Arquitetura de aplicações WEB
 - Funcionamento e necessidade do servidor WEB
 - A necessidade do interpretador de comandos PHP
- Compreender a sintaxe básica de construções da linguagem
 - Marcações PHP
 - Variáveis
 - Constantes
 - Operadores
- Compreender os mecanismos de entrada e saída
 - A funcionalidade dos métodos HTTP mais comuns (essencialmente o GET e o POST)
 - Mecanismos de passagem de parâmetros pelos métodos GET e POST
 - Construção de formulários WEB

Qual a principal tarefa do servidor WEB?

O servidor web atende a requisição do cliente e devolve algo.

Através do *navegador web* o cliente consegue realizar uma requisição a um recurso presente no *servidor web*. Geralmente esse recurso é um arquivo de áudio, imagem, html, pdf, entre muitos outros formatos padrão. Caso o recurso não exista, o servidor devolve código e mensagem de erro.

Como o servidor WEB gerência programas PHP?

O servidor web sabe o que fazer requisições para com recursos estáticos. Receber a requisição para um recurso e devolver o recurso, caso exista. Para saber lidar com recursos dinâmicos, como programas PHP, o servidor web precisa de configuração adicional para integração com o interpretador PHP. Nosso ambiente de desenvolvimento para o curso básico utiliza o servidor Apache como servidor web.

A necessidade do interpretador de comandos PHP

Sabemos que o servidor web por si só não reconhece PHP. Mas, ele sabe repassar o recurso solicitado para um programa que reconhece instruções em linguagem PHP. O famoso interpretador de comandos PHP. Ele quem executa o texto fonte do programa PHP e devolve, para o servidor web, o resultado. O servidor web quem, por sua vez, devolve o resultado final ao usuário.

Um programa PHP simplista

Considere que você precisa exibir a data e hora de entrada do usuário na sua página.

```
<html>

<head>

    <title>Primeiro programa</title>

</head>

<body>

<h2>

<?php

    print date("d-m-Y H:i:s");

?>

</h2>

</body>

</html>
```

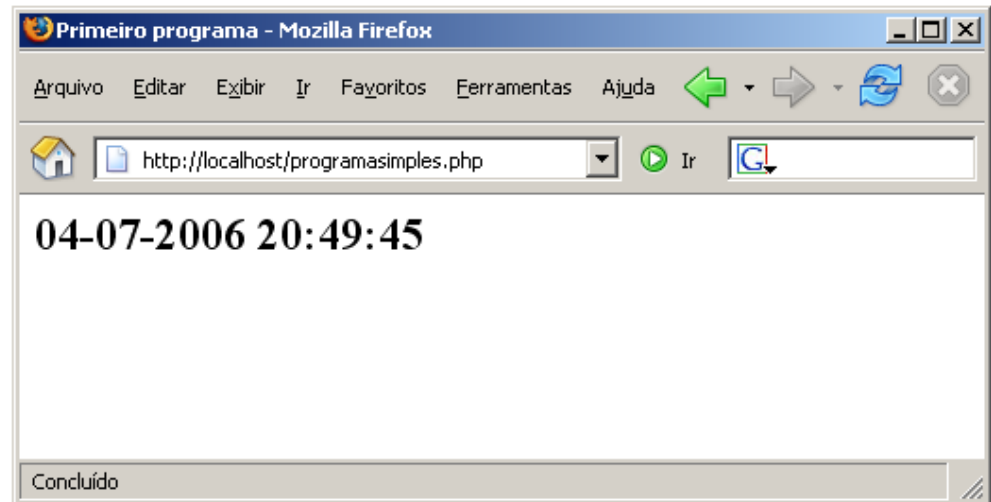
Exceto o conteúdo entre as marcações **<?php** e **?>**, temos uma simples página HTML. Quando o interpretador encontra o símbolo **<?php**, ele entende que deve executar os comandos que seguem esta marcação. No exemplo, a chamada a uma

função que devolve a data/hora atual. Até encontrar o fim da seção, que termina na marcação **?>**. Daí em diante, o conteúdo estático continua a ser enviado ao navegador do usuário.

Testando o programa

Após criado o programa, o procedimento para teste é bastante simples. O único detalhe é certificar se o programa PHP foi salvo na pasta **htdocs** do servidor WEB Apache. Além disso o Apache esteja “rodando” (em execução). Feito isso, acesse o programa PHP diretamente pelo navegador web no endereço: **http://localhost/programasimples.php**

O resultado apresentado deve ser semelhante a este.



Variáveis

Como sabemos, uma variável é um lugar onde guardamos informação. Você pode atribuir um valor a ela, e ler um valor a partir dela. Em qualquer contexto que PHP encontrar uma variável em expressões, o seu valor será usado. Por exemplo, vamos criar uma página que utiliza valores de variáveis numéricas.

```
<html>

<head>

    <title>Uso das variáveis</title>

</head>

<body>

<?php

//isto é um comentário, o interpretador PHP irá ignorar essa linha.

$a = 10;

$b = 2;
```

```

print "a = $a" . "<br>";

print "b = $b" . "<br>";

print "A = $A" . "<br>";

print "B = $B" . "<br>";

?>

</body>

</html>

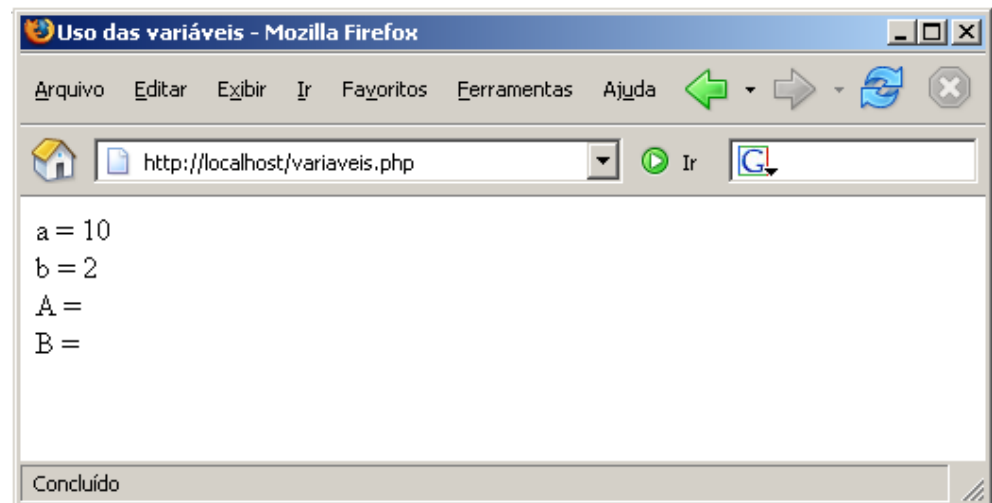
```

Percebemos algumas características sintáticas importantes, da PHP, nesta página:

- Comentários de linha única iniciam com **duas barras**
- As variáveis são precedidas pelo caractere **cifrão**
- Cada comando deve terminar com um **ponto-e-vírgula**
- Você deve usar um **ponto** para concatenar texto

Uma observação sobre variáveis. Na PHP elas são sensíveis ao caso. Ou seja, a variável \$a é diferente da variável \$A. Assim como a variável **\$USUARIO** é diferente da variável **\$usuario**.

Visualize o resultado no navegador WEB para melhor entendimento da saída gerada.



Constantes

Constantes são um tipo particular de variáveis. Variáveis podem ter seus valores alterados a qualquer instante, de acordo com a necessidade. Mas algumas vezes queremos criar uma variável com a expectativa de que o valor dela não mudará. Esse é o conceito de uma constante. Por exemplo, podemos definir uma constante que representa o título da página.

```

<?php

// A instrução abaixo define a constante TITULO_PAGINA

```

```

define("TITULO_PAGINA", "Programação PHP");

?>

<html>

<head>

<title><?= TITULO_PAGINA ?></title>

</head>

<body>

    Testando o uso de constants PHP...

</body>

</html>

```

Operadores

Os operadores existem, basicamente, para agirem sobre literais (valores constantes) ou variáveis, resultando num valor final da expressão. Para facilitar o aprendizado, podemos dividir os operadores em categorias: **aritméticos**, **atribuição**, **lógicos** e **relacionais**.

Vejam os tabelas com todos os operadores na linguagem PHP em ordem de precedência. Não se faz necessário memorizar toda esta tabela. Pois, com a prática regular da programação em PHP, isso se tornará natural até para os iniciantes.

Tabela com todos os operadores em ordem de precedência

Associação	Operador
não associativo	new
direita	[
direita	! ~ ++ -- (int) (float) (string) (array) (object) @
esquerda	* / %
esquerda	+ - .
esquerda	<< >>
não associativo	< <= > >=
não associativo	== != === !==
esquerda	&
esquerda	^
esquerda	
esquerda	&&
esquerda	
esquerda	? :

Associação	Operador
direita	= += -= *= /= .= %= &= = ^= <<= >>=
direita	print
esquerda	and
esquerda	xor
esquerda	or
esquerda	,

Tabela com exemplos de operadores Aritméticos

Exemplo	Nome	Resultado
\$a + \$b	Adição	Soma de \$a e \$b
\$a - \$b	Subtração	Diferença entre \$a e \$b
\$a * \$b	Multiplicação	Produto de \$a e \$b
\$a / \$b	Divisão	Quociente de \$a por \$b
\$a % \$b	Módulo	Resto de \$a dividido por \$b
++\$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

Tabela com exemplos de operadores de atribuição e comparação

Exemplo	Nome	Resultado
\$a = 10.25	Atribuição	Armazena o valor 10.25 na variável \$a
\$a == \$b	Igual	Verdadeiro (TRUE) se \$a é igual a \$b
\$a != \$b	Diferente	Verdadeiro se \$a não é igual a \$b
\$a < \$b	Menor que	Verdadeiro se \$a é estritamente menor que \$b
\$a > \$b	Maior que	Verdadeiro se \$a é estritamente maior que \$b
\$a <= \$b	Menor ou igual	Verdadeiro se \$a é menor ou igual a \$b
\$a >= \$b	Maior ou igual	Verdadeiro se \$a é maior ou igual a \$b

Tabela com exemplos de operadores lógicos

Exemplo	Nome	Resultado
\$a && \$b	E	Verdadeiro se tanto \$a quanto \$b são verdadeiros

Exemplo	Nome	Resultado
<code>\$a \$b</code>	OU	Verdadeiro se \$a ou \$b são verdadeiros
<code>!\$a</code>	NOT	Verdadeiro se \$a não é verdadeiro

Passagem de parâmetros usando URLs

Realizaremos a passagem de parâmetros GET através de URLs, ou **Uniform Resource Locators**, que provavelmente você já conhece bem. Além da URL identificar de forma única um recurso ela também permite a passagem de informações, ou parâmetros, adicionais. Segue exemplo de formato detalhado de URL com passagem de parâmetros.

`http://pt.wikipedia.org/w/index.php?title=PHP&printable=yes`

Protocolo: `http://`

Domínio: `pt.wikipedia.org`

Caminho do programa PHP: `w/index.php`

Parâmetro 1: `title`

Valor do parâmetro 1: `PHP`

Parâmetro 2: `printable`

Valor do parâmetro 2: `yes`

Parâmetros transmitidos pelo método GET são acessíveis no programa PHP através da variável array predefinida **`$_GET`**, seguida do nome do parâmetro. A seguir criaremos programa PHP que ilustra a manipulação de parâmetros GET.

```
<html>

<head>

    <title>Parametros GET</title>

</head>

<body>

<?php

$nome  = $_GET["nome"];

$idade = $_GET["idade"];

print "<font size='5'>";

print "<i>$nome</i> possui <i>$idade</i> anos.";

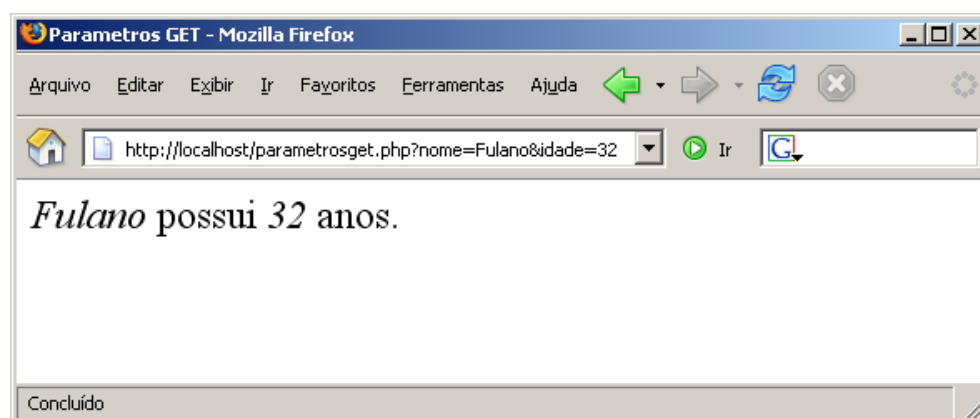
print "</font>";

?>

</body>
```

```
</html>
```

Visualize o resultado no navegador WEB para melhor entendimento da saída gerada. Observe atentamente como os parâmetros GET esperados pelo programa acima foram passados na URL abaixo.



Resultado da execução do script

Passagem de parâmetros usando Formulários

Formulários web são o meio mais utilizado para captação de dados por programas PHP. Os formulários web são escritos em linguagem HTML. Quando o usuário envia o formulário, na maioria das vezes através de um botão, o navegador web envia um array de valores ao servidor web. Nessa seção, veremos como acessar os valores contidos nesse array.

Exemplo de formulário de login

```
<html>

<head><title>Trabalhando com forms</title></head>

<body>

<h2>Página de Login</h2>

<form action="parametrospost2.php" method="POST">

<table>

<tr>

    <td>Login</td>

    <td><input type="text" name="txtlogin"/></td>

</tr>

<tr>

    <td>Senha</td>

    <td><input type="password" name="txtsenha"/></td>
```

```
</tr>

<tr>

    <td>&nbsp;</td>

    <td><input type="submit" value="OK"/></td>

</tr>

</form>

</body>

</html>
```

```
<html>

<body>

<?php

$login = $_POST["txtlogin"];

$senha = $_POST["txtsenha"];


print "Seu login: <b>$login</b><br>";

print "Sua senha: <b>$senha</b><br>";

?>

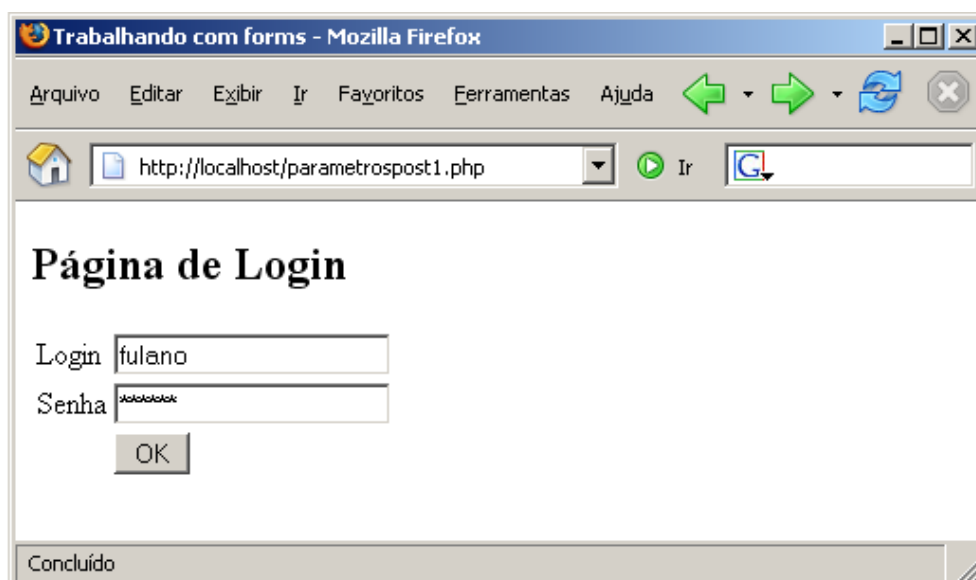
</body>

</html>
```

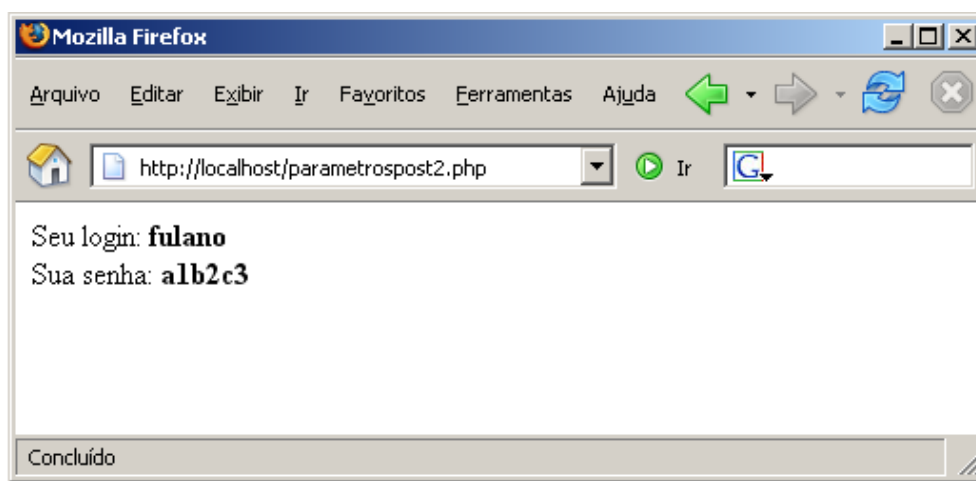
Como funciona o envio de parâmetros POST

Os atributos **action** e **method** do formulário são importantes para o perfeito funcionamento. O valor do atributo **action** informa qual o programa PHP receberá as informações do formulário. Já o valor do atributo **method**, por sua vez, informa qual o método de requisição HTTP utilizado. Neste exemplo, POST.

Os valores do formulário estão presentes na variável array predefinido `$_POST`. Seu uso é praticamente idêntico ao da variável `$_GET` estudada anteriormente. Estudaremos mais sobre variáveis array em aula futura. Por agora, concentre-se na resposta que aparecerá no navegador web. Abaixo temos as telas que detalham ainda mais o funcionamento do programa transcrito acima.



Página com formulário para entrada de dados



Página de resultado

Exercício Resolvido

Problema: Conversão de Temperatura

Criar um programa que recebe temperatura em graus Fahrenheit e apresenta a conversão da mesma para graus Celsius.

Solução:

```
<html>

<head>

    <title>Conversão de Temperaturas</title>

</head>

<body>

<h3>Conversor de Temperatura</h3>

<form action="temperatura2.php" method="POST">
```

```
<table>

  <tr>

    <td>Temperatura em F</td>

    <td><input type="text" name="tempF" value=""
maxlength="5"/></td>

  </tr>

  <tr>

    <td colspan="2" align="right">

      <input type="submit" value="Calcular" name="btnAcao"/>

    </td>

  </tr>

</table>

</form>

</body>

</html>
```

```
<html>

<body>

<?php

$tempF = $_POST["tempF"];

$tempC = ($tempF - 32) * (5.0/9.0);

print "<h3>";

print "$tempF Fahrenheit equivale a $tempC Celsius";

print "</h3>";

?>

</body>

</html>
```

Exercícios Propostos

Exercício 1

Escolha a opção equivalente ao resultado da execução do seguinte programa?

```
<?php  
  
$a = 10;  
  
$b = 12;  
  
$a = $b = $c = 2;  
  
$b = ++$a;  
  
$c = $b++;  
  
print $a . " " . $b . " " . --$c;  
  
?>
```

- ☐ 10 12 2
- ☐ 3 4 3
- ☐ 3 4 2
- ☐ 3 4 4
- ☐ Nenhuma das alternativas

Exercício 2

Ler dois valores reais a partir de um formulário, calcular e imprimir na tela:

a) A soma destes valores b) O produto deles c) O quociente entre eles.

Exercício 3

Calcular a quantidade de dinheiro gasta por um fumante. Dados: o número de anos que ele fuma, o nº de cigarros fumados por dia e o preço de uma carteira.

Exercício 4

Tendo como dados de entrada a altura de uma pessoa, construa um programa que calcule o peso ideal, tanto para homens quanto para mulheres, utilizando as fórmulas seguintes.

Para homens: $(72.7 * h) - 58$

Para mulheres: $(62.1 * h) - 44.7$

Observação: $h = \text{altura}$

AULA 2

Controle de Fluxo

Uma tarefa essencial da programação é a tomada de decisões baseado no resultado de testes lógicos.

Os comandos de controle de fluxo são essenciais a qualquer linguagem de programação, porque governam o fluxo de execução do programa. São poderosos e ajudam a explicar a popularidade da linguagem. Em PHP, temos as estruturas de seleção simples, composta e de múltipla escolha. Além das indispensáveis estruturas de repetição.

OBJETIVOS

- Comandos de Seleção
 - Seleção simples: *if*
 - Seleção composta: *if-else*
 - Operador ternário
 - Seleção múltipla: *switch*
- Comandos de Repetição
 - Repetição com *while*
 - Repetição com *do-while*
 - Repetição com *for*
- Compreender como aplicar e associar os comandos de controle de fluxo

Seleção simples: comando IF

Caracteriza o controle de fluxo unidirecional. Se a condição for **verdadeira** (*true*) o computador executará o comando ou bloco. Um bloco são as linhas de comandos contidas entre as chaves { }. Sua sintaxe é a seguinte:

```
if(condição) {  
  
    bloco de comandos;  
  
}
```

Abaixo temos um exemplo que aplica este comando.

```
<?php  
  
$login = $_POST["txtLogin"];  
$senha = $_POST["txtSenha"];  
  
if($login == NULL || $senha == NULL) {  
  
    print "Login e/ou senha inválida!";  
  
    exit();  
  
}  
  
print "Dados de acesso OK!";  
  
?>
```

No exemplo acima evitamos que o usuário prossiga, caso não forneça a descrição do login e senha. Utilizando para isso o teste condicional **if**.

Seleção composta: comando IF-ELSE

Seu funcionamento é equivalente ao **if**. Com a vantagem de permitir o controle bidirecional. Ou seja, o programador pode executar um bloco quando a condição for **verdadeira** e outro bloco quando **falsa**. Sua sintaxe é a seguinte:

Sua sintaxe é a seguinte:

```
if(condição) {  
  
    bloco de comandos do if;  
  
}  
  
else {
```

```

        bloco de comandos do else;

    }

```

No exemplo abaixo, tomamos a decisão de enviar mensagem ao usuário informando duas situações possíveis; se um número fornecido é par ou ímpar.

```

<?php

$numero = (int) $_POST["txtNumero"];

if(($numero % 2) == 0) {

    print "O número inteiro $numero é par";

}

else {

    print "O número inteiro $numero é ímpar";

}

?>

```

Operador ternário

Este operador nada mais é do que uma maneira compacta de expressar o comando *if-else*. Conveniente em situações onde as expressões de teste são simples e compactas. Sua sintaxe é a seguinte:

(condição) ? expressão 1 : expressão 2;

O exemplo de programa PHP abaixo utiliza o operador ternário para verificar qual o maior entre dois números.

```

<?php

$x = (int) $_POST["txtx"];

$y = (int) $_POST["txty"];

$maior = ($x > $y) ? $x : $y;

print "maior valor = $maior";

?>

```

Seleção múltipla: comando SWITCH

O comando `switch`, por sua vez, é muito útil para efetuar ações baseadas em várias condições.

Seu funcionamento é o seguinte. Uma variável é testada sucessivamente contra uma lista de valores. Depois de encontrar uma igualdade, o comando ou bloco de comandos é executado. Até que o comando **`break`** seja encontrado. Se nenhuma coincidência for encontrada o comando default será executado.

Vejamos abaixo um exemplo em código que aplica o comando *switch*:

```
<?php
$voto = (int) $_POST["voto"];

switch ($voto) {

    case 1: print "Opção 1 = Lula";

            break;

    case 2: print "Opção 2 = Ciro";

            break;

    case 3: print "Opção 3 = Eneias";

            break;

    default: print "Opção inválida!";

}

?>
```

Comando while

O `while` é utilizado para a execução de um bloco de comandos repetidamente. Até que o teste condicional se torne falso e finalize a execução do bloco. Sua sintaxe é a seguinte:

```
while(condição) {

    bloco de comandos;

}
```

O processo de execução do `while` é o seguinte:

- A condição é avaliada;
- Caso a condição seja verdadeira, o bloco de comandos é executado;
- Retorna ao passo 1;
- Caso a condição seja falsa, o próximo comando após o `while` é executado.

Estes componentes do comando **while** são importantes para o entendimento dos comandos **do-while** e *for* também. Portanto, fique atento para notar as semelhanças.

Temos a seguir um exemplo que utiliza o comando **while** para imprimir os valores de um contador.

```
<?php
$c = 0;
while ($c < 10)
    print $c . "<br>";
    $c++;
}
?>
```

Comando do-while

O comando **do-while** também é executado enquanto uma condição for verdadeira, porém, o teste da condição só é feito no final. Conseqüentemente, o bloco de comandos é executado pelo menos uma vez.

Temos a seguir o mesmo exemplo anterior. Entretanto, utilizando o comando **do-while** para imprimir os valores de um contador.

```
<?php
$c = 0;
do {
    print $c . "<br>";
    $c++;
}while ($c < 10);
?>
```

Comando for

Por fim, temos o comando **for**. O **for** é um comando extremamente poderoso. Em PHP ele tem uma forma elegante e permite construções bem versáteis. A sintaxe do comando é a seguinte:

```
for(exp1; exp2; exp3) {
    bloco de comandos;
}
```

Conheça os componentes do comando **for**. Tente perceber as semelhanças com o comando **while**.

- `exp1` - zero ou mais expressões que serão executadas antes da repetição do bloco de comandos;
- `exp2` - teste lógico que, no caso de ser avaliado como falso, encerra a repetição;
- `exp3` - zero ou mais expressões que serão executadas no momento do término de uma repetição. Aqui, geralmente, realizamos operações de incrementos/decrementos.

Temos a seguir o mesmo exemplo anterior. Entretanto, utilizando o comando **for** para imprimir os valores de um contador.

```
<?php
for ($c = 0; $c < 10; $c++) {
    print $c . "<br>";
}
?>
```

Exercício Resolvido

Problema: Conversão de temperaturas

Criar um programa em que o usuário fornece dois valores, inicial e final, de temperaturas em graus Fahrenheit. O mesmo deverá apresentar, em uma tabela, a conversão das temperaturas para graus Celsius. Com incrementos de dez em dez.

Solução:

```
<html>
<head>
    <title>Conversão de Temperatura : F -> C</title>
</head>
<body>
<h3>Conversor de Temperatura</h3>
<form action="" method="POST">
<table>
    <tr>
        <td>Temperatura inicial em F</td>
        <td>
```

```

        <input type="text" name="tempInicial" value="<?=$_POST["tempInicial"] ?>" maxlength="5"/>

    </td>

</tr>

<tr>

    <td>Temperatura final em F</td>

    <td>

        <input type="text" name="tempFinal" value="<?=$_POST["tempFinal"] ?>" maxlength="5"/>

    </td>

</tr>

<tr>

    <td colspan="2" align="right">

        <input type="submit" value="Calcular" name="btnAcao"/>

    </td>

</tr>

<tr>

    <td colspan="2">&nbsp;</td>

</tr>
</table>

<?php

// verifica se acionou o botão para cálculo

if ($_POST != NULL) {

    // leitura das temperaturas

    $inicial = $_POST["tempInicial"];

    $final   = $_POST["tempFinal"];

    // validação dos parâmetros

    if ($inicial == NULL

```

```

        || $final == NULL

        || !is_numeric($inicial)

    || !is_numeric($final)) {

        print "<font color='red'>";

        print "Parâmetros inválidos. Tente novamente!";

        print "</font>";

        exit();

    }

?>

<br/>

Resultado<br/>

<hr align="left" width="40%"/>

<table border="1" width="30%">

    <tr bgcolor="#7f7fa5" align="center">

        <td>Fahrenheit</td>

        <td>Celsius</td>

    </tr>

    <?php

        // cálculo de conversão da faixa de temperaturas

        for ($t = $inicial; $t <= $final; $t += 10) {

            ?>

            <tr align="center">

                <td><?= $t ?></td>

                <td><?= number_format(($t - 32) * (5.0/9.0), 4) ?></td>

            </tr>

            <?php

                } //fim do for

            ?>

        </table>

```



```
<?php  
} //fim do if  
?>  
</form>  
</body>  
</html>
```

Exercícios Propostos

Problema 1

Ler dois números inteiros, a partir de formulário. Se o segundo for diferente de zero, calcular e imprimir o quociente do primeiro pelo segundo. Caso contrário, imprimir a mensagem “DIVISÃO POR ZERO”.

Problema 2

Ler três números inteiros, a partir de formulário, e imprimir na tela qual o maior e menor valor fornecido.

Problema 3

Escolha a opção equivalente ao resultado da execução do seguinte programa?

```
<?php  
$n = 6;  
$fatorial = 1;  
$contador = 1;  
while ($contador <= $n) {  
    $fatorial = $fatorial * $contador;  
    $contador++;  
}  
print $fatorial; //Resultado...  
?>
```

- a) 1440
- b) 760
- c) 320
- d) Nenhuma das alternativas

Problema 4

Implemente uma calculadora primitiva (operações de soma, subtração, divisão e multiplicação). Considere que o usuário irá fornecer três parâmetros de entrada: valor numérico A, valor numérico B, operação aritmética. Como saída, lhe será apresentada tela com o resultado do cálculo.

Problema 5

Imprimir a tabuada de um número qualquer fornecido pelo usuário. O resultado deve ser exibido numa tabela HTML seguindo o formato abaixo.

Expressão	Resultado
4 x 1	4
4 x 2	8
4 x 3	12
...	...

Problema 6

Elabore um programa, que calcule o que deve ser pago por um produto considerando o preço normal de etiqueta e a escolha da condição de pagamento. Utilize os códigos da tabela a seguir para ler qual a condição de pagamento escolhida e efetuar o cálculo adequado.

Código	Condição de Pagamento
1	À vista em dinheiro ou cheque, recebe 10% de desconto.
2	À vista no cartão de crédito, recebe 5% de desconto
3	Em 2 vezes, preço normal de etiqueta sem juros.
4	Em 3 vezes, preço normal de etiqueta mais juros de 10%

AULA 3

Arrays

Arrays são estruturas de dados que permitem armazenar um ou mais valores, mantidos pela mesma referência (a variável do tipo array).

Os arrays da linguagem PHP são muito poderosos e fáceis de usar. Estaremos construindo as primitivas para trabalhar com arrays e, logo em seguida, implementando algoritmos clássicos que necessitam de tal estrutura. Basicamente o estudo será voltado para os dois tipos de arrays disponíveis: *lineares* e *associativos*.

OBJETIVOS

- Arrays Lineares
- Arrays Associativos
- Arrays Multidimensionais

Arrays Lineares

Vamos entender o conceito do array linear através de um exemplo.

Considere que você é professor de uma turma de informática e precisa armazenar os nomes de todos os alunos da turma. Não seria conveniente termos uma variável para cada nome. Na realidade, seria impraticável em algumas situações. Considere que a turma inicialmente tenha quinze alunos. Assim seu programa teria quinze variáveis do tipo string para armazenar os nomes. Entretanto, foram matriculados mais cinco novos alunos. Fique sabendo que você será convocado para alterar o programa!

Ao invés disso, vamos utilizar um array linear para armazenar em um única variável todos os nomes. E, além disso, poder aumentar o tamanho da variável array para comportar novos nomes.

Segue fragmento de código PHP para ilustrar:

```
<?php

$alunos = array();

$alunos[1] = "João";
$alunos[2] = "Maria";
$alunos[3] = "José";
...
...
$alunos[15] = "Ana";

?>
```

De posse dos nomes dos alunos, podemos precisar pesquisar o nome do aluno pelo número da chamada. Vejamos como realizar esta operação:

```
<?php

print "O aluno número $num da chamada é o " . $alunos[$num];

?>
```

Também, podemos precisar imprimir uma listagem da chamada. Isso é conseguido com o seguinte fragmento de código:

```
<?php

for($c = 1; $c <= count($alunos); $c++) {

    print $c . " - " . $alunos[$c] . "<br>";

}

?>
```

Os arrays são geralmente encontrados em conjunção aos laços de repetição. O que permite o processamento automático de todos os elementos do array. Como exemplificado acima.

É importante notar as seguintes características do arrays em PHP:

- O primeiro elemento inicia no índice ou posição 0(zero), caso não seja especificado
- O valor do índice ou posição do elemento acessado deve ser especificado entre colchetes, após o nome da variável do tipo array

Arrays Associativos

Até o momento, exemplificamos array que faz uso de índices inteiros como chave para acesso aos valores. Estes são mais conhecidos como *arrays lineares*. Entretanto, você pode especificar uma chave do tipo string também. Arrays com essa característica são conhecidos como *arrays associativos*. Veja exemplo de como defini-los:

```
$dadosUsuario = array("nome" => "Fulano de Tal",

    "email" => "fulano@email.com",

    "senha" => "a1b2c3");
```

Para acessar os valores deste array, utilizamos as chaves (strings) especificadas:

```
<?php

print $dadosUsuario["nome"] . "<br>";

print $dadosUsuario["senha"] . "<br>";

print $dadosUsuario["email"] . "<br>";

?>
```

A vantagem dos arrays associativos é que, não só permite o acesso a elementos individualizados, mas também não nos obriga a conhecer qual a posição ou ordem dos elementos dentro do array. O array **\$_POST**, já estudado, é um exemplo de

array associativo já disponível ao programador PHP. Que armazena todos os valores de um formulário, usando os atributos name como chave.

Para concluir esta seção. Serão ilustradas construções de repetição para imprimir todos os valores de um array associativo.

```
<?php

// Descrição: imprimindo array associativo com auxílio do laço while

while(list($chave, $valor) = each($dadosUsuario)) {

    print "dadosUsuario[$chave] = $valor <br>";

}

?>
```

```
<?php

// Descrição: imprimindo array associativo com auxílio do laço
foreach

foreach ($dadosUsuario as $valor) {

    print "Valor: $valor <br>";

}

?>
```

```
<?php

// Descrição: imprimindo array linear com auxílio do laço foreach

$a = array(10, 20, 30, 40, 50);

foreach ($a as $v) {

    print "Valor atual = $v <br>";

}
```

```
?>
```

Arrays Multidimensionais

É importante conhecermos como criar arrays com mais de uma dimensão. A sintaxe é bastante simples e intuitiva. Basta adicionar mais um par de colchetes para cada dimensão adicional.

Vejamos um exemplo:

```
<?php

// Descrição: array linear bidimensional de inteiros

$tabela[0][0] = 10;
$tabela[0][1] = 20;
$tabela[0][2] = 30;
$tabela[1][0] = 40;
$tabela[1][1] = 50;
$tabela[1][2] = 60;

for($i = 0; $i < 3; $i++) {
    for($j = 0; $j < 3; $j++) {
        print $tabela[$i][$j] . " ";
    }
    print "<br>";
}

?>
```

Exercício Resolvido

Problema

A partir de dois arrays de inteiros A e B, construir um terceiro C que irá armazenar os elementos comuns em A e B. Imprimir os valores do array C. Um valor em cada linha.

Solução

```
<?php

$A = array(10, 20, 30, 40, 50, 60, 70, 80, 90, 100);

$B = array(25, 50, 75, 100);

$C = array();

for($i = 0; $i < count($A); $i++) {
    for($j = 0; $j < count($B); $j++) {
        if($A[$i] == $B[$j]) {
            $C[] = $A[$i];
        }
    }
}

print "Valores do array C: <br>";

for($i = 0; $i < count($C); $i++) {
    print $C[$i] . "<br>";
}

?>
```

Exercícios Propostos

Problema 1

Criar um programa que utiliza o array abaixo e imprime seus elementos na ordem inversa. Um valor em cada linha.

```
$valores = array(0, 10, 20, 30, 40, 50, 60, 70, 80, 90);
```

Problema 2

Criar um programa que utiliza o array abaixo e imprime, somente, seus elementos pares. Um valor em cada linha.

```
$valores = array(0, 10, 3, 1, 7, 55, 15, 21, 99);
```

Problema 3

- *Teste com imagens presentes no seu computador, as imagens acima são apenas para exemplificar*

Criar um programa que utiliza o array abaixo, que contém nomes de arquivos de imagens, para exibir, somente, uma imagem a cada solicitação feita à página do programa. De forma aleatória. Para isto, utilize a função `rand(0, 3)` para gerar o número da posição do elemento a ser lido do array.

```
$imagens = array("paisagem.gif", "foto.jpg", "logo.gif",  
"animacao.gif");
```

Problema 4

- *\$_SERVER é um dos arrays predefinidos da linguagem, e já possui valores para você utilizar.*

Criar um programa que imprima todo o conteúdo do array associativo **\$_SERVER** dentro de uma tabela HTML.

AULA 4

Funções

“Funções dividem grandes tarefas de computação em tarefas menores, e habilitam as pessoas a construírem sobre uma base que alguém já tenha feito ao invés de iniciar a partir de rascunhos. Funções apropriadas escondem detalhes das partes do programa que não precisam conhecê-los...”

Brian W. Kernighan e Dennis M. Ritchie

OBJETIVOS

- Em que situação usar Funções
- Vantagens
- Criando Funções
- Utilizando Funções
- Devolvendo Valor
- Recursão
- Criando Biblioteca de Funções

Em que situação usar Funções?

Quando estamos envolvidos na construção de aplicações, de qualquer tamanho ou nível de complexidade, é frequente realizarmos certos cálculos, validações, e outras operações, em que o código se repete ou se assemelha a algo que já foi feito. Nesse caso, é interessante empacotar esse código em um componente de software conhecido como função.

Vantagens

- Reutilização de códigos pré-fabricados
- Modularização do programa
- Facilidade de teste e manutenção
- O trabalho em equipe e em paralelo

Criando Funções

Antes de realizarmos alguma implementação real, faz-se necessário ilustrar a estrutura básica de qualquer função definida pelo programador.

```
function nomeDaFuncao([$parametro1, $parametro2,...])  
  
{  
  
    //comandos da função...  
  
    return $valorDeRetorno; //valor de retorno da função  
  
}
```

A seguir, temos um primeiro exemplo de código que cria uma função:

```
//validaSenha: imprime mensagem de senha válida ou inválida  
function validaSenha ($senha, $senhaConfirmacao)  
{  
  
    if($senha == $senhaConfirmacao)  
    {  
  
        print "<p>Senha ok. Obrigado.</p>";  
  
    }  
  
    else  
  
    {  
  
        print "<p>Senhas não confere. Por favor, tente novamente.</p>";  
  
    }  
  
}
```

```
}
}
```

Utilizando Funções

Partindo do ponto em que temos definidos o nome da função e o que ela faz, podemos utilizá-la em qualquer programa. Para melhor compreensão, segue exemplo de programa completo que utiliza a função acima definida.

```
<?php

//validarSenha: imprime mensagem de senha válida ou inválida

function validarSenha ($senha, $senhaConfirmacao)
{
    if ($senha == $senhaConfirmacao)
    {
        print "<P>Senha ok. Obrigado.</P>";
    }
    else
    {
        print "<P>Senhas não confere. Por favor, tente novamente.</P>";
    }
}

?>

<HTML>

<BODY>

<?php

//parâmetros advindos de um formulário

$senha1 = $_POST["senha1"];

$senha2 = $_POST["senha2"];

print "Primeira senha: " . $senha1 . "<BR/>";

print "Segunda senha: " . $senha2 . "<BR/>";
```

```

validarSenha($senha1, $senha2); //chamada ou uso da função
validaSenha

?>

</BODY>

</HTML>

```

Devolvendo Valor

Você pode criar uma função que realiza a sua lógica associada e retorna um valor resultante. Muito mais conveniente, na maioria das situações, que imprimir o resultado dentro do próprio corpo da função. Vejamos um programa conhecido que utiliza uma função com esse comportamento.

```

<?php

//listarPares: retorna um novo array com os valores pares do array
$valores

function listarPares ($valores)
{
    $rv = array();

    $tam = count($valores);

    for ($c = 0; $c < $tam; $c++)
    {
        if (($valores[$c] % 2) == 0)
        {
            $rv[] = $valores[$c];
        }
    }
}

$valores = array(1, 2, 3, 4, 5, 6, 7, 8, 9);

$valoresPares = listarPares($valores); //recebendo o resultado

```

```
print_r($valoresPares);
```

```
?>
```

Recursão

Recursão é um termo da ciência da computação para caracterizar implementações de funções que chamam a si para solucionar um problema. Seguindo a maioria das referências, ilustraremos esse conceito com o exemplo do cálculo do fatorial de um número inteiro.

```
<?php
```

```
//fatorial: retorna valor inteiro correspondente ao fatorial de $n
```

```
function fatorial ($n)
```

```
{
```

```
    if ($n == 0 || $n == 1)
```

```
    {
```

```
        return 1;
```

```
    }
```

```
    return $n * fatorial($n-1);
```

```
}
```

```
print "Fatorial de 6 = " . fatorial(6) . "<br>";
```

```
print "Fatorial de 8 = " . fatorial(8);
```

```
?>
```

Criando Biblioteca de Funções

Considere uma aplicação ou mesmo um sistema de informação, que contenham muitas rotinas que são candidatas a tornarem-se funções. Torna-se evidente que precisamos de um local para “organizar” nossas funções. Fazendo uma metáfora das funções de linguagem de programação com as ferramentas de um eletricitista, teríamos a necessidade duma “caixa de ferramentas”. A linguagem PHP utiliza os próprios arquivos texto do sistema como sendo essa “caixa de ferramentas”. Além das funções **include()** e **require()** para utilizarmos nossas ferramentas

(funções). Como resultado, separamos a implementação das funções do código do(s) programa(s) que as utilizam.

Vejamos o resultado do explicado acima em código:

```
<?php //arquivo: minhasFuncoes.php
```

```
function listarPares ($valores)
```

```
{
```

```
...
```

```
}
```

```
function fatorial ($n)
```

```
{
```

```
...
```

```
}
```

```
?>
```

```
<?php //arquivo: programa.php
```

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

```
print "Fatorial de 6 = " . fatorial(6);
```

```
?>
```

Exercícios Propostos

Problema 1

Escreva uma função que recebe, por parâmetro, o nome de um aluno e imprime uma tabela HTML com as informações abaixo:

ESCOLA TÉCNICA DE BRASÍLIA – ETB	
Curso Básico: PHP	
Turno: matutino	Turma: 3124
Aluno: <seu nome aqui>	

Problema 2

Escreva uma função que recebe, por parâmetro, o número de linhas, colunas e um conteúdo (texto) qualquer. Que serão utilizados para criar uma tabela HTML com essas características.

Problema 3

Escreva uma função que transforme horas, minutos e segundos em segundos. Ex.: 2h40min10seg = 9610 segundos.

Problema 4

Escreva uma função que recebe um array de números e retorna o maior valor contido no array.

Problema 5

Escreva uma função que recebe um array de números e retorna um novo array com os elementos em ordem crescente.

Problema 6

Implemente o exemplo da função fatorial, utilizando uma solução iterativa (com loops for ou while) ao invés de recursiva. Compare questões como: facilidade de leitura, desempenho, utilização dos recursos, etc.

Problema 7

Criar uma função para calcular o imposto de renda dos contribuintes, considerando que os dados de cada contribuinte, número de CPF, número de dependentes e renda mensal são valores fornecidos pelo usuário. Para cada contribuinte será feito um desconto de 5% do salário mínimo por dependente.

Os valores para cálculo do imposto são:

Renda líquida	Alíquota
Até 2 salários mínimos	Isento
2 a 3 salários mínimos	5%
3 a 5 salários mínimos	10%

<i>Renda líquida</i>	<i>Alíquota</i>
5 a 7 salários mínimos	15%
Acima de 7 salários mínimos	20%

AULA 5

Manipulação de Arquivos

Uma das tarefas básicas da programação é a criação e manipulação de estruturas de dados. Estas podem ser transientes – estruturas em memória primária como variáveis simples, arrays e objetos, ou persistentes – entidades armazenadas em disco como arquivos e tabelas de banco de dados.

Os arquivos serão utilizados nesta aula de hoje como meio de armazenamento persistente. E como ferramenta para o desenvolvimento do Projeto 1.

OBJETIVOS

- Introdução
- Modos de acesso a arquivos
- Como utilizar arquivos
- Leitura de arquivos
- Escrita em arquivos
- Exercício Resolvido
- Especificação do Projeto prático 1

Introdução

Uma das tarefas básicas da programação é a criação e manipulação de estruturas de dados. Estas podem ser transientes – estruturas em memória primária como variáveis simples, arrays e objetos, ou persistentes – entidades armazenadas em disco como arquivos e tabelas de banco de dados. Arquivos são uma abstração do SO (Sistema Operacional) para organizar e lidar com dados de diferentes dispositivos de armazenamento persistente através duma interface única.

PHP possui recursos poderosos para ler e escrever dados em arquivos do sistema. Além de funções da API para manipulação de diretórios e permissões.

Modos de acesso a Arquivos

Ao iniciar o trabalho com arquivos, utilizamos um parâmetro que determina a operação a ser realizada sobre o arquivo. Havendo um número variado de modos disponíveis. Para o propósito do nosso curso, é importante conhecermos os modos descritos na tabela abaixo:

Modo	O que significa
r	Abre um arquivo no modo LEITURA
w	Abre um arquivo no modo ESCRITA
a	Abre um arquivo no modo CONCATENAÇÃO; o conteúdo existente no arquivo é preservado.

Como utilizar Arquivos

Seja qual for o programa, para utilizar arquivos deve-se seguir os seguintes passos:

Abrir o arquivo

```
$f = fopen("acessos.txt", "r");
```

Escrever no arquivo

```
fwrite($f, "Olá!");
```

Ler do arquivo

```
fgets($f);  
  
//ou  
  
fread($f, filesize($nomeArquivo));  
  
//ou  
  
file($nomeArquivo);
```

Fechar o arquivo

```
fclose($f);
```

Leitura de Arquivos

Nosso primeiro exemplo efetuará a tarefa simples, mas muito utilizada, de abrir um arquivo e ler todo o seu conteúdo. Para esse exemplo, assumo que existe um arquivo cujo nome seja “paragrafo.txt”. Caso não, o crie ou altere o nome para outro arquivo no seu sistema.

```
<?php

// arquivo a ser lido

$nomeArquivo = "paragrafo.txt";


// abre arquivo no modo leitura e retorna manipulador de arquivo

$f = fopen($nomeArquivo, "r") or die("Não foi possível abrir
arquivo");


// lê conteúdo do arquivo associado ao manipulador

$dados = fread($f, filesize($nomeArquivo)) or die("Erro na
leitura");


// fecha arquivo

fclose($f);


// imprime dados

print $dados;

?>
```

Uma forma alternativa de realizar a mesma ação é ler o conteúdo linha-a-linha através da função **fgets()** em combinação com a função de teste de fim de arquivo **feof()**.

```
<?php

$nomeArquivo = "paragrafo.txt";
```

```

$f = fopen($nomeArquivo, "r") or die("Não foi possível abrir
arquivo");

// lê e imprime conteúdo do arquivo
while(!feof($f))
{
    $dados = fgets($f);
    print $dados;
}

fclose($f);

print $dados;

?>

```

E, por fim, temos a possibilidade de carregar o conteúdo inteiro do arquivo em um array. Operação realizada pela função **file()**, que associa cada linha do arquivo a uma posição do array.

● A função `file` efetua a operação de abrir e fechar o arquivo para você!

```

<?php

$nomeArquivo = "paragrafo.txt";

// lê arquivo e retorna conteúdo em array
$dados = file($f) or die("Erro na leitura");

// imprime conteúdo do array
foreach ($dados as $linha)
{
    print $dados;
}

print $dados;

?>

```


Escrita em Arquivo

Para concluir, veremos como realizar a operação complementar de escrita em arquivos. Uma vez executado o programa abaixo, será criado um arquivo chamando “mensagem.txt” contendo o texto “Seja bem vindo!” seguida de uma quebra de linha.

```
<?php

$nomeArquivo = "mensagem.txt";

// abre arquivo para escrita

$f = fopen($nomeArquivo, "w") or die("Erro ao abrir arquivo");

// escreve conteúdo no arquivo com fwrite

fwrite($f, "Seja bem vindo!\n") or die("Erro ao escrever no
arquivo");

fclose($f);

?>
```

Exercício Resolvido

Problema

Manter registro de quantas vezes uma página foi acessada. Ou seja, implementar um contador de acesso simples a uma página.

Solução

Segue uma solução para o problema. Julgue a solução confrontando com a necessidade dessa funcionalidade em sites web. Procure falhas e as corrija, caso existam.

```
<?php

// Descrição: realiza contagem de acessos a esta página

$nomeArquivo = "contadorPagina.txt";

$contagem = 0;

if(file_exists($nomeArquivo)) {

    $f = fopen($nomeArquivo, "r");

    $contagem = fread($f, filesize($nomeArquivo));
```

```
        fclose($f);  
    }  
  
    $contagem++;  
  
    $f = fopen($nomeArquivo, "w");  
    fwrite($f, $contagem);  
    fclose($f);  
?>  
  
<html>  
  
<head>  
  
<title>Contador de página</title>  
  
</head>  
  
<body>  
  
<h2>Quantidade de acessos a esta página: <?= $contagem ?></h2>  
  
</body>  
  
</html>
```

Projeto 1

Criar uma aplicação com funcionalidades de enquete. Uma enquete nada mais é do que uma pesquisa de opinião. Será constituída de uma única pergunta e várias respostas possíveis. Pode-se escolher apenas uma das respostas a cada voto. Qualquer usuário pode votar, não havendo a necessidade de cadastro prévio para tal. Além da interface para votação, disponível ao usuário, deve-se também disponibilizar tela que apresenta o resultado parcial da enquete, em porcentagem.

Telas do programa

Fig.1 - Exemplo de interface de votação **Fig.2 - Exemplo de interface de resultado da votação**

Qual a melhor forma de procurar estágio?

☐ Diretamente nas empresas (local ou site)

☐ Através de eventos e sites de recrutamento e/ou voltados a universitários

☐ Procurar a Central de Estágio da faculdade

☐ Indicação de amigos

☐ Criar um site com seu próprio currículo, para ampliar a divulgação de seus objetivos profissionais

[resultado](#)

Qual a melhor forma de procurar estágio?	
Diretamente nas empresas (local ou site)	24,7%
Através de eventos e sites de recrutamento e/ou voltados a universitários	19,37%
Procurar a Central de Estágio da faculdade	19,96%
Indicação de amigos	30,24%
Criar um site com seu próprio currículo, para ampliar a divulgação de seus objetivos profissionais	5,73%
Voltar	

O que você precisa saber

- Como manipular os valores enviados pelo formulário. Para identificar qual a opção escolhida na tela de votação
- Conhecer os arrays associativos e algumas funções de strings e arrays. Para tornar o código mais simples, compacto e eficiente
- Como ler e escrever informação em arquivo texto. Para manter os votos atuais gravados no Sistema, mesmo que o usuário feche a seção ou que o servidor seja reinicializado, estarão lá
- Complementando o item anterior, decidir como (referente ao leiaute) a quantidade de votos será armazenada no arquivo texto. Para ter condição de ler os votos atuais do arquivo e associar os valores de voto a cada opção disponível na enquete
- Álgebra básica (aritmética racional) e, conseqüentemente, os operadores e expressões da linguagem. Para realizar os cálculos das porcentagens de votos atuais para cada opção da enquete.

AULA 6

Programação Orientada a Objetos

As linguagens de programação modernas geralmente suportam ou oferecem uma aproximação para o desenvolvimento de software Orientado a Objetos (OO). O software construído de forma OO utiliza os conceitos de Classes, Herança e Polimorfismo para compor o programa ou mesmo um sistema de software.

Neste capítulo, trataremos o assunto de maneira prática e sistemática.

OBJETIVOS

- PHP e Programação Orientação a Objetos
- Definindo Classes
- Usando Classes
- Herança
- Polimorfismo

PHP e Programação Orientação a Objetos

- *Utilizaremos a sintaxe da versão 4 devido a necessidade de simplificar ao máximo o assunto, já que o objetivo é fornecer uma introdução sobre POO.*

Desde o PHP 3 temos o suporte integrado na linguagem para aplicarmos os conceitos de Programação Orientada a Objetos (POO). Primitivas para definir Classes, expressar Herança e Polimorfismo. A versão 5 trouxe aperfeiçoamentos nessa área, comparado ao que está presente nas linguagens já estabelecidas, C++ e Java, as quais a PHP pegou emprestado muitas construções.

Definindo Classes

A seção de texto fonte seguinte ilustra como definir uma classe em PHP. Nesse momento não se preocupe com uma utilidade para ela, apenas visualize cada componente e a estrutura básica que todas as classes possuirá.

```
class MinhaClasse

{

    var $atributo1;

    var $atributo2;

    function operacao1()

    {

        //implementação...

    }

}
```

Percebe-se que uma classe pode ser caracterizada como a junção de variáveis associadas logicamente (os atributos) e um conjunto de operações (métodos) que agem sobre estes dados.

Exemplo de Classe simples

Digamos que você precise armazenar as coordenadas de um retângulo e realizar alguns cálculos no seu programa. Primeiramente, suas variáveis serão denominadas x1, y1, x2 e y2. x1 e y1 representam o canto superior esquerdo. x2 e y2 representam o canto inferior direito. Estas quatro variáveis representam um retângulo. Quais as operações (métodos) úteis a implementar nesta classe? Depende da necessidade do programa, mas calcular a altura, o comprimento e a área são bons exemplos. A classe poderia ser implementada como a seguir:

```
class Retangulo

{

    var $x1, $y1, $x2, $y2;
```

```
function Retangulo($v1, $v2, $v3, $v4)
{
    $this->x1 = $v1; $this->y1 = $v2;
    $this->x2 = $v3; $this->y2 = $v4;
}

function altura()
{
    return abs($this->y2 - $this->y1);
}

function largura()
{
    return abs($this->x2 - $this->x1);
}

function area()
{
    return $this->largura() * $this->altura();
}

function perimetro()
{
    return 2 * $this->largura() + 2 * $this->altura();
}
}
```

Criando objetos Retangulo

Após definida a classe Retangulo(ou seja, suas variáveis e métodos) estamos habilitados a criar quantos objetos Retangulo necessitarmos.

```
// Continuação do código acima...
```

```
$retang1 = new Retangulo(2, 2, 25, 10);
```

```
$retang2 = new Retangulo(0, 8, 4, 0);
```

```

print $retang1->x1 . "," . $retang1->y1 . "<br>";

print $retang1->area() . "<br>";

print $retang2->perimetro() . "<br>";

$retang3 = $retang2;

print $retang3->perimetro();

```

Herança

Vamos examinar um exemplo específico para percebermos como a herança funciona. Considere a classe **CarrinhoCompra** já disponível para você. Ela representa todos os dados de compra numa variável(atributo) chamada **\$listaProdutos**, do tipo array associativo. A classe **CarrinhoCompra** já implementa duas operações: adicionar e retirar item do carrinho. O problema é o seguinte: precisamos de uma nova operação para verificar se um item já está presente no carrinho. Entretanto, não podemos alterar a classe original **CarrinhoCompra**.

Utilizaremos a herança para demonstrar como utilizar a implementação existente **CarrinhoCompra** e estender ou especializar ainda mais nossa classe **MeuCarrinhoCompra** com as operações desejadas.

Solução final proposta

```

<?php

// Classe Para carrinho de compras que somente salva, remove e
devolve os itens

// armazenados

class CarrinhoCompra
{
    var $listaProdutos;

    function salvar($produto, $quantidade)
    {
        $this->listaProdutos[$produto] += $quantidade;
    }
}

```



```

function remover($produto, $quantidade)
{
    if($this-> listaProdutos[$produto] > $quantidade)
        $this-> listaProdutos[$produto] -= $quantidade;
    else
        $this-> listaProdutos[$produto] = 0;
}

function getListaProdutos()
{
    return $this->listaProdutos;
}
}

// Classe que melhora a classe anterior, implementando operação de
pesquisa e impressão do carrinho

class MeuCarrinhoCompra extends CarrinhoCompra
{
    function pesquisar($produto)
    {
        $produtos = $this->getListaProdutos();
        return $produtos[$produto];
    }

    function imprimeTabelaProdutos()
    {
        ?>

        <TABLE BORDER="1">

        <TR>

            <TH>Produto</TH>

            <TH>Quantidade</TH>

```

```

</TR>

<?
    foreach($this->getListaProdutos() as $produto => $quantidade)
    {
?>

    <TR>

        <TD><?= $produto ?></TD>

        <TD><?= $quantidade ?></TD>

    </TR>

<?

    } //fim foreach

?>

</TABLE>

<?

    } //fim da função

} //fim da classe

//Fluxo principal do programa

$carrinho = new MeuCarrinhoCompra();

$carrinho->salvar("Camiseta", 7);
$carrinho->salvar("Bermuda", 2);
$carrinho->remover("Camiseta", 1);

print "Quantidade de Camisetas: " . $carrinho-
>pesquisar("Camiseta");

$carrinho->imprimeTabelaProdutos();

?>

```

Polimorfismo

É interessante utilizar o polimorfismo quando identificamos operações semelhantes a uma categoria de classe. Entretanto, estas operações são implementadas de forma particular ou diferente (poli = muitas, morphos = formas) em cada subclasse.

No exemplo abaixo, definimos a classes pai Produto, onde todo objeto da classe Produto possui o método **getDescricao()**. Entretanto, cada subclasse de Produto implementará o método **getDescricao()**, particularizando com as informações do respectivos objeto.

```
<?php

class Produto {

    function getDescricao() {

        exit();

    }

}

class Relogio extends Produto {

    function getDescricao() {

        return "Relógio com ponteiros e marcadores inspirados em
modelos automotores ...";

    }

}

class Tennis extends Produto {

    function getDescricao() {

        return "Tênis esportivo com solado de borracha anti-
derrapante e sistema de absorção de impactos ...";

    }

}

//Fluxo principal do programa

$produtos = array(new Relogio(), new Tennis());

print "<h3>Descrição de Produtos</h3>";

print "<ul>";
```

```
foreach($produtos as $objetoProduto) {  
  
print "<li>" . $objetoProduto->getDescricao() . "</li>";  
  
}  
  
print "</ul>";  
  
?>
```

Exercício Proposto

Problema 1

Criar um programa que possui uma classe chamada **ContaBancaria**. Com os atributos nome, agência, conta e saldo. E as operações **sacar(valor)**, **depositar(valor)** e **imprimirSaldo()**.

AULA 7

Conectividade a Banco de Dados SQL – I

Manter atualizado o conteúdo HTML de um site web é uma tarefa muito custosa. Podemos criar programas web mais fáceis de manter com o uso de um banco de dados. Conseguindo separar a informação do formato utilizado para exibí-la. Seu uso não se restringe apenas a este caso de uso. Bancos de dados são utilizados largamente em soluções para fóruns, comércio eletrônico, gerência do fluxo de trabalho corporativo, entre muitas outras aplicações.

OBJETIVOS

- Introdução
- Suporte PHP para Banco de Dados
- Terminologia de Banco de Dados
- Comandos DDL da SQL
 - CREATE
 - DROP
 - USE
- Comandos DML da SQL
 - INSERT
 - DELETE
 - UPDATE
 - SELECT
 - ORDER BY
 - LIMIT

Introdução

Pelo que foi citado na contextualização acima, torna-se, assim, virtualmente impossível criar uma aplicação WEB, de qualquer tamanho, sem interagir com um banco de dados. Isso pela necessidade freqüente de armazenar a informação para uso futuro. Estaremos explorando os recursos necessários para armazenamento em banco de dados nesta aula.

Para aproveitar ao máximo o conteúdo desta aula, você deve estar familiarizado com o básico da PHP. Iniciaremos com uma visão geral sobre banco de dados SQL e prosseguiremos com estudo de caso ilustrativo de uma “Agenda de contatos WEB”.

Suporte PHP para Banco de Dados

PHP oferece suporte a Banco de Dados Relacionais mais populares disponíveis na atualidade. A grande maioria é compatível com o padrão SQL. PHP oferece um conjunto separado de funções para cada produto de banco de dados ou SGBD (MySQL, PostgreSQL, Oracle, Sybase, MS SQLServer, são alguns exemplos).

O nosso curso utilizará o MySQL como banco de dados para uso nas aplicações. Por motivos importantes quando se está iniciando o estudo nesta área. Entre eles, a facilidade de instalação e uso, documentação de qualidade, interfaces de programação para diversas linguagens, performance otimizada, uso crescente pelas empresas, entre outros.

Terminologia de Banco de Dados

- Um *banco de dados* é uma forma simples e organizada de armazenar itens de dados relacionados
- As informações são organizadas logicamente em *tabelas*
- Cada banco de dados contém uma ou mais tabelas de informação
- Cada tabela possui um ou mais campos, ou colunas, e podem armazenar qualquer número de linhas ou registros
- Sistemas de bancos de dados sofisticados permitem relacionar estes registros. Por exemplo, relacionando os dados de compra realizadas por um cliente. Esses sistemas são conhecidos como Sistemas de Banco de Dados Relacionais
- Cada campo, ou coluna, de uma tabela possui um tipo de dado associado. Semelhante ao tipo de dado de uma variável do PHP. Sendo os maiores grupos os tipos texto, data, numérico
- Além do tipo de dados, campos de tabela podem ser chave primária. Forçando assim que o valor do campo seja único para cada registro
- SQL é a linguagem padrão para consulta e manipulação dos dados de um banco de dados relacional.

Comandos DDL da SQL

Os comandos classificados deste tipo servem para alimentar os metadados do banco de dados. Eles explicitam como serão os dados, seus tipos e tamanhos, além das restrições de chaves. São os comandos executados antes de começar a real alimentação do banco de dados.

Create

O comando CREATE da SQL cria um novo esquema ou uma nova tabela de banco de dados. Seguem dois exemplos de comandos para criarmos um esquema, e uma tabela, respectivamente, no MySQL.

Exemplo:

● *Estes comandos devem ser executados no prompt do MySQL.*

```
CREATE DATABASE meu_banco;

USE meu_banco;

CREATE TABLE contato_tab (

    co_contato INT AUTO_INCREMENT PRIMARY KEY,

    ds_nome VARCHAR(50) NOT NULL,

    ds_email VARCHAR(50) UNIQUE,

    ds_telefone VARCHAR(10),

    ds_endereco VARCHAR(200),

    ds_observacao TEXT,

    dt_aniversario DATE

);
```

Drop

O comando DROP elimina um esquema ou uma a tabela.

Exemplo(s):

```
DROP TABLE contato_tab;
```

Já este elimina o esquema inteiro:

```
DROP DATABASE meu_banco;
```

Use

Troca o esquema atual, ou seja, seta o esquema passado como parâmetro como default. Na prática isso quer dizer que os comandos subsequentes irão ser executados sobre o banco escolhido.

Exemplo:

● *Comando específico do MySQL.*

```
USE meu_banco;
```

Comandos DML da SQL

Os comandos classificados deste tipo servem para manipular dados armazenados no SGBD (Sistema Gerenciador de Banco de Dados). Os comandos a

seguir atuam sobre os dados do banco de dados, ou seja, sobre as linhas(registros) que estão armazenados sob a estrutura das tabelas. Estes são os comandos normalmente encontrados em aplicações que interagem com o SGBD.

Insert

Após criada a tabela, já temos o suficiente para inserir os dados. Esta tarefa é realizada através do comando INSERT da SQL. Cada comando INSERT da SQL insere um registro apenas na tabela. O seguinte comando insere dados de um contato.

Exemplo:

```
INSERT INTO contato_tab (ds_nome, ds_email, ds_telefone,
ds_endereco, ds_observacao, dt_aniversario)

VALUES ('Fulano de Tal', 'fulano@mail.com', '35555555', 'Taguatinga-
DF', 'Sem comentários...', '1980-07-25');
```

Delete

O comando DELETE da SQL exclui um ou mais registros da tabela. Para escolher quais registros devem ou não ser excluídos, utilize a cláusula WHERE.

O seguinte comando exclui todos os contatos cadastrados:

```
DELETE FROM contato_tab;
```

● Caso não seja especificada a cláusula WHERE, todos os registros serão excluídos.

E o seguinte comando exclui o contato “Fulano de Tal”.

```
DELETE FROM contato_tab WHERE ds_nome = 'Fulano de Tal';
```

Update

O comando UPDATE da SQL pode ser utilizado para modificar o conteúdo de registro(s) da tabela. Assim como o comando DELETE, utiliza-se a cláusula WHERE para especificar quais registros atualizar. Por exemplo, o seguinte comando atualiza o email do contato “Fulano de Tal”.

```
UPDATE contato_tab SET ds_email = 'fulano.tal@mail.com' WHERE
ds_nome = 'Fulano de Tal'
```

● Caso não seja especificada a cláusula WHERE, todos os registros serão atualizados.

Select

O comando SELECT da SQL é o mais utilizado. Com este comando podemos ler qualquer informação cadastrada nas tabelas do banco de dados. Tanto de uma ou mais tabelas num mesmo comando.

O comando mais simples, que retorna todos os registros da tabela contato é ilustrado à seguir.

```
SELECT * FROM contato_tab;
```

Para buscar campos específicos da tabela contato_tab, basta especificar a listagem de campos separados por vírgulas.

```
SELECT ds_nome, ds_email, dt_aniversario FROM contato_tab;
```

Para selecionar registros específicos utilizamos a cláusula WHERE após o nome da(s) tabela(s).

```
SELECT ds_email, ds_endereco, dt_aniversario  
  
FROM contato_tab  
  
WHERE ds_nome = 'Fulano de Tal'
```

Por fim, pode-se também pesquisar por um padrão de texto. Por exemplo, caso não saibamos o nome completo é possível encontrar com o seguinte comando:

```
SELECT * FROM contato_tab WHERE ds_nome LIKE '%Fulano%';
```

Order By

A cláusula **ORDER BY** lhe permite a ordenação da pesquisa realizada pelo comando **SELECT**.

Por exemplo, o comando à seguir pesquisa todos os contatos ordenados por nome:

```
SELECT * FROM contato_tab ORDER BY ds_nome;
```

Limit

Outra cláusula muito útil para consultas é a **LIMIT**. Que permite retornar somente um número especificado de registros da tabela. Muito útil para efeito de paginação do resultado, quando o mesmo não cabe na janela de exibição.

Por exemplo, esta consulta retorna somente os dez primeiros contatos cadastrados:

```
SELECT * FROM contato_tab LIMIT 1, 10;
```

AULA 8

Conectividade a Banco de Dados SQL - II

Continuação...

OBJETIVOS

- API PHP de Conectividade ao MySQL
- Programas de Cadastro
- Programas de Consulta
- Especificação do Projeto prático 2

API PHP para conectividade ao MySQL

Após familiarizado com os conceitos básicos de banco de dados, temos que conhecer a biblioteca disponível na linguagem, para acesso ao MySQL. SGBD que será utilizado para o curso. Veremos, basicamente, que a biblioteca é constituída de funções simples para abrir conexão, selecionar um banco de dados, executar comando SQL, obter resultados, entre outras menos utilizadas.

A seguir, temos uma tabela explicativa com as funções mais úteis para o nosso aprendizado:

● É fortemente aconselhável um estudo no manual da linguagem PHP para familiarizar-se com as funções presentes nesta biblioteca.

Função	Descrição
<code>resource mysql_connect (string maquinaBD, string usuarioBD, string senhaBD)</code>	Retorna recurso que representa a conexão ao MySQL
<code>bool mysql_close (resource conexaoBD)</code>	Fecha ou libera conexão ao MySQL
<code>bool mysql_select_db (string nomeBD [, resource conexaoBD])</code>	Conecta ao um banco específico no servidor MySQL representado pela conexão
<code>resource mysql_query (string comandoSQL [, resource conexaoBD])</code>	Retorna resultado da execução de consulta ao servidor MySQL associado a conexão
<code>array mysql_fetch_array (resource resultadoConsulta [, resource conexaoBD])</code>	Retorna um registro do resultado em formato de array
<code>int mysql_num_rows (resource resultado)</code>	Retorna quantidade de registros no resultado
<code>int mysql_errno ([resource conexaoBD])</code>	Retorna código interno de erro de operação anterior
<code>string mysql_error ([resource conexaoBD])</code>	Retorna texto de erro de operação anterior
<code>int mysql_affected_rows ([resource conexaoBD])</code>	Retorna o número de linhas afetadas em operação anterior realizada no MySQL

Conectando ao Servidor MySQL

O primeiro passo é estabelecer uma conexão entre o PHP e o servidor de banco de dados MySQL. Isso é realizado pela chamada à função `mysql_connect()`. É obrigatório o fornecimento dos parâmetros de conexão caso não se utilize a conexão padrão.

● Utilize o usuário `root` em suas aplicações somente para fins didáticos!

```
$conexaoBD = mysql_connect("localhost", "root", "") or  
die(mysql_error());
```

Se o PHP não conseguir estabelecer a conexão, o processamento pára(*die*). Sendo exibida uma mensagem que explica o que ocorreu.

Assumindo que tudo ocorreu bem, a conexão se manterá disponível até que a feche, ou até que a página termine o processamento.

Conectando ao Banco de Dados

Como um servidor MySQL comporta vários bancos de dados, você precisa, além de abrir conexão ao servidor, selecionar o banco de dados que deseja trabalhar. Tarefa esta realizada pela função **mysql_select_db()**.

```
mysql_select_db("meu_banco") or die(mysql_error());
```

Agora, estamos prontos para realizar quaisquer operações com o banco de dados selecionado. Veremos que, na grande maioria, são apenas cinco: *Criação, Inclusão, Exclusão, Atualização e Consulta*. Importante também perceber que estas operações são realizadas através de comandos da linguagem SQL. Não só para o MySQL como também para qualquer produto compatível com a SQL.

Criando tabela

Para o nosso exemplo, utilizaremos a tabela de contatos. Segue, novamente, o comando SQL que será utilizado para criar tal tabela.

```
CREATE TABLE contato_tab (
    co_contato INT AUTO_INCREMENT PRIMARY KEY,
    ds_nome VARCHAR(50) NOT NULL UNIQUE,
    ds_email VARCHAR(50),
    ds_telefone VARCHAR(10),
    ds_endereco VARCHAR(200),
    ds_observacao TEXT,
    dt_aniversario DATE
);
```

● Para executar este comando, pode-se fazê-lo pela própria PHP ou mesmo por um cliente MySQL. Como o MySQL Query Browser, MySQL-Front, PHPMyAdmin, entre muitos outros.

Inserindo registros

Agora é a hora de inserir dados na tabela acima criada. Para isto utilizaremos o comando INSERT da SQL passando os dados recebidos de um formulário de cadastro. Para executar o comando SQL pela PHP utilizaremos a função **mysql_query()**. A seção de código PHP terá a seguinte aparência:

```
$sql =
"INSERT INTO contato_tab (ds_nome, ds_email, ds_telefone,
ds_endereco, ds_observacao,dt_aniversario) VALUES ('$nome',
'$email', '$telefone', '$endereco', '$obs', '$dataAniv') ";

$resultado = mysql_query($sql);
```

Fechando conexão

Necessitamos ter ciência da importância de fechar a conexão ao banco de dados. Para não ter problemas com excesso de conexões ou até mesmo a conseqüente

“queda” do servidor de banco de dados. Temos, novamente, uma função para realizar este trabalho: **mysql_close()**.

```
mysql_close($conexaoBD);
```

Basta chamar a função **mysql_close()** passando a conexão a ser fechada como parâmetro.

Juntando tudo em página de Cadastro

Observação: Para complementar o código a seguir, você deve criar, antes, tela com os seguintes campos de entrada submetendo para página de processamento do cadastro.

● Esta implementação não inclui o código fonte para a criação do formulário HTML. Que deve ser feito em página à parte pelo aluno.

A seguir, temos o código fonte completo do programa para realizar o cadastro de um contato.

```
<?php
// Descrição: salva dados de um contato no banco de dados

$nome  = $_POST["txtNome"];
$email = $_POST["txtEmail"];
$obs   = $_POST["txtObs"];
$telefone = $_POST["txtTelefone"];
$endereco = $_POST["txtEndereco"];
$email    = $_POST["txtEmail"];
$dataAniv = explode("/", $_POST["txtDataAniv"]);
```

```

$dataAniv = array_reverse($dataAniv);

$dataAniv = implode("-", $dataAniv);

//abre conexao ao servidor MySQL

$conexaoBD = mysql_connect("localhost", "root", "") or
die(mysql_error());

//conecta-se ao banco de dados meu_banco

mysql_select_db("meu_banco", $conexaoBD) or die("Erro:" .
mysql_error());

//comando SQL de cadastro

$sql =

"INSERT INTO contato_tab (ds_nome, ds_email, ds_telefone,
ds_endereco, ds_observacao,dt_aniversario) VALUES ('$nome',
'$email', '$telefone', '$endereco', '$obs', '$dataAniv') ";

mysql_query($sql) or die("Erro no comando SQL: " . mysql_error());

if(mysql_affected_rows($conexaoBD) == 1) {

    print "Contato cadastrado com sucesso!";

}

else {

    print "Cadastro não realizado. Motivo: " . mysql_error();

}

//fecha conexao ao servidor MySQL

mysql_close($conexaoBD);

?>

```

Juntando tudo em página de Consulta

É importante, também, desenvolver uma página completa para ilustrar o comando **SELECT**. Basicamente, criaremos uma página que consulta informações dos

contatos cadastrados no banco de dados. Uma explicação mais detalhada dos recursos novos está disponível após o exemplo.

```
<HTML>

<BODY>

<?php

// Descrição: apresenta listagem dos contatos cadastrados no banco
de dados


//abre conexao ao servidor MySQL

$conexaoBD = mysql_connect("localhost", "root", "") or

die("Erro:" . mysql_error());


//conecta-se ao banco de dados meu_banco

mysql_select_db("meu_banco", $conexaoBD) or die("Erro:" .
mysql_error());


//cria consulta SQL

$sql = "SELECT ds_nome, ds_email, ds_endereco, ds_telefone,
dt_aniversario FROM contato_tab ORDER BY ds_nome ";


//executa consulta

$resultado = mysql_query($sql, $conexaoBD) or die("Erro:" .
mysql_error());

?>

<TABLE BORDER="1">

  <TR>

    <TD><b>Nome</b></TD>

    <TD><b>Email</b></TD>

    <TD><b>Endereço</b></TD>

    <TD><b>Telefone</b></TD>

    <TD><b>Aniversário</b></TD>

  </TR>
```

```

<?
while($registro = mysql_fetch_array($resultado)) //lê registro
{
?>

    <TR>

        <TD><?= $registro["ds_nome"] ?></TD>

        <TD><?= $registro["ds_email"] ?></TD>

        <TD><?= $registro["ds_endereco"] ?></TD>

        <TD><?= $registro["ds_telefone"] ?></TD>

        <TD><?= $registro["dt_aniversario"] ?></TD>

    </TR>

<?
}
?>

</TABLE>

</BODY>

</HTML>

<?

//fecha conexao ao servidor MySQL

mysql_close($conexaoBD);

?>

```

Bem, o que temos de novo é a chamada à função `mysql_fetch_array()`. Necessária para ler o resultado da execução do comando `SELECT`. Seu uso é simples, passando o resultado da consulta ela nos retorna um registro da tabela por vez, em forma de array associativo. Então, utilizamos os valores presentes no array para processamento da informação. Perceba que é feita uma cópia do registro da tabela para uma variável array.

Projeto 2

Criar um módulo de autenticação para aplicações web. O motivo se deve a esta funcionalidade ser muito útil em sítios ou mesmo sistemas que precisam de um certo nível de segurança no acesso à informação.

O mecanismo de autenticação deve funcionar da seguinte forma. É apresentada tela ao usuário com dois campos de edição: login e senha. Além de um botão, que quando clicado dispara o procedimento de autenticação. O procedimento de autenticação realiza pesquisa no banco de dados verificando se existe usuário cadastrado com o login e senha fornecidos. Caso exista, redirecionar o fluxo para tela inicial de boas vindas. Do contrário, retornar o fluxo para a tela de autenticação, exibindo mensagem de falha na autenticação.

Fig.1 Tela inicial de autenticação

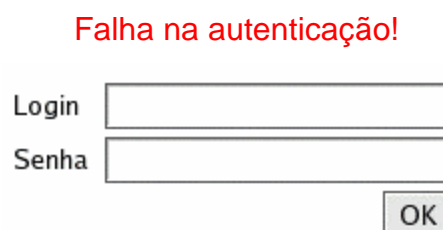


Formulário de autenticação inicial com campos para Login e Senha, e um botão OK.

Login:

Senha:

Fig.2 Tela inicial de autenticação com falha



Tela de autenticação com falha, exibindo a mensagem "Falha na autenticação!" em vermelho. O formulário contém campos para Login e Senha, e um botão OK.

Falha na autenticação!

Login:

Senha:

O que você precisa saber

- Como manipular os valores enviados pelo formulário.
- Conceitos básicos de banco de dados
- Comandos básicos da linguagem SQL
- API padrão do PHP para conectividade ao MySQL

AULA 9

Conectividade a Banco de Dados SQL – III

Continuação...

OBJETIVO

- Integração dos conceitos
- Estudo de Caso

Estudo de Caso

Implementação de páginas de cadastro e pesquisa. A seguir, tem-se a definição da tabela de banco de dados e as páginas que implementam as operações de cadastro e a pesquisa.

Passo1: Criar tabela de livros no banco de dados

```
CREATE TABLE livro_tab (  
    isbn CHAR(10) NOT NULL PRIMARY KEY,  
    titulo VARCHAR(100),  
    autor VARCHAR(50),  
    editora VARCHAR(50)  
);
```

Passo 2: Criar página de cadastro de livros

```
<?php  
  
if($_POST) {  
    $isbn = $_POST["txtISBN"];  
    $titulo = $_POST["txtTitulo"];  
    $autor = $_POST["txtAutor"];  
    $editora = $_POST["txtEditora"];  
  
    $conexao = mysql_connect("localhost", "root", "") or  
die(mysql_error());  
  
    mysql_select_db("banco_aluno") or die(mysql_error());  
  
    if ($isbn != NULL && $titulo != NULL) {  
  
        $sql = "INSERT INTO livro_tab (isbn, titulo, autor, editora)  
            VALUES ('$isbn', '$titulo', '$autor', '$editora') ";
```

```
mysql_query($sql) or die(mysql_error());

mysql_close($conexao);

$mensagem = "Livro cadastrado com sucesso!";
}
}
?>

<HTML>

<HEAD><TITLE>Tela Cadastro</TITLE></HEAD>

<BODY>

<FORM ACTION="" METHOD="post">

<CENTER>Cadastro de Livros</CENTER>

<TABLE ALIGN="center">

<TR>

<TD>ISBN:</TD>

<TD><INPUT TYPE="text" NAME="txtISBN"/></TD>

</TR>

<TR>

<TD>Título:</TD>

<TD><INPUT TYPE="text" NAME="txtTitulo"></TD>

</TR>

<TR>

<TD>Autor:</TD>

<TD><INPUT TYPE="text" NAME="txtAutor"></TD>

</TR>

<TR>

<TD>Editora:</TD>

<TD><INPUT TYPE="text" NAME="txtEditora"></TD>
```

```

</TR>

<TR>

    <TD COLSPAN="2" ALIGN="right">

        <INPUT TYPE="submit" NAME="btnCadastrar" VALUE="Cadastrar"/>&nbsp;

        <INPUT TYPE="reset" VALUE="Limpar"/>

    </TD>

</TR>

<BR/><CENTER><?= $mensagem ?></CENTER>

</TABLE>

</BODY>

</HTML>

```

Passo 3: Criar página de consulta de livros

```

<?php

$isbn = $_POST["txtISBN"];
$titulo = $_POST["txtTitulo"];
$autor = $_POST["txtAutor"];
$editora = $_POST["txtEditora"];

$conexao = mysql_connect("localhost", "root", "") or
die(mysql_error());

mysql_select_db("meu_banco") or die(mysql_error());

if ($_POST) {

    $sql = "SELECT isbn, titulo, autor, editora FROM livro_tab WHERE
1 = 1 ";

    if ($isbn != NULL)

        $sql = $sql . " AND isbn = '$isbn' ";

```



```

        if ($titulo != NULL)

            $sql = $sql . " AND titulo LIKE '%$titulo%' ";

        if ($autor != NULL)

            $sql = $sql . " AND autor LIKE '%$autor%' ";

        if ($editora != NULL)

            $sql = $sql . " AND editora LIKE '%$editora%' ";

        $sql = $sql . " ORDER BY titulo ";

        $resultado = mysql_query($sql) or die(mysql_error());
    }
?>

<HTML>

<HEAD><TITLE>Tela Cadastro</TITLE></HEAD>

<BODY>

<FORM ACTION="" METHOD="post">

<CENTER>Consulta Livros</CENTER>

<TABLE ALIGN="center">

<TR>

    <TD>ISBN:</TD>

    <TD><INPUT TYPE="text" NAME="txtISBN"/></TD>

</TR>

<TR>

    <TD>Título:</TD>

    <TD><INPUT TYPE="text" NAME="txtTitulo"></TD>

</TR>

<TR>

    <TD>Autor:</TD>

    <TD><INPUT TYPE="text" NAME="txtAutor"></TD>

</TR>

```

```

<TR>

    <TD>Editora:</TD>

    <TD><INPUT TYPE="text" NAME="txtEditora"></TD>

</TR>

<TR>

    <TD COLSPAN="2" ALIGN="right">

        <INPUT TYPE="submit" NAME="btnConsultar"
        VALUE="Consultar"/>&nbsp;

    </TD>

</TR>

</TABLE>

<?php
if ($resultado != NULL) {
?>

<TABLE ALIGN="center">

<TR>

<TD><B>ISBN</B></TD>

<TD><B>Título</B></TD>

<TD><B>Autor</B></TD>

<TD><B>Editora</B></TD>

<TD><B>&nbsp;</B></TD>

<TD><B>&nbsp;</B></TD>

</TR>

<? while ($registro = mysql_fetch_array($resultado)) { ?>

<TR>

    <TD><?= $registro["isbn"] ?></TD>

    <TD><?= $registro["titulo"] ?></TD>

    <TD><?= $registro["autor"] ?></TD>

    <TD><?= $registro["editora"] ?></TD>

    <TD>

```

```
<a
href="excluirLivro.php?isbn=<?=$registro["isbn"]?>&acao=excluir">

    Excluir

</a>

</TD>

<TD>

    <a
href="atualizarLivro.php?isbn=<?=$registro["isbn"]?>&acao=atualizar"
">

        Atualizar

    </a>

</TD>

</TR>

<? } //while

} //if

?>

</TABLE>

</BODY>

</HTML>

<?php
mysql_close($conexao);

?>
```

Exercício Proposto

Implemente os programas para realizar as operações de Exclusão e Atualização dos livros cadastrados.