PHP com MySQL

Enviada por:
José Henrique Monteiro De Almeida

Sumário

1. Introdução ao PHP

1.1. O que é PHP?

O PHP é uma linguagem que permite criar sites WEB dinâmicos, possibilitando uma interação com o usuário através de formulários, parâmetros da URL e links. A diferença de PHP com relação a linguagens semelhantes a Javascript é que o código PHP é executado no servidor, sendo enviado para o cliente apenas html puro.

Desta maneira é possível interagir com bancos de dados e aplicações existentes no servidor, com a vantagem de não expor o código fonte para o cliente. Isso pode ser útil quando o programa está lidando com senhas ou qualquer tipo de informação confidencial.

O que diferencia PHP de um script CGI escrito em C ou Perl é que o código PHP fica embutido no próprio HTML, enquanto no outro caso é necessário que o script CGI gere todo o código HTML, ou leia de um outro arquivo.

1.2. Como surgiu a linguagem PHP?

A linguagem PHP foi concebida durante o outono de 1994 por Rasmus Lerdorf. As primeiras versões não foram disponibilizadas, tendo sido utilizadas em sua home-page apenas para que ele pudesse ter informações sobre as visitas que estavam sendo feitas.

A primeira versão utilizada por outras pessoas foi disponibilizada em 1995, e ficou conhecida como “Personal Home Page Tools” (ferramentas para página pessoal). Era composta por um sistema bastante simples que interpretava algumas macros e alguns utilitários que rodavam “por trás” das home-pages: um livro de visitas, um contador e algumas outras coisas.

Em meados de 1995 o interpretador foi reescrito, e ganhou o nome de PHP/FI, o “FI” veio de um outro pacote escrito por Rasmus que interpretava dados de formulários HTML (Form Interpreter).

Ele combinou os scripts do pacote Personal Home Page Tools com o FI e adicionou suporte a mSQL, nascendo assim o PHP/FI, que cresceu bastante, e as pessoas passaram a contribuir com o projeto.

Estima-se que em 1996, PHP/FI estava sendo usado por cerca de 15.000 sites pelo mundo, e em meados de 1997 esse número subiu para mais de 50.000. Nessa época houve uma mudança no desenvolvimento do PHP.

Ele deixou de ser um projeto de Rasmus com contribuições de outras pessoas para ter uma equipe de desenvolvimento mais organizada. O interpretador foi reescrito por Zeev Suraski e Andi Gutmans. Esse novo interpretador foi à base para a versão 3.

O lançamento do PHP4, ocorrido em 22/05/2000, trouxe muitas novidades aos programadores de PHP. Uma das principais foi o suporte a sessões, bastante útil pra identificar o cliente que solicitou determinada informação.

Além das mudanças referentes a sintaxe e novos recursos de programação, o PHP4 trouxe como novidade um otimizador chamado Zend, que permite a execução muito mais rápida de scripts PHP.A empresa que produz o Zend promete para este ano o lançamento de um compilador de PHP. Códigos compilados serão executados mais rapidamente, além de proteger o fonte da aplicação. Hoje, já se encontra disponível no mercado a versão 5.x do PHP.

1.3. Características da Linguagem PHP

- É uma linguagem de fácil aprendizado;
- Tem suporte a um grande número de bancos de dados como: dBase, Interbase, mSQL, mySQL, Oracle, Sybase, PostgreSQL e vários outros.
- Tem suporte a outros serviços através de protocolos como IMAP, SNMP, NNTP, POP3 e, logicamente, HTTP;
- É multiplataforma, tendo suporte aos sistemas Operacionais mais utilizados no mercado;
- Seu código é livre, não é preciso pagar por sua utilização e pode ser alterado pelo usuário na medida da necessidade de cada usuário
- Não precisa ser compilado.

1.4. Configuração do IIS

Antes de começarmos a trabalhar com o PHP, precisamos instalar e configurar o PHP, o MySQL e o IIS (Internet Information Service).

Para configurá-los siga as orientações do tutorial **Instalando e Configurando o PHP, IIS e MySQL** que acompanha o CD do curso de PHP.

1.5. Testando o PHP

Vamos criar um pequeno script em PHP para testarmos e verificarmos se o PHP e o IIS foram instalados e configurados corretamente no seu computador.

Utilizando o Dreamweaver ou o Bloco de Notas, digite o script a seguir.

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

<?
    echo "Primeiro Exemplo";
?>

</body>
</html>
```

Salve o arquivo como “**primeiro.php**” na pasta **C:\Cursos\alunos\seudiretório**.

Agora, abra o navegador e, na barra de endereços, digite o seguinte endereço:

http://localhost/seunome/primeiro.php, onde seu nome é o diretório virtual que você criou, lembra?

Observe que o seu browser exibiu a frase **Primeiro exemplo**. Sinal de que o PHP e o IIS foram instalados e configurados corretamente no seu computador.

2. Sintaxe do PHP

2.1. Delimitando o código PHP

O código PHP fica embutido no próprio HTML. O interpretador identifica quando um código é PHP pelas seguintes tags:

```
<?php
    comandos;
?>

<script language=”php”>
    comandos;
</script>

<?
    Comandos;
?>

<%
    comandos;
%>
```

O tipo de tags mais utilizado é o terceiro, que consiste em uma “abreviação” do primeiro. Para utilizá-lo, é necessário habilitar a opção short-tags na configuração do PHP. O último tipo serve para facilitar o uso por programadores acostumados à sintaxe de ASP. Para utilizá-lo também é necessário habilitá-lo no PHP, através do arquivo de configuração ***php.ini***.

2.2. Separador de instruções

Entre cada instrução em PHP é preciso utilizar o **ponto-e-vírgula**, assim como em C, Perl e outras linguagens mais conhecidas. Na última instrução do bloco de script não é necessário o uso do ponto-e-vírgula, mas por questões estéticas recomenda-se o uso sempre.

2.3. Comentários

Há dois tipos de comentários em código PHP:

Comentários de uma linha:

Marca como comentário até o final da linha ou até o final do bloco de código PHP – o que vier antes. Pode ser delimitado pelo caracter “#” ou por duas barras (//). O delimitador “//”, normalmente, é o mais utilizado.

Exemplo:

```
<?

    echo “teste”; // este teste é similar ao anterior
    echo “teste”; /# este teste é similar ao anterior
    // sql “teste”;

?>
```

Comentários de mais de uma linha:

Tem como delimitadores os caracteres "/*" para o início do bloco e "*/" para o final do comentário. Se o delimitador de final de código PHP (?>) estiver dentro de um comentário, não será reconhecido pelo interpretador.

Exemplo:

```
<?
    echo "teste";

    /*
        este é um comentário com mais de uma linha.

        echo "teste. Este comando echo é ignorado pelo interpretador do PHP por ser um comentário."
    */
?>
```

2.4. Imprimindo código html

Um script php geralmente tem como resultado uma página html, ou algum outro texto. Para gerar esse resultado, deve ser utilizada uma das funções de impressão, echo e print.

Sintaxes:

```
print(argumento);

echo (argumento1, argumento2, ... );

echo argumento;
```

Exemplos:

```
<?

    $a = 10;
    $b = 20;
    $soma = $a + $b;

    //Imprimindo a soma na tela com o comando print.

    print ("A soma é: ". $soma."<p>");

    ////Imprimindo a soma na tela com o comando echo.

    echo ("A soma é: ". $soma."<p>");

    echo "<br>";

    echo $a." ".$b;

?>
```

3 – Variáveis

3.1. Nomes de variáveis

Toda variável em PHP tem seu nome composto pelo caracter **$(dólar)** e uma string, que deve iniciar por uma letra ou o caracter "_". O PHP é **case sensitive**, ou seja, as variáveis $vivas e $VIVAS 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 algumas variáveis pré-definidas cujos nomes são formados por letras maiúsculas.

Exemplo de variáveis:

- $teste = nome correto de variável
- teste= nome errado de variável.
- _teste= nome correto de variável

Obs.: Normalmente, a variável é criada utilizando-se, na maioria das vezes, o caracter **$ (dólar)**.

3.2. Tipos Suportados

O PHP suporta os seguintes tipos de dados:

- Inteiro
- Ponto flutuante
- String
- Array
- Objeto

O PHP utiliza checagem de tipos dinâmica, ou seja, uma variável pode conter valores de diferentes tipos em diferentes momentos da execução do script. Por este motivo não é necessário declarar o tipo de uma variável para usá-la. O interpretador PHP decidirá qual o tipo daquela variável, verificando o conteúdo em tempo de execução. Ainda assim, é permitido converter os valores de um tipo para outro desejado, utilizando o ***typecasting*** ou a função ***settype*** (ver adiante).

Inteiros (integer ou long)

Uma variável pode conter um valor inteiro com atribuições que sigam as seguintes sintaxes:

- $vivas = 1234; # inteiro positivo na base decimal
- $vivas = -234; # inteiro negativo na base decimal
- $vivas = 0234; # inteiro na base octal-simbolizado pelo 0 equivale a 156 decimal
- $vivas = 0x34; # inteiro na base hexadecimal(simbolizado pelo 0x) – equivale a 52 decimal.

A diferença entre **inteiros simples** e **long** está no número de bytes utilizados para armazenar a variável. Como a escolha é feita pelo interpretador PHP de maneira transparente para o usuário, podemos afirmar que os tipos são iguais.

Ponto Flutuante (double ou float)

Uma variável pode ter um valor em ponto flutuante com atribuições que sigam as seguintes sintaxes:

- **$vivas = 1.234;**
- **$vivas = 23e4; # equivale a 230.000**

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 exemplo 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';
    $texto = “Curso de PHP”;
    echo "$vivas";

?>
```

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

```
<?

    $texto = “Curso de PHP”;
    $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

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[0] = “amarelo”;
    $cor[1] = “vermelho”;
    $cor[2] = “verde”;
    $cor[3] = “azul”;
    $cor[4] = “anil”;
    $cor[“teste”] = 1;
?>
```

Equivalentemente, pode-se escrever:

```
<?
    $cor = array(0 => “amarelo”, 1 => “vermelho”, 2 => “verde”, 3 => “azul”, 4 => “anil”, 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 serinteiros, 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;

    echo “$a<br>”;
    echo “$b<br>”;
    echo “$c<br>”;
    echo “$d<br>”;
?>
```

Após a execução do código acima temos os seguintes valores:

$a == null (vazio, não aparece nada)
$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.

Booleanos

O 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.

3.3. Transformação de tipos

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

a) 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:

```
<?

    //declaração da variável vivas
    //-------------------------------------
    $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

    echo “O valor de vivas é: $vivas”;

?>
```

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
    echo “O valor de vivas é: $vivas”;

?>
```

b) Transformação explícita de tipos

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

    echo “O valor de vivas é: $vivas”;

?>
```

Os tipos permitidos na ***Transformação explícita*** 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”.

Sintaxe:

Settype(nomedavariável,novo tipo da variávbel)

Exemplo:

```
<?

    $vivas = 15;                // $vivas é integer
    settype($vivas,double)  // $vivas é double
    $vivas = 15;                // $vivas é integer
    settype($vivas,string)   // $vivas é string

    echo “O valor de vivas é: $vivas”;

?>
```

4. Operadores

4.1. Operadores Aritméticos

Só podem ser utilizados quando os operandos são números (integer ou float). Se forem de outro tipo, terão seus valores convertidos antes da realização da operação. São eles:

+	adição
-	subtração
*****	multiplicação
/	divisão
%	módulo

4.2. Operadores de Strings

Só há um operador exclusivo para strings:

. concatenação

Exemplo:

```
<?

    $nome = “Gláucio”;
    $sobrenome = “Nascimento”;
    echo $nome.” “.$sobrenome;

?>
```

A saída do script acima será: Gláucio Nascimento

4.3. Operadores de atribuição

Existe um operador básico de atribuição e diversos derivados. Sempre retornam o valor atribuído. No caso dos operadores derivados de atribuição, a operação é feita entre os dois operandos, sendo atribuído o resultado para o primeiro. A atribuição é sempre por valor, e não por referência.

=	atribuição simples
+=	atribuição com adição
-=	atribuição com subtração
*=	atribuição com multiplicação
/=	atribuição com divisão
%=	atribuição com módulo
.=	atribuição com concatenação

Exemplo:

```
<?

    $a = 7;
    $a += 2; // $a passa a conter o valor 9

?>
```

4.4. Operadores bit a bit

Comparam dois números bit a bit.

&	“e” lógico
|	“ou” lógico
^	ou exclusivo
~	não (inversão)
<<	shift left
>>	shift right

4.5. Operadores Lógicos

Utilizados para inteiros representando valores booleanos

and	“e” lógico
or	“ou” lógico
xor	ou exclusivo
!	não (inversão)
&&	“e” lógico
||	“ou” lógico

Existem dois operadores para “e” e para “ou” porque eles têm diferentes posições na ordem de precedência.

4.6. Operadores de Comparação

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

4.7. Operadores de Expressão condicional

Existe um operador de seleção que é **ternário**. Funciona assim:

(expressao1)?(expressao2):(expressao3)

O interpretador PHP avalia a primeira expressão. Se ela for verdadeira, a expressão retorna o valor de expressão2. Senão, retorna o valor de expressão3.

4.8. Operadores de incremento e decremento

- ++ incremento
- decremento

Podem ser utilizados de duas formas: antes ou depois da variável. Quando utilizado antes, retorna o valor da variável antes de incrementá-la ou decrementá-la.

Quando utilizado depois, retorna o valor da variável já incrementado ou decrementado.

Exemplos:

```
<?

    $a = $b = 10;    // $a e $b recebem o valor 10
    $c = $a++;       // $c recebe 10 e $a passa a ter 11
    $d = ++$b;       // $d recebe 11, valor de $b já incrementado

?>
```

5. Estruturas de Controle

As estruturas que veremos a seguir são comuns para as linguagens de programação imperativas, bastando, portanto, descrever a sintaxe de cada uma delas, resumindo o funcionamento.

5.1. Blocos

Um bloco consiste de vários comandos agrupados com o objetivo de relacioná-los com determinado comando ou função. Em comandos como **if, for, while, switch** e em declarações de funções blocos podem ser utilizados para permitir que um comando faça parte do contexto desejado. Blocos em PHP são delimitados pelos caracteres **"{"** e **"}"**. A utilização dos delimitadores de bloco em uma parte qualquer do código não relacionada com os comandos citados ou funções não produzirá efeito algum, e será tratada normalmente pelo interpretador.

Exemplo:

```
<?
    if ($x == $y)
    comando1;
    comando2;
?>
```

Para que comando2 esteja relacionado ao if é preciso utilizar um bloco:

```
<?
    if ($x == $y) {
    comando1;
    comando2;
    }
?>
```

5.2. Estrutura if / else / elseif

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:

Sintaxes:

```
if (expressão)
        comando;

if (expressão) {
        comando;
        . . .
        . . .
        . . .
        comando;
}
```

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

Exemplo 1:

```
<?

$a = 10;
$b = 20;

if ($a > $b)

    echo " o Valor de a é: ".$a + $b;

?>
```

Exemplo 2:

```
<?

$a = 10;
$b = 20;

if ($a >= $b) {

    $media = ($a + $b) / 2;
    $resto = $a % $b;//calcula o resto da divisão de A por B
    echo " o Valor de A é: ".$a "<br>";
    echo " o Valor de B é: ".$b <br>;
    echo " o Valor da média é: ".$media <br>;
    echo " O resto da divisão de A por B é: ".$resto;
}

?>
```

Else

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:

Sintaxes:

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

ou

```
if (expressão) {
        comando 1;
        comando n
else {
        comando 1;
        comando n
}
```

Exemplos:

```
<?
```

```
        $a = 10;
        $b = 20;

        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 seqüencialmente, 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.

Elseif

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

Sintaxe:

```
if (expressao1)
        comando;
elseif (expressao2)
        comando;
else
        comando;
```

ou

```
if (expressão 1) {
        comando 1;
        comando n;
}
elseif (expressão 2) {
        comando 1;
        comando n;
}
else {
        comando 1;
        comando n;
}
```

Exemplo:

<?

```
$nota1 = 6;
$nota 2 = 8;
$media = ($nota1 + $nota2) / 2;

if ($media > 7) {
    echo “Média: ”.$media .”<br>”;
    echo “Aluno aprovado.”;
}
elseif ($media <7) {
    echo “Média: ”.$media .”<br>”;
    echo “Aluno reprovado.”;
}
else {
    echo “Média: ”.$media.”<br>”;
    echo “Aluno em recuperação.”;
}

?>
```

5.3. Estrutura Switch Case

O comando **switch** atua de maneira semelhante a uma série de comandos **if** na mesma expressão. Freqüentemente 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**.

Sintaxe:

```
swith ($valor){
    case "____";
        comandos;
        comandos;
        break;
    case "____";
        comandos;
        comandos;
        break;
    case "____";
        comandos;
        comandos;
        break;
    case "____";
        comandos;
        comandos;
        break;
    default;
        comandos;
        comandos;
        break;
```

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**:

Exemplos 1: Estrutura IF

```
<?
```

```
    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";

?>
```

Exemplo 2: Estrutura SWITCH

```
<?
    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.

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:

Exemplo:

```
<?
    switch ($s) {
        case “casa”:
            print “A casa é amarela”;
        case “arvore”:
            print “A árvore é bonita”;
    }

    ?>
```

5.4. Estruturas de Repetição

As estruturas de repetição são utilizadas quando o programador precisa, por exemplo, repetir o mesmo comando várias vezes. Vamos ver as estruturas de repetição existentes.

5.4.1. 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:

Sintaxes:

```
While (condição)
        comando;
```

Ou

```
While (condição) {
        comandos;
        comandos;
}
```

Exemplo:

```
<?
    $a = 0;

    while($a i<= 10) {
        echo $a.”<br>";
        $a++;
    }
?>
```

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++;
        print "<br>";

?>
```

5.4.2. 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.

Sintaxe:

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

Exemplo:

```
<?
    $a = 10;

    do {
        echo $a++;
        echo "<br>";
    }
    while ($a <=10)
?>
```

5.4.3. 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.

Sintaxe:

```
for (<inicializacao>;<condicao>;<incremento>) {
    <comando>;
    . . .
    <comando>;
}
```

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

- **Inicialização:** comando ou seqüência 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>
}
```

Exemplo:

```
<?
    For ($a = 0, $a <10, $a++) {
    Echo " O valor de A é: ". $a;
    Echo "<br>";
    }
?>
```

5.5. Quebra de fluxo

5.5.1. 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 pára imediatamente a execução do laço, seguindo normalmente o fluxo do script.

Exemplo 1:

```
<?
    $x = 20;
    while ($x < 0) {
        if ($x == 5) {
           echo "Número inválido";
        break;
        }
        echo "Número ".$x."<br>";
        $x--;
    }
?>
```

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.

5.5.2. 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.

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.

6. Funções

Sintaxe:

```
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 aos 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.

Exemplos:

```
<?
    function imprime($texto) {
        echo $texto;
    }
    imprime("teste de funções");
?>

<?
    function soma($v1, $v2) {
        $soma = $v1+$v2;
        return soma;
    }
?>
```

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 mais8($numero) {
        $numero += 8;
        echo $numero. "<br>";
    }

    $a = 3;
    mais8($a); //$a continua valendo 3

    echo $a;

?>
```

No exemplo acima, como a passagem de parâmetros é por valor, a função **mais8** 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:

Exemplo:

```
<?

    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.

Exemplo:

```
<?

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

    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.

Exemplo:

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

    teste(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.

6.1. Escopo de Variáveis

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. Os escopos de variáveis existentes no PHP são:

- Globais;
- Locais;
- Estáticas;
- Constantes;

6.2. Globais

As variáveis globais são, por definição, as variáveis que podem ser acessadas dentro de todo o script. Porém, quando se cria escopo local como nas funções, precisaremos utilizar um tipo de chamada especial, ou seja, definir aquela variável como uma variável global.

Exemplo:

```
<?
    $curso = “PHP”;
    function mostra() {
        global $curso;
        echo $curso;
    }
    mostra();
?>
```

Uma declaração “**global**” pode conter várias variáveis, separadas por vírgulas. Uma 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 caracter $. O exemplo acima e o abaixo produzem o mesmo resultado:

Exemplo:

```
<?
    $curso = “PHP”;
    function mostra() {
        echo $GLOBAL[“curso”];
        echo $curso;
    }
    mostra();
?>
```

6.3. Locais

As variáveis Locais são os tipos mais restritos dentro do PHP. Elas funcionam apenas dentro desse escopo.

Exemplo:

```
<?
    $curso = “PHP”;
    function mostra() {
        $var_local = “variable local”;
        echo $var_local;
    }
    echo “<b>$var_local</b>”;
?>
```

6.4. Estáticas

As variáveis Estáticas são variáveis que possuem o mesmo tempo de vida das variáveis globais, com a diferença de funcionarem apenas em escopos locais e serem inicializadas apenas uma vez.

Exemplo:

```
<?
    function contador() {
        static $a = 0;
        echo $a++."<br>";
    }
    for ($a = 0, $a <=5, $a++) {
        contador();
    }
?>
```

6.5. Constantes

Variáveis Constantes são variáveis que recebem um único valor que não será alterado dentro da função, permanecendo sempre o mesmo valor. Uma constante só pode conter valores escalares, não podendo conter nem um array nem um objeto.

Sintaxe:

int define(string nome_da_constante, mixed valor); onde:

- **string nome_da_contante** é o nome da variável;
- **mixed valor** é o valor que essa constante irá receber.

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

Exemplo:

```
define ("pi", 3.1415926536);
$circunf = 2*pi*$raio;
```

7. Utilizando formulários HTML

7.1. Definindo um formulário

Por ser uma linguagem de marcação, a sintaxe do HTML na maioria dos casos exige uma "tag" de início e uma de final daquele bloco. É exatamente isso que ocorre com a definição de um formulário: uma tag no início e outra no final, sendo que todos os elementos do formulário devem estar entre as duas tags. Isto torna possível a inclusão de mais de um formulário num mesmo html. As tags citadas são:

<form name="" action="" method="" enctype="">

Onde temos:

- **name:** o identificador do formulário. Utilizado principalmente em Scripts client-side (JavaScript);
- **action:** nome do script que receberá os dados do formulário ao ser submetido;
- **method:** método de envio dos dados: get ou post;
- **enctype:** formato em que os dados serão enviados. O default é urlencoded. Se for utilizado um elemento do tipo upload de arquivo (file) é preciso utilizar o tipo multipart/form-data.

Exemplo:

<**form action**="exemplo.php" **method**="post">

(textos e elementos do form)

</form>

Cada elemento do formulário deve possuir um nome que irá identificá-lo no momento em que o script indicado no ACTION for tratar os dados.

A tag <input>

Muitos elementos de um formulário html são definidos pela tag <input>. Cada tipo de elemento possui parâmetros próprios, mas todos possuem pelo menos dois parâmetros em comum: **type**, que define o tipo de elemento, e **name**, que como já foi dito define o nome daquele elemento. Existem, ainda, outros parâmetros, também chamados atributos que são comuns a maioria dos campos de um formulário. São eles:

- **Value** - o valor pré-definido do elemento, que aparecerá quando a página for carregada;
- **Size** - O tamanho do elemento na tela, em caracteres;
- **Maxlength** - O tamanho máximo do texto contido no elemento, em caracteres;

Campo de Texto (Text)

Exibe na tela um campo para entrada de texto com apenas uma linha.

Sintaxe:

<input type="**text**" name="" value="" size="" maxlength="">

Campo de Texto com Máscara (Password)

Tipo de campo semelhante ao anterior, com a diferença que neste caso os dados digitados são substituídos por asteriscos, e por isso são os mais recomendados para campos que devam conter senhas. É importante salientar que nenhuma criptografia é utilizada. Apenas não aparece na tela o que está sendo digitado.

Sintaxe:

```
<input type="password" name="" value="" size="" maxlength="">
```

Checkbox

Utilizado para campos de múltipla escolha, onde o usuário pode marcar mais de uma opção.

Sintaxe:

```
<input type="checkbox" name="" value="" checked>
```

Onde:

- **Value** - o valor que será enviado ao servidor quando o formulário for submetido, no caso do campo estar marcado;
- **Checked** - O estado inicial do elemento. Quando presente, o elemento já aparece marcado;

Radio Button

Utilizado para campos de múltipla escolha, onde o usuário pode marcar apenas uma opção. Para agrupar vários elementos deste tipo, fazendo com que eles sejam exclusivos, basta atribuir o mesmo nome a todos do grupo.

Sintaxe:

```
<input type="radio" name="" value="" checked>
```

onde:

- **Value** - o valor que será enviado ao servidor quando o formulário for submetido, no caso do campo estar marcado;
- **Checked** - O estado inicial do elemento. Quando presente, o elemento já aparece marcado;

Upload de arquivos (file)

Exibe na tela do browser um campo de texto e um botão, que ao clicado abre uma janela para localizar um arquivo no disco. Para utilizar este tipo de componente, o formulário deverá utilizar o método "**POST**" e ter o parâmetro "**enctype**" com o valor "**multipart/form-data**".

Sintaxe:

```
<form name="form1" enctype="multipart/form-data" method="post" action="">
    <input type="file" name="" size="30">
</form>
```

Onde:

- **Size** - O tamanho do campo de texto exibido.

TextArea

Exibe na tela uma caixa de texto, com o tamanho definido pelos parâmetros “cols” e “rows”.

Sintaxe:

<**textarea cols**="" **rows**="" name="" wrap="">texto</textarea>

Onde:

- **Cols** - número de colunas do campo, em caracteres;
- **Rows** - número de linhas do campo, em caracteres;
- **Wrap** - Maneira como são tratadas as quebras de linha automáticas. O valor soft faz com que o texto “quebre” somente na tela, sendo enviado para o servidor o texto da maneira como foi digitado; O valor “hard” faz com que seja enviado para o servidor da maneira como o texto aparece na tela, com todas as quebras de linhas inseridas automaticamente; o valor “off” faz com que o texto não quebre na tela e nem quando enviado ao servidor.

Obs.: O elemento do tipo textarea não possui o parâmetro “*value*”. O valor pré-definido do campo é o texto que fica entre as tags *<textarea> e </textarea>*.

Select

Campo utilizado para que o usuário faça a seleção a partir de uma lista de opções.

Sintaxe:

```
<select name="" size="" multiple>

    <option value="">texto</option>
    <option value="">texto</option>
    <option value="">texto</option>
    <option value="">texto</option>
    <option value="">texto</option>
    <option value="">texto</option>

</select>
```

Onde:

- **Size** - número de linhas exibidas. Default: 1;
- **Multiple** - parâmetro que, se presente, permite que sejam selecionadas duas ou mais linhas, através das teclas Control ou Shift;
- **Option** - Cada item do tipo “option” acrescenta uma linha ao select;
- **Value** - Valor a ser enviado ao servidor se aquele elemento for selecionado. Default: o texto do item;
- **Texto** - valor a ser exibido para aquele item. Não é definido por um parâmetro, mas pelo texto que fica entre as tags <option> e </option>

Se o parâmetro “size” tiver o valor 1 e não houver o parâmetro “multiple”, exibe na tela uma “combo box”. Caso contrário, exibe na tela uma “select list”.

Hidden

Campo oculto que é utilizado para se passar parâmetros para o servidor. Este campo não é visível para o usuário.

Sintaxe:

<input type="**hidden**" name="" value="">

Onde:

- **Value** - o parâmetro que será passado para o servidor.

Submit Button

Utilizado para enviar os dados do formulário para o script descrito na seção “action” da definição do formulário

Sintaxe:

<input type="**submit**" name="" value="">

Onde:

- **Value** - o texto que aparecerá no corpo do botão.

Reset Button

Utilizado para fazer todos os campos do formulário retornem ao valor original, quando a página foi carregada. Bastante utilizado como botão “limpar”, mas na realidade só limpa os campos se todos eles têm como valor uma string vazia.

Sintaxe:

<input type="**reset**" name="" value="">

Onde:

- **Value** - o texto que aparecerá no corpo do botão.

Button

Utilizado normalmente para ativar funções de scripts client-side (JavaScript, por exemplo). Sem essa utilização, não produz efeito algum.

Sintaxe:

<input type="**button**" name="" value="">

Onde:

- **Value** - o texto que aparecerá no corpo do botão.

No próximo tópico vamos ver como fazer para criar a interação do PHP com os formulários. Vamos ver como manipular dados do formulário com o PHP.

7.2. Interagindo o PHP com os Formulários HTML

O que você deverá observar quando criar seus formulários para manipular dados no PHP:

- ✓ Seu formulário deve conter um botão "**SUBMIT**" para poder enviar as informações;
- ✓ Todos os campos do formulário que serão tratados no script PHP devem conter o parâmetro "**NAME**", caso contrário, os dados não serão passados para o script PHP;

Como as informações do formulário são passadas para esse script PHP e como as informações do formulário enviado são tratadas, dependem de você. Existem 2 métodos como as informações podem ser passadas: **GET** e **POST**. O recomendável sempre, para todos os formulários é usar o método **POST**, onde os dados enviados **não são visíveis** nas URLs, ocultando possíveis importantes informações e permitindo o envio de longas informações. O **GET** é totalmente o contrário disso.

7.3. Variável $_POST

Esta variável é utilizada para receber as variáveis vindas do formulário pelo método post.

Sintaxe:

```
$_POST[campo_do_formulário]
```

onde:

campo_do_formulário – é o campo que foi criado no formulário que se deseja recuperar.

Exemplo:

```
<?
    $_POST[campo1];
?>
```

Vamos ver um outro exemplo utilizando um formulário HTML e um script PHP que recebe os dados do formulário.

```
<form action="script.php" method="post">
    Campo 1: <input type=text name=campo1><br>
    Campo 2: <input type=text name=campo2><br>
    <input type=submit value="OK">
</form>
```

O formulário acima usa o método POST para envio das informações, então em PHP, teremos o seguinte script:

```
<?
    echo "O valor de CAMPO 1 é: " . $_POST["campo1"];
    echo "<br>O valor de CAMPO 2 é: " . $_POST["campo2"];
?>
```

Se o formulário tivesse sido enviado usando o método **GET**, você simplesmente usaria **$_GET** no lugar de **$_POST**.

Vamos criar um pequeno exemplo de como se trabalhar com formulários no PHP. Nossa página irá enviar os dados de um formulário para o servidor e exibir esses mesmos dados numa página de resposta criada em PHP.

O formulário que iremos montar a seguir (página **form1.html**) irá solicitar que você preencha alguns dados. Ao clicar num botão ***submit***, o que você digitou e preencheu no formulário, será enviado ao servidor especificado para que possa ser produzida uma resposta, no nosso caso, ao arquivo **RespForm1.php** . O **PHP** trata esses valores como variáveis, cujo nome é o nome do campo definido no formulário. O exemplo abaixo, mostra também que o código **PHP** pode ser inserido em qualquer parte do código **HTML**.

Digite o código HTML a seguir (**form1.html**)

```
<html>
<head><title>Curso de PHP com MySQL</title></head>
<body>
<form name="form1" method="get" action="RespForm1.php">
<p>Nome:<br>
    <input name="nome" type="text" id="nome" size="30" maxlength="30"></p>
<p>Senha:<br><input name="senha" type="password" id="senha" size="10" maxlength="10"></p>
<p>Sexo:<br>
    <input name="sexo" type="radio" value="Masculino">Masculino
    <input name="sexo" type="radio" value="Feminino">Feminino</p>
<p>Selecione o Turno:<br>

    <input name="turno" type="checkbox" id="turno" value="Manhã">Manhã
    <input name="turno" type="checkbox" id="turno" value="Tarde">Tarde
    <input name="turno" type="checkbox" id="turno" value="Noite">Noite</p>

<p>Cidade:<br>
<select name="cidade" id="cidade">

    <option value="vazio"> </option>
    <option value="Rio">Rio de Janeiro</option>
    <option value="São Paulo">São Paulo</option>
    <option value="Belo Horizonte">Belo Horizonte</option>

</select></p>
<p><input type="submit" value="Enviar"></p>
<input type='hidden' name='btnOK' value='1'>
</form>
</body>
</html>
```

Salve o arquivo na sua pasta com o nome **Form1.html**. Agora, vamos criar o arquivo **RespForm1.php**.

Arquivo RespForm1.php

```
<?

    // Recebe os dados do formulário com a variável $_POST

    $nome = $_POST["nome"];
    $senha = $_POST["senha"];
    $sexo = $_POST["sexo"];
    $turno = $_POST["turno"];
    $cidade = $_POST["cidade"];

    // Exibe os dados na página de resposta: RespForm.php

    echo "Os dados recebidos do formulário são: <p>";
    echo "
            <table width = '400' border='1' cellspacing='0' cellpadding='0'>
        <tr>
                <td width = '100'>Nome:<td>
                <td width = '300'>$nome</td>
        </tr>
        <tr>
                <td width = '100'>Senha:<td>
                <td width = '300'>$senha</td>
        </tr>
        <tr>
                <td width = '100'>Sexo:<td>
                <td width = '300'>$sexo</td>
        </tr>
        <tr>
                <td width = '100'>Turno:<td>
                <td width = '300'>$turno</td>
        </tr>
        <tr>
                <td width = '100'>Cidade:<td>
                <td width = '300'>$cidade</td>
        </tr>
        </table>
    ";

?>

<html>
<head><title>Curso de PHP</title>
</head>
<body>
<p><font face="Arial, Helvetica, sans-serif" size="4">
<a href="form1.html">Clique aqui para voltar ao formul&aacute;rio.</a>
</font>
</body>
</html>
```

Após salvar os arquivos **Form1.html** e **RespForm1.php**, abra o **browser** e, na linha de endereço digite:

http://localhost/seudiretorio(o diretório virtual criado no IIS)/Form1.html.

Você verá apenas um formulário que contem os campos para você preencher, conforme mostra a figura 01 a seguir.

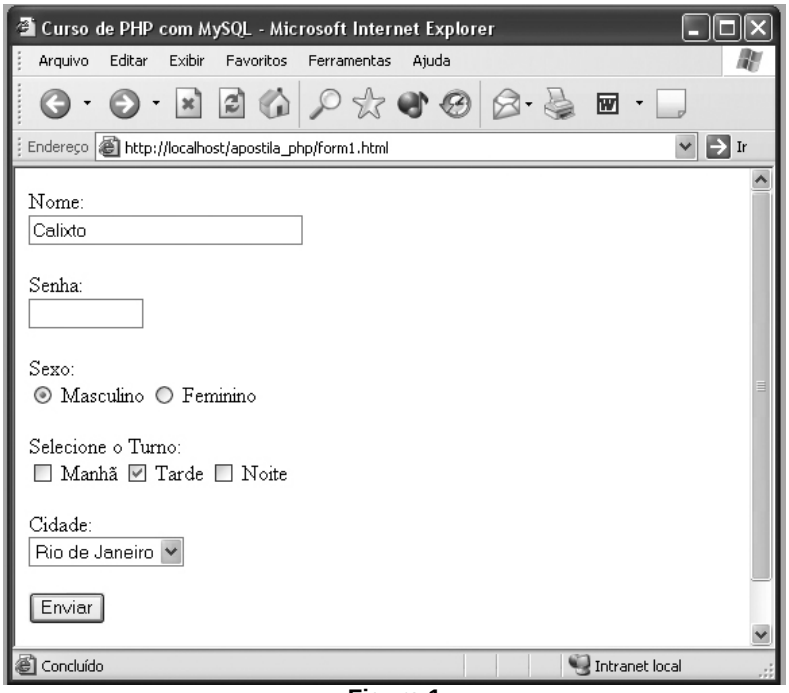

Figura 1

Após preencher o formulário, clique no botão **Enviar** para submeter este formulário ao servidor. Uma nova página, que é a resposta do servidor, deverá aparecer conforme mostra a figura 2 a seguir.

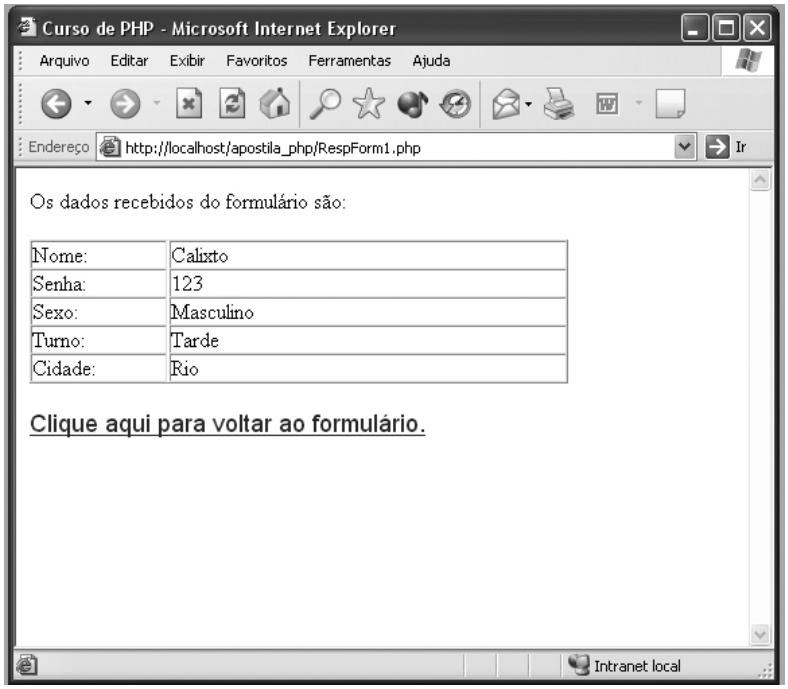

Figura 2

Observações:

Em vez de usar **$_GET** ou **$_POST** você pode escrever a variável com o mesmo nome do campo do formulário (no exemplo, $campo1 e $campo2). Mas, esse uso não é recomendado, pois se a diretiva "register_globals" na configuração do seu PHP estiver desativada, as variáveis com nome dos campos dos formulários, terão um valor vazio.

Uma solução para isso é usar a função **import_request_variables** no começo dos seus scripts que interpretam formulários. Essa função aceita 3 letras como argumentos: P, G e C, referentes a $_POST, $_GET e $_COOKIE respectivamente.

Exemplo:

```
<?

import_request_variables("gP");

?>
```

O que acontece?

Exemplo:

Você possui formulário com os campos "nome", "endereço" e "idade". Assuma que a diretiva "register_globals" do seu PHP esteja desligada, mas, você já havia programado o script usando as variáveis no escopo global, no lugar de $_POST.

Adicionando aquela função no começo do script, as variáveis do seu formulário postado:

$_POST["nome"], $_POST["endereco"] e $_POST["idade"] serão extraídas cada para uma variável diferente: $nome, $endereco e $idade.

8. Trabalhando com MySQL

O MySQL surgiu a partir da necessidade da equipe que criou o SGBD, de utilizar algum mecanismo que permitisse a conexão de tabelas criadas na linguagem SQL para um determinado fim. A princípio, o grupo iria utilizar o mSQL, mas logo perceberam que esta ferramenta não era rápida o suficiente para atender às necessidades do projeto. O jeito foi criar uma solução própria. Nascia o MySQL.

O MySQL foi criado por Michael Widenius na companhia suíça TcX. Por volta de 1979 Michael desenvolveu um banco de dados chamado UNIREG, sendo rescritos em várias linguagens desde então" [YAR 99]. Em 1994, a empresa TcX começou o desenvolvimento de aplicações baseadas na Web, tendo como base o banco UNIREG, porém esse banco possuía muito "overhead" para obter sucesso em uma aplicação para geração de páginas dinâmicas na Web. Então a empresa TcX começou a procurar por outro banco o mSQL, uma ferramenta baseada em SQL mas com características pobres não possuindo por exemplo suporte a índices, e com desempenho inferior ao UNIREG.

Foi então que o desenvolvedor do banco UNIREG contatou o David Hughes criador do mSQL, para saber do interesse dele em unir os dois bancos. Sendo positivo o interesse de David , a empresa TcX resolveu desenvolver um novo banco, mas mantendo ao máximo a compatibilidade com mSQL. TcX foi esperta o suficiente para não reinventar o que já estava bem feito, ela construiu seu servidor baseado na estrutura que já estava montada do UNIREG e utilizou grande número de utilitários escritas para mSQL e fez API's para o novo servidor praticamente iguais ao mSQL. Como resultado usuários do mSQL que decidissem mudar para o novo servidor da TcX, teriam apenas que fazer pequenas e simples mudanças nos códigos existentes.

Então foi em maio de 1995 que, definitivamente, a primeira versão do MySQL foi lançada. Um dos parceiros da TcX sugeriu a distribuição do servidor na Internet, o objetivo disso era a utilização de um modelo pioneiro desenvolvido por Aladdin Peter Deutsch. O resultado foi um maior flexibilidade em sem "copyright", que fez do MySQL mais difundido gratuitamente do que mSQL.

8.1. O Banco de Dados MySQL

O MySQL foi desenvolvido pela TCX em 1996. Atualmente a MySQL AB desenvolve o programa. MySQL AB é a companhia dos fundadores e principais desenvolvedores do MySQL. Eles criaram-no porque precisavam de um banco de dados relacional que pudesse tratar grandes quantidades de dados em máquinas de custo relativamente barato.

É um dos bancos de dados relacionais mais rápidos do mercado, apresenta quase todas as funcionalidades dos grandes bancos de dados . MySQL é uma linguagem simples, em que você facilmente pode gravar, alterar e recuperar informações num web site com segurança e rapidez O MYSQL é executado, principalmente, em sistemas que participam da filosofia UNIX, embora outros sistemas S.O também fornecem suporte, como Windows, por exemplo.

O Servidor MySQL foi desenvolvido originalmente para lidar com bancos de dados muito grandes de maneira muito mais rápida que as soluções existentes e tem sido usado em ambientes de produção de alta demanda por diversos anos de maneira bem sucedida. Apesar de estar em constante desenvolvimento, o Servidor MySQL oferece hoje um rico e proveitoso conjunto de funções. A conectividade, velocidade, e segurança fazem com que o MySQL seja altamente adaptável para acessar bancos de dados na Internet.

O Programa de Banco de Dados MySQL é um sistema cliente/servidor que consiste de um servidor SQL multitarefa que suporta acessos diferentes, diversos programas clientes e bibliotecas, ferramentas administrativas e diversas interfaces de programação (API's). Também concedemos o Servidor MySQL como uma biblioteca multitarefa que você pode ligar à sua aplicação para chegar a um produto mais rápido, menor e mais facilmente gerenciável.

8.2. Características do MySQL

- Multi-plataforma, portanto suporta diferentes plataformas: Win32, Linux, FreeBSD, Unix, etc...
- Suporte a múltiplos processadores
- Um sofisticado sistema de senhas criptografadas flexível e Seguro.
- Suporta até 16 índices por tabela
- Código fonte escrito em C e C++ e testado com uma variedade de diferentes compiladores
- As tabelas criadas podem ter tamanho de até 4 GB
- Banco de dados de código aberto e gratuito
- Suporte as API´s das Seguintes linguagens: PHP, Perl, C,C++,Java, Pynthon, etc...
- Suporte à ODBC, você pode facilmente conectar o Access a um banco de dados do MySQL
- O Cliente conecta no MySQL através de conexões TCP/IP.
- Nenhum problema com o Y2K, visto que o MySQL usa o relógio do Unix que não apresentará problemas até 2069
- Capacidade para manipular bancos com até 50 milhões de registros
- Reduz a administração, engenharia e a sustentação custa por até 50%

8.3. O que o MySQL faz de melhor

- Aplicações Web
- Suporte a código fonte aberto
- Requisitos de sistema baixo
- Tabelas com tamanho grande
- Estabilidade

8.3. O MySQL é gratuito ?

Pessoas confundem "free" com "grátis" o que é comum aqui no Brasil. Mas em se tratando de software este "free" é de open source e não gratuito.

Para poder utilizar o MySQL sob a licença GPL e não precisar pagar, o produto desenvolvido precisa ser GPL também, senão, orientamos a compra da licença comercial, com baixo custo, sendo comercializada por servidor, sem limites de usuários e processadores e ainda com garantia perpétua de atualização de versão para o resto da vida.

8.4. Tipos de dados do MySQL

Os tipos de dados que pode ter um campo, podem-se agrupar em três grandes grupos:

1. Tipos numéricos
2. Tipos de Data
3. Tipos de Cadeia

1) Tipos numéricos:

Existem tipos de dados numéricos, que se podem dividir em dois grandes grupos, os que estão em vírgula flutuante (com decimais) e os que não.

TinyInt: é um número inteiro com ou sem sinal. Com sinal a margem de valores válidos é desde -128 até 127. Sem sinal, a margem de valores é de 0 até 255

SmallInt: número inteiro com ou sem sinal. Com sinal a margem de valores válidos é desde -32768 até 32767. Sem sinal, a margem de valores é de 0 até 65535.

MediumInt: número inteiro com ou sem sinal. Com sinal a margem de valores válidos é desde -8.388.608 até 8.388.607. Sem sinal, a margem de valores é de 0 até 16777215.

Int: número inteiro com ou sem sinal. Com sinal a margem de valores válidos é desde -2147483648 até 2147483647. Sem sinal, a margem de valores é de 0 até 429.496.295

BigInt: número inteiro com ou sem sinal. Com sinal a margem de valores válidos é desde -9.223.372.036.854.775.808 até 9.223.372.036.854.775.807. Sem sinal, a margem de valores é de 0 até 18.446.744.073.709.551.615.

Float: número pequeno em vírgula flutuante de precisão simples. Os valores válidos vão desde -3.402823466E+38 até -1.175494351E-38,0 eté desde 175494351E-38 até 3.402823466E+38.

Double: número em vírgula flutuante de dupla precisão. Os valores permitidos vão desde -1.7976931348623157E+308 até -2.2250738585072014E-308, 0 e desde 2.2250738585072014E-308 até 1.7976931348623157E+308

Decimal: Número em vírgula flutuante desempacotado. O número armazena-se como uma cadeia.

Tipo de Campo	**Tamanho de Armazenamento**
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
BIGINT	8 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes
DECIMAL(M,D)	M+2 bytes se D > 0, M+1 bytes se D = 0

2) Tipos data:

Na hora de armazenar datas, há que ter em conta que MySQL não verifica de uma maneira estricta se uma data é válida ou não. Simplesmente comprova que o mês está compreendido entre 0 e 12 e que o dia está compreendido entre 0 e 31.

Date: tipo data, armazena uma data. A margem de valores vai desde o 1 de Janeiro de 1001 ao 31 de dezembro de 9999. O formato de armazenamento é de ano-mes-dia.

DateTime: Combinação de data e hora. A margem de valores vai desde o 1 ed Janeiro de 1001 às 0 horas, 0 minutos e 0 segundos ao 31 de Dezembro de 9999 às 23 horas, 59 minutos e 59 segundos. O formato de armazenamento é de ano-mes-dia horas:minutos:segundos

TimeStamp: Combinação de data e hora. A margem vai desde o 1 de Janeiro de 1970 ao ano 2037. O formato de armazenamento depende do tamanho do campo:

Tamanhodo campo	Formato
14	AnoMesDiaHoraMinutoSegundo aaaammddhhmmss
12	AnoMesDiaHoraMinutoSegundo aammddhhmmss
8	AnoMesDia aaaammdd
6	AnoMesDia aammdd
4	AnoMes aamm
2	Ano aa

Time: armazena uma hora. A margem de horas vai desde -838 horas, 59 minutos e 59 segundos. O formato de armazenamento é 'HH:MM:SS'.

Year: armazena um ano. A margem de valores permitidos vai desde o ano 1901 ao ano 2155. O campo pode ter tamanho dois ou tamanho 4 dependendo de se queremos armazenar o ano com dois ou quatro algarismos.

Tipo de Campo	Tamanho de Armazenamento
DATE	3 bytes
DATETIME	8 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

3 Tipos de cadeia:

Char(n): armazena uma cadeia de longitude fixa. A cadeia poderá conter desde 0 até 255 caracteres.

VarChar(n): armazena uma cadeia de longitude variável. A cadeia poderá conter desde 0 até 255 caracteres. Dentro dos tipos de cadeia pode-se distinguir dois subtipos, os tipo Test e os tipo Blob (Binary Large Object) A diferença entre um tipo e outro é o tratamento que recebem na hora de ordená-los e compará-los. No tipo test ordena-se sem ter importância as maiúsculas e as minúsculas e no tipo blob ordena-se tendo em conta as maiúsculas e minúsculas.

TinyText e TinyBlob: Coluna com uma longitude máxima de 255 caracteres.

Blob e Text: um texto com um máximo de 65535 caracteres.

MediumBlob e MediumText: um texto com um máximo de 16.777.215 caracteres.

LongBlob e LongText: um texto com um máximo de caracteres 4.294.967.295. Há que ter em conta que devido aos protocolos de comunicação os pacotes podem ter um máximo de 16 Mb.

Enum: campo que pode ter um único valor de uma lista que se especifica. O tipo Enum aceita até 65535 valores diferentes.

Tipo de campo	**Tamanho de Armazenamento**
CHAR(n)	n bytes
VARCHAR(n)	n +1 bytes
TINYBLOB, TINYTEXT	Longitude+1 bytes
BLOB, TEXT	Longitude +2 bytes
MEDIUMBLOB, MEDIUMTEXT	Longitude +3 bytes
LONGBLOB, LONGTEXT	Longitude +4 bytes
ENUM('value1','value2',...)	1 ó dos bytes dependendo do número de valores

8.5. Comandos Básicos do SQL

Iremos ver os comandos básicos do SQL, utilizados pela maioria dos bancos de dados , inclusive o MySQL, que serão necessários para o desenvolvimento do nosso projeto, pois o objetivo deste curso não é aprender o SQL, mas sim, o PHP.

Observe que todo comando SQL termina com um **; (ponto e vírgula)**.

Comando Create Database

Este comando permite a criação do banco de dados.

Sintaxe:

```
CREATE DATABASE < nome_db >;
```

onde:

- **nome_db** - indica o nome do Banco de Dados a ser criado.

Exemplo:

```
Create database focus;
```

Comando Create Table

Este comando permite a criação de tabelas no banco de dados.

Sintaxe:

```
CREATE TABLE < nome_tabela > (
nome_atributo1 < tipo > [ NOT NULL ],
nome_atributo2 < tipo > [ NOT NULL ],
.....
nome_atributoN < tipo > [ NOT NULL ]
) ;
```

onde:

- **nome_table** - indica o nome da tabela a ser criada.
- **nome_atributo** - indica o nome do campo a ser criado na tabela.
- **tipo -** indica a definição do tipo de atributo (integer(n), char(n), ...).

Exemplo:

```
Create table alunos (
Id_aluno UNSIGNED INT(3) NOT FULL,
nome CHAR(40) NOT NULL,
endereco CHAR (50) NOT FULL
turma CHAR(20) NOT NULL,
PRIMARY KEY (matricula)
);
```

Comando Drop

Este comando elimina a definição da tabela, seus dados e referências.

Sintaxe:

DROP TABLE < nome_tabela > ;

Exemplo:

Drop table alunos;

Comando Alter

Este comando permite inserir/eliminar atributos nas tabelas já existentes.

Sintaxe:

ALTER TABLE < nome_tabela > ADD / DROP (
nome_atributo1 < tipo > [NOT NULL],
nome_atributoN < tipo > [NOT NULL]
) ;

Exemplo:

Alter table alunos **ADD COLUMN** turno char(10) NOT NULL;

8.6. Manipulando dados das tabelas

Comando SELECT

Permite recuperar informações existentes nas tabelas.

Sintaxe:

SELECT [DISTINCT] expressao [AS nom-atributo] [FROM from-list] [WHERE condicao] [ORDER BY attr_name1 [ASC | DESC]

onde:

- **DISTINCT** - Para eliminar linhas duplicadas na saída.
- **Expressão** - Define os dados que queremos na saída, normalmente uma ou mais colunas de uma tabela da lista FROM.
- **AS nom-atributo** - um alias para o nome da coluna, exemplo:
- **FROM** - lista das tabelas na entrada
- **WHERE** - critérios da seleção
- **ORDER BY** - Critério de ordenação das tabelas de saída. Podem ser:
    - **ASC** - ordem ascendente (crescente);
    - **DESC** - ordem descendente (decrescente)

Exemplo:

Select cidade, estado **from** brasil **where** populacao > 100000 **order by** Desc;

Comando INSERT

Adiciona um ou vários registros a uma tabela. Isto é referido como consulta anexação.

Sintaxe:

INSERT INTO destino [(campo1[, campo2[, ...]])]
VALUES (valor1[, valor2[, ...])

Onde;

- **Destino -** O nome da tabela ou consulta em que os registros devem ser anexados.
- **campo1, campo2** - Os nomes dos campos aos quais os dados devem ser anexados
- **valor1, valor2** - Os valores para inserir em campos específicos do novo registro. Cada valor é inserido no campo que corresponde à posição do valor na lista: Valor1 é inserido no campo1 do novo registro, valor2 no campo2 e assim por diante.

Os valores devem ser separados com uma vírgula e os campos de textos entre aspas duplas ou simples.

Exemplo:

Insert into alunos (Id_aluno, nome, endereço, turma, turno)
Values (1, 'Glaucio', 'Av. das Américas', '1101', 'manhã');

Comando UPDATE

Cria uma consulta atualização que altera os valores dos campos em uma tabela especificada com base em critérios específicos.

Sintaxe:

UPDATE tabela SET campo1 = valornovo, ... WHERE critério;

Onde:

- **Tabela** - O nome da tabela cujos dados você quer modificar.
- **Valornovo** - Uma expressão que determina o valor a ser inserido em um campo específico nos registros atualizados.
- **critério** - Uma expressão que determina quais registros devem ser atualizados. Só os registros que satisfazem a expressão são atualizado.

Exemplo:

Update alunos **Set** turno = 'tarde' **where** turma = '1101';

UPDATE é especialmente útil quando você quer alterar muitos registros ou quando os registros que você quer alterar estão em várias tabelas. Você pode alterar vários campos ao mesmo tempo.

UPDATE não gera um conjunto de resultados. Se você quiser saber quais resultados serão alterados, examine primeiro os resultados da consulta seleção que use os mesmos critérios e então execute a consulta atualização.

Comando DELETE

Remove registros de uma ou mais tabelas listadas na cláusula FROM que satisfaz a cláusula WHERE.

Sintaxe:

```
DELETE [tabela.*]
FROM tabela
WHERE critério
```

onde:

tabela.* - O nome opcional da tabela da qual os registros são excluídos.

tabela - O nome da tabela da qual os registros são excluídos.

critério - Uma expressão que determina qual registro deve ser excluído.

Exemplo:

```
Delete from alunos WHERE turno='Manhã';
```

DELETE é especialmente útil quando você quer excluir muitos registros.

Para eliminar uma tabela inteira do banco de dados, você pode usar o método Execute com uma instrução DROP.

Entretanto, se você eliminar a tabela, a estrutura é perdida. Por outro lado, quando você usa DELETE, apenas os dados são excluídos. A estrutura da tabela e todas as propriedades da tabela, como atributos de campo e índices, permanecem intactos.

Você pode usar DELETE para remover registros de tabelas que estão em uma relação um por vários com outras tabelas. Operações de exclusão em cascata fazem com que os registros das tabelas que estão no lado "vários" da relação sejam excluídos quando os registros correspondentes do lado "um" da relação são excluídos na consulta.

Por exemplo, nas relações entre as tabelas Clientes e Pedidos, a tabela Clientes está do lado "um" e a tabela Pedidos está no lado "vários" da relação. Excluir um registro em Clientes faz com que os registros correspondentes em Pedidos sejam excluídos se a opção de exclusão em cascata for especificada.

O DELETE exclui registros inteiros e não apenas dados em campos específicos. Se você quiser excluir valores de um campo específico, crie uma consulta atualização que mude os valores para Null.

Após remover os registros usando uma consulta exclusão, você não poderá desfazer a operação. Se quiser saber quais arquivos foram excluídos, primeiro examine os resultados de uma consulta seleção que use o mesmo critério e então, execute a consulta exclusão. Mantenha os backups de seus dados. Se você excluir os registros errados, poderá recuperá-los a partir dos seus backups.

9. Conexão com o MySQL via PHP

Todos os exemplos e arquivos do nosso projeto referentes a acesso de banco de dados utilizarão o gerenciador de banco de dados MySQL que encontra-se disponível no CD que acompanha este curso de PHP. Estamos utilizando a versão 5.x do MySQL. No CD estão disponíveis as versões 4.x e 5.x.

Para interagir com uma base de dados SQL existem três comandos básicos que devem ser utilizados: um que faz a conexão com o servidor de banco de dados, um que seleciona a base de dados a ser utilizada e um terceiro que executa uma "query" (consulta) SQL. Existe, ainda, um quarto comando que pode ou ser utilizado, que é o comando que fecha a conexão com o banco de dados.

São eles:

9.1. Abrindo e fechando conexão com o MySQL

Mysql_connect( )

Comando utilizado para criar a conexão com a base de dados num servidor mySQL.

O comando **mysql_pconnect** também pode ser utilizado para criar a conexão. A diferença entre os dois comandos é que o **mysql_pconnect** estabelece uma conexão permanente, ou seja, que não é encerrada ao final da execução do script. Utilizaremos a primeira opção: ***mysql_connect***.

Sintaxe:

```
mysql_connect(string [host[:porta]] , string [login] , string [senha] ) [ or die ("mensagem de erro")];
```

onde:

- **string [host[:porta]]** – é o endereço do servidor, onde o banco está armazenado;
- **string [login]** – é o usuário do banco de dados;
- **string [senha]** – é a senha do usuário do banco de dados MySQL.
- **die** – parâmetro opcional que exibe uma mensagem indicando que a conexão não foi efetuada.

Exemplo:

```
<?
    $conexao = mysql_connect ("localhost", "root", "focus") or die ("Conexão não efetuada");
?>
```

O valor de retorno é um inteiro que identifica a conexão, ou falso se a conexão falhar. Antes de tentar estabelecer uma conexão, o interpretador PHP verifica se já existe uma conexão estabelecida com o mesmo host, o mesmo login e a mesma senha. Se existir, o identificador desta conexão é retornado. Senão, uma nova conexão é criada.

Assim, se a conexão for bem sucedida (existir um servidor no endereço especificado que possua o usuário com a senha fornecida), o identificador da conexão fica armazenado na variável $conexao, caso contrário será mostrada a mensagem "Conexão não efetuada".

Mysql_close( )

Comando utilizado para encerrar uma conexão estabelecida com o comando mysql_connect antes de chegar ao final do script. Caso esse comando não seja utilizado a conexão é encerrada no final do script.

Sintaxe:

mysql_close(int [string da conexão]);

Se o identificador não for fornecido, a última conexão estabelecida será encerrada.

Exemplo:

```
<?

    mysql_close ($conexao);

?>
```

Obs.: O comando mysql_close não encerra conexões estabelecidas com o *mysql_pconnect*.

Mysql_select_db( )

Comando utilizado para selecionar a base de dados depois que a conexão for estabelecida. Se nenhuma de conexão é especificado, a ultima conexão aberta é assumida. Se nenhuma conexão esta aberta, a função irá tentar abrir uma conexão como se **mysql_connect()** fosse chamada sem argumentos e usá-la.

Sintaxe:

mysql_select_db(banco de dados, [string de conexao]) or die [(“mensagem”)];

Onde:

- **banco de dados – é o banco de dados que será utililzado;**
- **string de conexão – é a conexão criada com o servidor MySQL.**
- **die** – parâmetro opcional que exibe uma mensagem indicando que a conexão não foi efetuada.

Exemplo:

```
<?

    $conexao = mysql_connect(“localhost”, “root”, “focus”);

    if ($conexao) {
       die (“Conexão não estabelecida”);
    }

    db_select = mysql_select_db(“focus”, $conexao);

?>
```

Mysql_query( )

Este comando é utilizado para realizar uma consulta SQL no MySQL.

Sintaxe:

Mysql_query(string da consulta);

Onde:

String da consulta – é um dos comandos utilizados do SQL para efetuar uma consulta, uma inclusão, uma alteração ou uma exclusão no banco de dados.

O comando mysql_query() envia uma query (consulta) para o banco de dados ativo no servidor da conexão informada em **string da consulta**. Se o parâmetro **string da consulta** não é especificado, a ultima conexão aberta é usada. Se nenhuma conexão esta aberta, a função tenta estabelecer uma conexão como mysql_connect() seja chamada sem argumentos e usá-la. O resultado é guardado em buffer.

Exemplo:

```
<?

    sql = “select * from alunos where id_aluno = 10”;

    mysql_query (sql);

?>
```

Vamos ver um exemplo de um script PHP utilizando os comandos vistos acima que cria uma conexão com o banco de dados Focus, seleciona a tabela alunos e insere alguns dados na tabela.

```
<?
    $conexao = mysql_connect (“localhost”, “root”, “focus”);
    mysql_select_db(“focus”, $conexão);

    $insere = “Insert into alunos (id_aluno, nome, endereço, turma, turno)
    Values (1, ‘Glaucio’, ‘Av. das Américas’, ‘1101’, ‘manhã’)”;

    $insere1 = “Insert into alunos (id_aluno, nome, endereço, turma, turno)
    Values (2, ‘Alexandre, ‘Av. das Américas’, ‘1101’, ‘tarde’)”;

    $insere2 = “Insert into alunos (id_aluno, nome, endereço, turma, turno)
    Values (3, ‘Lucinaldo’, ‘Av. das Américas’, ‘1101’, ‘tarde’)”;

    mysql_query ($insere, $conexao) or die (“Não foi possível executar a inserção.”);
    mysql_query ($insere1, $conexao) or die (“Não foi possível executar a inserção.”);
    mysql_query ($insere2, $conexao) or die (“Não foi possível executar a inserção.”);

    $delete = Delete from alunos where turno = “tarde”);

    mysql_query ($delete, $conexao);

    echo (“ todos os alunos do turno da tarde foram excluídos.”);
    mysql_close ($conexao);
?>
```

9.2. Tratamento de resultados de query Select

Ao executar uma query SQL SELECT através do comando mysql_query, o identificador do resultado deve ser armazenado numa variável que pode ser tratada de diversas formas. Duas maneiras interessantes de fazê-lo usam um dos comandos a seguir: **mysql_result, mysql_fetch_row ou mysql_fetch_array**.

Mysql_result( )

Esta função retorna o resultado de uma query SQL.

Sintaxe:

mysql_result(resultado, linha, mixed [campo]);

Onde:

- **resultado** - é o identificador do resultado, obtido com o retorno da função mysql_query;
- **linha** - especifica o registro a ser exibido, já que uma query SELECT pode retornar diversos registros;
- **campo** - é o identificador do campo a ser exibido, sendo o tipo descrito como mixed pela possibilidade de ser de diversos tipos (neste caso, inteiro ou string).

Exemplo:

```
<?
    $conexao = mysql_connect (“localhost”, “root”, “focus”);
    mysql_select_db(“focus”, $conexão);

    $insere = “Insert into alunos (id_aluno, nome, endereço, turma, turno)
              Values (1, ‘Glaucio’, ‘Av. das Américas’, ‘1101’, ‘manhã’)”;

    $insere1 = “Insert into alunos (id_aluno, nome, endereço, turma, turno)
              Values (2, ‘Alexandre, ‘Av. das Américas’, ‘1101’, ‘tarde’)”;

    mysql_query ($insere, $conexao) or die (“Não foi possível executar a inserção.”);
    mysql_query ($insere1, $conexao) or die (“Não foi possível executar a inserção.”);

    $consulta = “Select Id_Aluno, nome, turno from alunos”;
    $resultado = mysql_query ($consulta, $conexao);
    $nome = mysql_result($resultado,0,"nome");
    $turno = mysql_result($resultado,0,”turno”);

    echo (“Nome: “.$nome.”<p>”.“Turno: “.$turno);

    mysql_close ($conexao);
?>
```

Com o exemplo acima, o resultado será:

Nome: Gláucio

Turno: Manhã

Mysql_fetch_array( )

Esta função lê uma linha do resultado e devolve um array, cujos índices são os nomes dos campos. A execução seguinte do mesmo comando lerá a próxima linha, até chegar ao final do resultado.

Sintaxe:

Mysql_fecth_array(string da cosulta SQL);

Exemplo:

```
<?

    $conexao = mysql_connect (“localhost”, “root”, “focus”);
    mysql_select_db(“focus”, $conexão);

    $insere = “Insert into alunos (id_aluno, nome, endereço, turma, turno)
              Values (1, ‘Glaucio’, ‘Av. das Américas’, ‘1101’, ‘manhã’)”;

    $insere1 = “Insert into alunos (id_aluno, nome, endereço, turma, turno)
              Values (2, ‘Alexandre, ‘Av. das Américas’, ‘1101’, ‘tarde’)”;

    mysql_query ($insere, $conexao) or die (“Não foi possível executar a inserção.”);
    mysql_query ($insere1, $conexao) or die (“Não foi possível executar a inserção.”);

    $consulta = “Select Id_Aluno, nome, turno from alunos”;
    $resultado = mysql_query ($consulta, $conexao);

    $dados = mysql_fecth_array($resultado);

    $nome = $dados[“nome”];
    $turno =$dados[turno];

    echo (“Nome: “.$nome.”<p>”.“Turno: “.$turno);

    mysql_close ($conexao);

?>
```

Com o exemplo acima, o resultado será:

Nome: Gláucio
Turno: Manhã

Mysql_fetch_row( )

Esta função é semelhante a função mysql_fetch_array, com a diferença que os índices do array são numéricos, iniciando pelo 0 (zero).

Sintaxe:

Mysql_fetch_row(string de consulta);

Exemplo:

O exemplo anterior substituindo a função **mysql_fecth_array** pela **mysql_fetch_row**.

Mysql_free_result( )

Esta função libera a memória do resultado de uma consulta.

Sintaxe:

mysql_free_result (resource result)

Exemplo:

```
<?

    $conexao = mysql_connect (“localhost”, “root”, “focus”);
    mysql_select_db(“focus”, $conexão);

    $insere = “Insert into alunos (id_aluno, nome, endereço, turma, turno)
              Values (1, ‘Glaucio’, ‘Av. das Américas’, ‘1101’, ‘manhã’)”;

    $insere1 = “Insert into alunos (id_aluno, nome, endereço, turma, turno)
              Values (2, ‘Alexandre, ‘Av. das Américas’, ‘1101’, ‘tarde’)”;

    mysql_query ($insere, $conexao) or die (“Não foi possível executar a inserção.”);
    mysql_query ($insere1, $conexao) or die (“Não foi possível executar a inserção.”);

    $consulta = “Select Id_Aluno, nome, turno from alunos”;
    $resultado = mysql_query ($consulta, $conexao);

    $dados = mysql_fecth_array($resultado);

    $nome = $dados[“nome”];
    $turno =$dados[turno];

    echo (“Nome: “.$nome.”<p>”.“Turno: “.$turno);

    mysql_free_result($resultado);

    mysql_close ($conexao);

?>
```

No exemplo acima, a função **mysql_free_result( )** irá liberar toda a memória usada com o identificador de resultado $resultado.

Obs.: mysql_free_result() somente precisa ser chamado se você esta preocupado em quanta memória esta sendo usada para query num grande conjunto de resultados. Toda a memória usada do resultado é liberada automaticamente ao final da execução do script.

10. Utilizando headers

O comando header permite enviar cabeçalhos html para o cliente. Deve ser utilizado por usuários que conheçam a função de cada header que pode ser enviado.

Sintaxe:

Header (string header);

Exemplo:

```
<?
    header (“location: login.php”);
?>
```

Não pode ser enviado depois de algum texto. veja o seguinte exemplo:

```
<html>
<? header("Location: http://www.php.net"); ?>
<body>
...
```

O código acima causará um erro, já que tentou-se enviar um header depois de ter sido enviado um texto(“<html>\n”).

Algumas utilizações do header são:

- redirecionar para outra página:

  Exemplo:

  ```
  header("Location: http://www.php.net");
  ```

- Definir o script como uma mensagem de erro:

  Exemplo:

  ```
  header("http/1.0 404 Not Found");
  ```

- Definir a expiração da página:

  Exemplos:

  ```
  header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
  header("Pragma: no-cache"); // HTTP/1.0
  ```

11. Utilizando cookies

11.1. O que são Cookies

Cookies são variáveis gravadas no cliente(browser) por um determinado site. Somente o site que gravou o cookie pode ler a informação contida nele. Este recurso é muito útil para que determinadas informações sejam fornecidas pelo usuário apenas uma vez. Exemplos de utilização de cookies são sites que informam a quantidade de vezes que você já visitou, ou alguma informação fornecida numa visita anterior.

Existem cookies persistentes e cookies de sessão. Os persistentes são aqueles gravados em arquivo, e que permanecem após o browser ser fechado, e possuem data e hora de expiração. Os cookies de sessão não são armazenados em disco e permanecem ativos apenas enquanto a sessão do browser não for encerrada.

Por definição, existem algumas limitações para o uso de cookies, listadas a seguir:

- 300 cookies no total
- 4 kilobytes por cookie.
- 20 cookies por servidor ou domínio.

11.2. Gravando cookies

Para gravar cookies no cliente, deve ser utilizada a função setcookie.

Sintaxe:

setcookie(string nome, string valor, int exp, string path,string dominio, int secure);

onde:

- **nome** - nome do cookie;
- **valor** - valor armazenado no cookie;
- **exp** - data de expiração do cookie (opcional), no formato Unix. Se não for definida, o cookie será de sessão;
- **path** - path do script que gravou o cookie;
- **domínio** - domínio responsável pelo cookie;
- **secure** - se tiver valor 1, indica que o cookie só pode ser transmitido por uma conexão segura (https).

Observações:

- Um cookie não pode ser recuperado na mesma página que o gravou, a menos que esta seja recarregada pelo browser;
- Múltiplas chamadas à função setcookie serão executadas em ordem inversa;
- Cookies só podem ser gravados antes do envio de qualquer informação para o cliente. Portanto todas as chamadas à função setcookie devem ser feitas antes do envio de qualquer header ou texto.

11.3. Lendo cookies gravados

Os cookies lidos por um script PHP ficam armazenados em duas variáveis:

- No array $HTTP_COOKIE_VARS[], tendo como índice a string do nome do cookie;
- E numa variável cujo nome é o mesmo do cookie, precedido pelo símbolo $.

Exemplo:

Um cookie que foi gravado numa página anterior pelo seguinte comando:

setcookie(“teste”, “meu cookie”);

Pode ser lida pela variável **$HTTP_COOKIE_VARS[“teste”]** ou pela variável **$teste**.

O script a seguir é um exemplo do uso de cookies. O script exibe quantas vezes o usuário visitou aquela página.

Arquivo cookie.php

```
<?

    error_reporting(134);
    /*
     * Erro no script
     * Nome do seu Script
     * Por Seu nome ou nick
     * e-mail = seumail@qualquercoisa.com.br
     */

    ?>

    <?

    if ($Test != '') {
       $Test++;
       setcookie("Test",$Test, time()+3600000); // selecionar o nome do cokie
    } else {
       setcookie("Test",1, time()+3600000);  //Criar o valor 1
       $Test = 1;
    }
    $numvisits = $Test;

?>
```

Esse é o código completo do arquivo **cookie.php**. Agora, digite o arquivo **visitas.php** que irá informar o número de vezes que o site foi visitado por um mesmo usuário.

Arquivo visitas.php

```
<?
    include("cookie.php");

    echo "Você já esteve aqui $numvisits vezes";
?>
```

Agora, vamos estudar o código do arquivo **cookie.php** separadamente, para você entendê-lo melhor.

Logo após as informações pessoais e o reporting, podemos ver que temos o código:

```
<?
    if ($Test != ‘ ‘) {
        $Test++;
        setcookie("Test",$Test, time()+3600000);
    }
?>
```

Essa parte será a responsável pela criação do cookie no computador cliente (do usuário). O comando else que vem logo após o o código acima, é usado para acionar o outro código caso o cookie já exista no cliente.

Seguindo abaixo temos a fase terminal do script que respondera caso o cookie já exista, retornando o número de vezes que o usuário já esteve no site.

```
<?
    // ...
    } else {
       setcookie("Test",1, time()+3600000);
       $Test = 1;
    }
    $numvisits = $Test;
?>
```

Esse código após verificar o cookie irá responder quantas vezes o cliente já esteve no site e ambas as partes completarão a função de gravar novamente os novos dados no cookie.

Vá para o seu browser, carregue a página **visitas.php** e observe que o browser lhe informará o número de visitas que você fez no site.

Outro exemplo de uso de cookies:

```
<?
    setcookie("cookie", "teste", time() + 3600*24*365);
?>
```

Esse cookie tem validade de um ano:

3600 segundos = 1 hora
3600 segundos vezes 24 horas
24 horas vezes 365 dias

Para excluir um cookie basta fazer a mesmo sistema que criar ele, mas usando um prazo de expiração negativa. Veja o exemplo a seguir:

```
<?
  setcookie("cookie", "teste", time() – 3600*24*365);
?>
```

Veja acima o cookie que tinha validade de um ano será excluído pq foi passado que o tempo dele agora é de um ano atrás, ou seja, já acabou.

13. Trabalhando com Sessões (Sessions)

Uma sessão (**session**) é um período de tempo durante o qual uma pessoa navega num site. Uma sessão é gravada no servidor, e o tempo de duração de uma sessão pode ser ate o usuário fechar a página. Quando o usuário entra no site criamos uma sessão e ela recebe um número identificador único, chamado de session_id, para uma pagina ter o acesso aos dados da sessão ela tem que ter esse número identificador.

As sessões geralmente são usadas pra criar lojas virtuais e sistemas de login, onde o usuário entrará com usuário e senha em um formulário. Buscará no banco de dados e se achar algum usuário gravará na sessão uma identificação que dirá que ele já foi logado. Isso somente durará até o fechamento do browser.

Session_start() – é o comando para se utilizar uma sessão. Este comando deve ser **utilizado no início do código e antes de qualquer saída html.**

Sintaxe:

```
Session_start();
```

Exemplo:

```
<?
    session_start();
    $_SESSION["teste"] = 1;
?>
```

No exemplo acima, foi criado uma sessão com o nome se teste e valor 1. E nas paginas seguintes você pode restaurar esse valor assim:

```
<?
    session_start();
    echo $_SESSION["teste"];
?>
```

13.1. Função Session_register()

Você pode usar **session_register()** para registrar suas variáveis, no caso, o nome da variável na sessão será a mesma do nome da variável gravado, exemplo:

Exemplo:

```
<?
    $linguagem = "PHP";
    session_register("linguagem");
?>
```

Nesse caso foi gravada a variável $linguagem na variável de sessão “linguagem”, eis a vantagem de se usar $_SESSION que é recomendado a partir da versão 4.1, já que você poderá definir o nome da variável na sessão.

Obs.: session_register trabalha com register_globals on, e como está em extinção o uso em on, é sempre recomendado então usar $_SESSION, aliás, as facilidades são muito maiores, já que você trabalhará com elas como uma variável normal.

14. Funções do PHP

Neste capítulo iremos listar algumas das funções utilizadas pelo PHP. Procuramos citar as funções que verificamos serem comumente usadas por desenvolvedores de PHP. Para maiores informações sobre as funções aqui citadas e sobre as demais funções, consulte o manual do PHP no site: **www.php.net**.

14.1. Funções diversas

Função chr( )

Esta função retorna o caracter correspondente ao código ASCII fornecido.

Sintaxe:

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

Exemplo:

```
<?
    $texto = 65;
    $ascii = chr($texto);
    echo "O caracter correspondente ao código ".$texto." da Tabela ASCII é: "."<b>$ascii</b>";
?>
```

Função ord( ) - Esta função retorna o código ASCII correspondente ao caracter fornecido.

Sintaxe:

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

Exemplo:

```
<?
    $texto = 65;
    $ord = ord($texto);
    echo "O código da tabela ASCII correspondente ao caracter ".$texto." é: "."<b>$ord</b>";
?>
```

Função strlen( ) - Esta função retorna o tamanho da string fornecida.

Sintaxe:

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

Exemplo:

```
<?
    $texto = “Curso de PHP";
    $retorno = strlen($texto);
    echo "Conteúdo de $texto: “.$texto<p>;
    echo “Quantidade de caracteres: "."<b>$retorno</b>";
?>
```

Função strtolower( ) - Esta função retorna a string fornecida com todas as letras minúsculas.

Sintaxe:

strtolower(string str);

Exemplo:

```
<?
    $texto = “CURSO DE  PHP"; // Digite o texto em letras maiúsculas.
    $retorno = strtolower($texto);
    echo "Conteúdo de $texto: “.$texto<p>;
    echo “Retorno da função: "."<b>$retorno</b>";
?>
```

Função strtoupper( ) - Esta função retorna a string fornecida com todas as letras maiúsculas.

Sintaxe:

string strtoupper(string str);

Exemplo:

```
<?
    $texto = “curso de  php"; // Digite o texto em letras minúsculas.
    $retorno = strtoupper($texto);
    echo "Conteúdo de $texto: “.$texto<p>;
    echo “Retorno da função: "."<b>$retorno</b>";
?>
```

Função ucfirst( ) - Esta função retorna a string fornecida com o primeiro caracter em letra maiúscula.

Sintaxe:

string ucfirst(string str);

Exemplo:

```
<?
    $texto = “curso de  php"; // Digite o texto em letras minúsculas.
    $retorno = ucfirst($texto);
    echo "Conteúdo de $texto: “.$texto<p>;
    echo “Retorno da função: "."<b>$retorno</b>";
?>
```

Função ucwords( ) - Esta função retorna a string fornecida com todas as palavras iniciadas por letras maiúsculas.

Sintaxe:

string ucwords(string str);

Exemplo:

```
<?
    $texto = “curso de  php"; // Digite o texto em letras minúsculas.
    $retorno = ucwords($texto);
    echo "Conteúdo de $texto: “.$texto<p>;
    echo “Retorno da função: "."<b>$retorno</b>";
?>
```

Função trim( ) - Esta função retira espaços e linhas em branco do início e do final da string fornecida.

Sintaxe:

string trim(string str);

Exemplo:

```
<?
    $texto = “   curso 01 ”;
    $retorno = trim($texto);
    echo “Retorno da função: "."<b>$retorno</b>";
?>
```

Função strstr( ) - Esta função 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.

Sintaxe:

string strstr(string str1, string str2);

Exemplo:

```
<?
    texto = “Primeiro Curso de PHP;
    $retorno = strstr($texto, “Curso”);
    echo "Conteúdo de $texto: “.$texto<p>;
    echo “Retorno da função: "."<b>$retorno</b>";
?>
```

Função isset( ) - Esta função é utilizada para verificar se uma variável possui algum valor setado. Retorna **true** se a variável estiver setada (ainda que com uma string vazia ou o valor zero), e **false** em caso contrário.

Sintaxe:

int isset(mixed var);

Exemplo:

```
<?
    $texto = “curso”;
    $isset = isset($texto);
    echo "valor do isset: ".$isset;
?>
```

Função empty( ) - Esta função verifica se uma variável está vazia ou se contem o valor 0 (zero). Caso esteja vazia, retorna o valor true(1), caso contrário, retorna o valor false(0).

Sintaxe:

int empty(mixed var);

Exemplo:

```
<?
    $texto = “0”;
    $empty = empty($texto);
    echo "valor do isset: ".$isset;
?>
```

Função date( ) - Esta função retorna a data ou hora atual do sistema.

Sintaxe:

date(“formato da data”);

Exemplo:

```
<?
    $data = date(“d/m/y”);
    echo a data é: $data”;
?>
```

A função date possui vários argumentos que serão utilizados de acordo com a forma como a data ou hora aparecerá para o usuário. Veja os exemplos no script a seguir.

Exemplo 2:

```
<?
    $dia = date('d');
    $mes = date('m');
    $ano2 = date('y');
    $ano4 = date('Y');
    $sem = date('w');
    $data = date('d/m/Y');
    $data1 = date('d/m/y');
    $hora = date('h:m:s');
    $h = date('h');
    $min = date("m");

    echo "
            Data atual exibindo o ano com 4 dígitos: <b>$data</b> e com 2 dígitos: <b>$data1.</b><p>
            \$dia = date('d')- exibe somente o dia da data atual do sistema. O dia é: <b>$dia</b><p>
            \$mes = date('m')- exibe somente o mês da data atual do sistema. O mês é: <b>$mes</b>$m<p>
            \$ano2 = date('y')- exibe somente o ano com 2 dígitos da data atual do sistema. O ano é: <b>$ano2</b><p>
            \$ano4 = date('Y')- exibe somente o ano com 4 dígitos da data atual do sistema. O ano é: <b>$ano4</b><p>
            \$sem = date('s')- exibe o número correspondente ao dia da semana, de 0(domingo) a 7(sábado). O número é: <b>$sem</b><p>
            \$hora = date('h:m:s') - exibe a hora atual do sistema. A hora é: <b>$hora</b><p>
            \$h = date('h') - exibe somente o valor correspondente a hora do sistema. O valor da hora é: <b>$h</b><p>
            \$min = date('m') - exibe somente o valor correspondente aos minuntos da hora do sistema. O valor dos minutos é: <b>$min</b><p>
    ";
?>
```

Função explode( ) - Esta função retorna um array contendo partes da string fornecida separadas pelo padrão fornecido, sem limitar o número de elementos do array.

Sintaxe:

array explode(string padrao, string str);

Exemplo:

```
<?
    $data = “11/12/1975”;
    $data_array = explode(“/”,$data);
    $novadata = $xdata_array[2]."-".$xdata[1]."-".$xdata[0];
    echo $novadata;
?>
```

O código acima faz com que a variável **$data_array** receba o valor: array(11,12,1975);

O **PHP** irá exibir a data no seguinte formato:

Ano-mes-dia, ficando:

1975-12-11, pis o MySQL armazena as datas nesse formato.

Função nl2br( ) - Esta função retorna a string fornecida substituindo todas as quebras de linha (“\n”) por quebras de linhas em html (“
”).

A função acima pode ser utilizada, por exemplo, em um campo de comentário em um formulário.

Sintaxe:

string nl2br(string str);

Exemplo:

```
<?
    $data = date("d/m/y");
    echo nl2br("Curso de PHP\n Data de início: $data\n Valor: R$ 700,00\n");
?>
```

No exemplo acima, a função **nl2br** irá retornar:

Curso de PHP
Data de início: a data atual do seu sistema
Valor: R$ 700,00

Função substr( ) – Esta função retorna uma parte de uma string.

Sintaxe:

string substr (string string, int start [, int length])

Se start não for negativo, a string retornada iniciará na posição **start** em string, começando em zero. Por exemplo, na string 'abcdef', o caractere na posição 0 é 'a', o caractere na posição 2 é 'c', e assim em diante.

Exemplos:

Exemplo 1) Uso básico de substr()

```
<?
```

```
    $rest = substr("abcdef", 1); // retorna "bcdef"
    $rest = substr("abcdef", 1, 3); // retorna "bcd"
    $rest = substr("abcdef", 0, 4); // retorna "abcd"
    $rest = substr("abcdef", 0, 8); // retorna "abcdef"

    // Outra opção é acessar através de chaves

    $string = 'abcdef';
    echo $string{0}; // retorna a
    echo $string{3}; // retorna d
?>
```

Se **start** for negativo, a string retornada irá começar no caractere start a partir do fim de string.

Exemplo 2) Usando um inicio negativo

```
<?
    $rest = substr("abcdef", -1); // retorna "f"
    $rest = substr("abcdef", -2); // retorna "ef"
    $rest = substr("abcdef", -3, 1); // retorna "d"
?>
```

Se **length** for dado e for positivo, a string retornada irá conter **length** caracteres começando em **start** (dependendo do tamanho de string). Se a string é menor do que **start**, será retornado **FALSE**.

Se **length** for dado e for negativo, então esta quantidade de caracteres será omitida do final de string (após a posição de inicio ter sido calculada quando **start** for negativo).

Se **start** denota uma posição além da truncagem, uma string vazia será retornada.

Exemplo 3) Usando um length negativo

```
<?
    $rest = substr("abcdef", 0, -1); // retorna "abcde"
    $rest = substr("abcdef", 2, -1); // retorna "cde"
    $rest = substr("abcdef", 4, -4); // retorna ""
    $rest = substr("abcdef", -3, -1); // retorna "de"
?>
```

14.2. Funções de manipulação de arquivos

Função File_exists( ) - Esta função checa se um arquivo ou diretório existe no servidor.

Sintaxe:

bool file_exists (string nomedoarquivo)

Esta função retorna **TRUE** se o arquivo ou diretório especificado por nomedoarquivo existir; **FALSE** caso contrário.

Obs.: No Windows, para checar arquivos em compartilhamentos de rede, use:

- //computername/share/filename; ou
- \\\\computername\share\filename

Exemplo:

```
<?
    $filename = '/caminho/para/qualquer.txt';

    if (file_exists($filename)) {
        print "O arquivo $filename existe";
    } else {
        print "O arquivo $filename não existe";
    }
?>
```

Obs.: O resultado desta função é cacheada. Veja clearstatcache() para mais detalhes.

Esta função não trabalha com arquivos remotos, de forma que o arquivo a ser examinado precisa ser acessível pelo sistema de arquivos do servidor.

Função filesize( ) - Esta função é utilizada para ler o tamanho do arquivo. Retorna o tamanho do arquivo, ou FALSE em caso de erro.

Sintaxe:

int filesize (string nomedoarquivo)

Obs1.: O resultado desta função é cacheado. Veja clearstatcache() para mais detalhes.

Esta função não trabalha com arquivos remotos, de forma que o arquivo a ser examinado precisa ser acessível pelo sistema de arquivos do servidor.

Exemplo:

```
<?
    // Exibe algo como: arquivo.txt: 1024 bytes
    $filename = 'arquivo.txt';
    echo $filename . ': ' . filesize($filename) . ' bytes';
?>
```

Função filetype( ) - Esta função é utilizada para ler o tipo de arquivo.

É muito utilizada quando se deseja fazer uploads de arquivos e você precisa verificar se o tipo de arquivo que está sendo enviado está de acordo com os tipos permitidos por você. Arquivos do tipo **.exe**, **.ini**, **.bat**, normalmente são arquivos executáveis e que podem conter vírus. Por isso, na maioria dos casos, não são permitidos que sejam enviados para o servidor.

Esta função retorna o tipo do arquivo (file type). Os valores possíveis são fifo, char, dir, block, link, file e unknown (desconhecido). Retorna **FALSE** se um erro ocorrer. Esta função filetype() também produzirá um erro E_NOTICE se a chamada a stat falhar ou se o tipo de arquivo for desconhecido.

Sintaxe:

string filetype (string nomedoarquivo)

Obs.: O resultado desta função é cacheado. Veja clearstatcache() para mais detalhes.

Esta função não trabalha com arquivos remotos, de forma que o arquivo a ser examinado precisa ser acessível pelo sistema de arquivos do servidor.

Exemplo:

```
<?
    echo filetype('/etc/passwd'); // file
    echo filetype('/etc/'); // dir
?>
```

Função: clearstatcache( ) - Esta função é utilizada para limpar as informações em cache de arquivos que foram verificados, por exemplo, pela função **filesize()**.

Sintaxe:

void clearstatcache (void)

Quando você chama stat(), lstat() ou qualquer uma das funções afetadas (listadas abaixo), o PHP mantém em cache as informações que essas funções retornam para melhoria de performance. Entretanto, em certos casos você pode precisar limpar as informações cacheadas. Por exemplo, se um mesmo arquivo é verificado várias vezes em um único script, e esse arquivo corre o risco de ser apagado ou modificado durante a operação do script, você precisa limpar os dados do cache. Nesses casos, você pode utilizar a função clearstatcache() para limpar todas as informações que o PHP mantém sobre um arquivo.

As funções afetadas são stat(), lstat(), file_exists(), is_writable(), is_readable(), is_executable(), is_file(), is_dir(), is_link(), filectime(), fileatime(), filemtime(), fileinode(), filegroup(), fileowner(), filesize(), filetype(), e fileperms().

Obs.: Esta função guarda informações sobre arquivos específicos, de forma que você somente precisa chamar clearstatcache() se você estiver realizando várias operações sobre o mesmo arquivo e necessita que a informação sobre esse arquivo em particular não seja cacheada.

Exemplo:

```
<?
    echo filetype('/etc/passwd'); // file
    echo filetype('/etc/'); // dir
    clearstatcache(); // neste momento esta função limpou as informações do cachê do arquivo.
?>
```