PHP

Para incluir a linguagem PHP em páginas HTML, é necessário iniciar o script incluindo '<?php' para iniciar os códigos php e '?>' para finalizá-los.

Comandos de saída (Comandos output)

Para imprimir dados na tela, utiliza-se "echo" ou "print", dispostos da seguinte forma:

```
<?php
echo 'Texto aqui';
?>

<?php
print 'Texto aqui';
?>
```

O comando "echo" imprime uma ou mais variáveis na tela ("echo 'a', 'b', 'c';" = abc), enquanto o comando "print" imprime um conjunto único de strings ("print 'abc';" = abc).

Extensão de arquivos

A forma mais comum de nomear programas em PHP é a seguinte:

Extensão	Significado
.php	Arquivo PHP contendo um programa.
.class.php	Arquivo PHP contendo uma classe.
.inc.php	Arquivo PHp a ser incluído, pode incluir constantes ou configurações.

Tipo de Dados

O PHP suporta oito tipos primitivos. São quatros tipos básicos: boolean, integer, float (número de ponto flutuante, ou também 'double') e string . Dois são tipos compostos: array, object. E, finalmente, dois tipos especiais: resource, NULL.

O PHP utiliza uma checagem dinâmica de tipos, isto é, uma variável pode conter valores de diferentes tipos em diferentes momentos da execução do script. Por esse motivo, não é preciso declarar o tipo de uma variável para implementá-la. O interpretador PHP decidirá qual é o tipo daquela variável, verificando o conteúdo em tempo de execução. Ainda assim, é permitido converter os valores de um tipo para outro, utilizando o typecasting ou a função settype.

Tipo Booleano

Também chamado de BOOLEAN. Este é o tipo mais fácil. Um booleano expressa um valor de verdade. Ele pode ser TRUE ou FALSE. Para especificar um literal booleano, use as palavras chave TRUE ou FALSE. Ambas são case insensitive.

```
<?php
$variavel=True;// assimila o valor TRUE para $variavel
?>
```

Vejamos agora exemplos sobre como fazer comparações com o tipo booleano:

```
<?php
if ($variavel){
      echo 'A variável é verdadeira.';
}else{
      echo 'A variável é falsa.';
}
?>

<?php
if ($variavel==TRUE){
      echo 'A variável é verdadeira.';
}
if (!$variavel){
      echo 'A variável é falsa.';
}
?>
```

No PHP pode-se fazer a conversão de tipos. Para converter explicitamente um valor para booleano, utilize-se dos modificadores (bool) ou (boolean). Entretanto, na maioria dos casos, você não precisa utilizar o modificador, desde que qualquer valor será convertido automaticamente se um operador, função ou estrutura de controle requerer um argumento booleano. Quando convertendo para booleano, os seguintes valores são considerados FALSE:

- o próprio booleano FALSE;
- o inteiro 0 (zero);
- o ponto flutuante 0.0 (zero);
- uma string vazia e a string "0";
- um array sem elementos;
- um objeto sem elementos membros;
- o tipo especial NULL (incluindo variáveis não definidas).

Qualquer outro valor é considerado TRUE (incluindo qualquer recurso). Exemplo de conversão:

```
<?php
$a="”; // uma string vazia
$b=(bool) $a; // b agora é booleano e igual a FALSE
$a=1;
$b=(bool) $a; // TRUE
$c=(bool) “string”; // TRUE
?>
```

Tipo Inteiro

Também chamado de INTEGER. Um inteiro é um número do conjunto Z = {..., -2, -1, 0, 1, 2, ...}. Inteiros podem ser especificados em notação decimal (base 10), hexadecimal (base 16) ou octal (base 8), opcionalmente precedido de sinal (- ou +).Para usar a notação octal, você precisa preceder o número com um 0 (zero). Para utilizar a notação hexadecimal, preceda número com 0x. Exemplo de definições:

```
<?php
$a=1234; # número decimal
$a=-123; # número negativo
$a=0123; # número octal (equivalente a 83 em decimal)
$a=0x1A; # número hexadecimal (equivalente a 26 em decimal)
?>
```

Assim como vimos no tipo Booleano, o PHP também permite a conversão de outros tipos para Inteiro. Para converter explicitamente um valor para inteiro, utilize-se dos modificadores (int) ou (integer). Entretanto, na maioria dos casos, você não precisa utilizar o modificador, desde que qualquer valor será automaticamente convertido se um operador, função ou estrutura de controle requerer um argumento inteiro. Você também pode converter o valor de um inteiro com a função intval(). Exemplo de conversão:

```
<?php
$a=TRUE;// booleano
$b=(int) $a;// retorna 1

$a=FALSE;// booleano
$b=(int) $a;// retorna 0

$a=2.2232;// ponto flutuante
$b=(int) $a;// retorna 2 (convertido para inteiro)
?>
```

Tipo Ponto Flutuante

Também chamado de FLOAT ou DOUBLE. Números de ponto flutuante podem ser especificados utilizando qualquer uma das sintaxes seguintes:

```
<?php
$a=1.234;
$a=1.2e3;
$a=7E-10;
?>
```

Tipo String

String é um conjunto de caracteres e pode ser definida de três formas diferentes: com apóstrofo, aspas ou sintaxe heredoc. A maneira mais simples para especificar uma string é delimitá-la entre apóstrofos (o caracter ').

Para especificar um apóstrofo dentro da string, você precisará "escapá-la" com uma contra barra (\), como em muitas outras linguagens. Se uma contra barra precisa ocorrer antes de um apóstrofo ou no final da string, você precisa duplicá-la. Note que se você tentar escapar qualquer outro caractere, a contra barra também será impressa! Então geralmente não é necessário escapar a própria contra barra. Vejamos exemplos abaixo:

```
<?php
echo 'isto é uma string comum';
echo 'Você pode incluir novas linhas em strings, dessa maneira estará
tudo bem.';
echo 'Arnold once said: "I'll be back".';
// Imprime: Arnold once said: "I'll be back".
echo 'Você tem certeza em apagar C:\\*.*?';
// Imprime: Você tem certeza em apagar C:\\*.*?
echo 'Isto não será substituído: \n uma nova linha.';
// Imprime: Isto não será substituído: uma nova linha.
echo 'Variáveis $tambem não $expandem';
// Imprime: Variáveis $tambem não $expandem
?>
```

Em caso de dúvida, consulte a tabela seguinte:

Nova linha	\n
Retorno de carro (semelhante a \n)	\r
Tabulação horizontal	\t
A própria barra (\)	\\
O símbolo $	\$
Aspas simples	\'
Aspas duplas	\"

Tipo Array

Array, ou Vetor, é atualmente um mapa ordenado. Um mapa é um tipo que relaciona valores por chaves. Um array pode ser criado com o construtor de linguagem array(). Ele pega certo número de pares separados por vírgula chave => valor. Exemplo:

```
<?php
$arr=array(”chave1”=>”valor1”,”chave2”=>”valor2”,3=>”valor3”);
echo $arr[”chave1”];// valor1
echo $arr[”chave2”];// valor2
echo $arr[”chave3”];// valor3
?>
```

Observe que a chave pode ser tanto uma string (“chave”) quanto um inteiro (3). Já o valor pode conter qualquer tipo suportado pelo PHP. Um valor pode ainda, conter outro array, como segue no exemplo:

```
<?php
$arr=array(”chave”=>array(6=>5,13=>9,”a”=>42));
echo $arr[”chave”][6];// 5
echo $arr[”chave”][13];// 9
echo $arr[”chave”][ ”a”];// 42
?>
```

Além de definir arrays com o comando array(), você ainda pode definir diretamente com atribuição. Veja o exemplo:

```
<?php
$arr[0]=‘Valor x’;
$arr[1]=‘Valor y’;
echo $arr[0];// Valor x
echo $arr[1];// Valor y
?>
```

Como vimos, as chaves podem conter inteiros. Você pode ainda inserir novos valores num array, apenas usando os colchetes. Continuando o exemplo anterior:

```
<?php
$arr[0]=‘João’;// chave 0
$arr[1]=‘Maria’;// chave 1
$arr[]=‘José’;// chave 2
$arr[]=‘Paulo’;// chave 3
echo $arr[0];// João
echo $arr[1];// Maria
echo $arr[2];// José
echo $arr[3];// Paulo
?>
```

Para apagar uma chave do vetor, você pode utilizar a função unset(). Como no exemplo a seguir:

```
<?php
$arr[0]=‘João’;// chave 0
$arr[1]=‘Maria’;// chave 1
unset ($arr[0]);
echo $arr[0]; // não existe a chave 0, portanto, não imprime
?>
```

Mais informações sobre Arrays, você irá visualizar na seção dedicada: Trabalhando com Arrays.

Tipo Objetos

Objetos são instâncias de classes definidas pelo usuário. Para inicializar um objeto, você usa a instrução “new”, criando uma instância do objeto em uma variável. Exemplo:

```
<?php
class foo{
      function do_foo(){
            echo “Fazendo foo.”;
      }
}
$bar=new foo;
$bar->do_foo();
?>
```

Tipo NULL

O valor especial NULL representa que a variável não tem valor. NULL é o único valor possível do tipo NULL (case insensitive). Exemplo:

```
<?php
$var=NULL;
?>
```

Para verificar se uma variável é NULL, utilize a função is_null(), passando como parâmetro a variável. Exemplo:

```
<?php
$var=NULL;
if(is_null($var)){
      echo “A variável foi setada como NULL”;
}
?>
```

Variáveis

As variáveis no PHP são representadas por um cifrão ($) seguido pelo nome da variável. Os nomes de variável no PHP fazem distinção entre maiúsculas e minúsculas (sensitive case). Os nomes de variável seguem as mesmas regras como outros rótulos no PHP. Um nome de variável válido se inicia com uma letra ou sublinhado, seguido de qualquer número de letras, algarismos ou sublinhados. Não é válido, portanto, variáveis que iniciem com números. Exemplo:

```
$var=“Grupo”;
$Var=”E-jovem”;
echo “$var, $Var”; // exibe “Grupo, E-jovem”
$2var=“teste”; // inválido; começa com um número
$_2var=“teste”; // válido; começa com um sublinhado
```

Constantes

Uma constante é um identificador (nome) para um único valor. Como o nome sugere, esse valor não pode mudar durante a execução do script. As constantes são case sensitive por padrão. Por convenção, os nomes de constantes são sempre em maiúsculas. O nome de uma constante tem as mesmas regras de qualquer identificador no PHP.

Um nome de constante válida começa com uma letra ou sublinhado, seguido por qualquer número de letras, números ou sublinhados. Você pode definir uma constante utilizando-se da função define(). Quando uma constante é definida, ela não pode ser mais modificada ou anulada.

Estas são as diferenças entre constantes e variáveis:

- Constantes não podem ter um sinal de cifrão ($) antes delas;
- Constantes só podem ser definidas utilizando a função define(), e não por simples assimilação;
- Constantes podem ser definidas e acessadas de qualquer lugar sem que as regras de escopo de variáveis sejam aplicadas;
- Constantes não podem ser redefinidas ou eliminadas depois que elas são criadas;
- Constantes só podem conter valores escalares.

Os tipos de dados aceitos numa constante são: Booleano, Inteiro, Float e String. Exemplo de definição de constante:

```
<?php
define(“CONSTANT”, “Hello, world.”);
echo CONSTANT;// imprime “Hello, world.”
echo Constant;// imprime “Constant” e gera um alerta notice.
?>
```

Operadores

Um operador é algo que você alimenta com um ou mais valores (ou expressões, no jargão de programação) e que devolve outro valor (e por isso os próprios construtores se tornam expressões). Assim, você pode pensar que as funções e os construtores que retornam valores (como o print) são operadores e os outros que não retornam nada (como echo) como outra coisa.

Há três tipos de operadores:

- os operadores unários, que operam em apenas um valor. Por exemplo, “!” (operador de negação) ou o “++” (operador de incremento).

- os operadores binários, o grupo que contém a maioria dos operadores que o PHP suporta.
- os operadores ternários: “?:”. Eles podem ser usados para selecionar entre dois valores dependendo de um terceiro, em vez de selecionar duas sentenças ou encadeamentos de execução.

  Englobar expressões ternárias com parênteses é uma boa idéia.

Operadores de atribuição

O operador básico de atribuição é “=”. A sua primeira vista deve ser a de definir isto como “é igual”. Não. Isto quer dizer, na verdade, que o operando da esquerda recebe o valor da expressão da direita, ou seja, “é configurado para”. O valor de uma expressão de atribuição é o valor atribuído. Ou seja, o valor de “$a = 3” é 3. Isto permite que você faça alguns truques. No exemplo abaixo, $a é igual a 9 agora e $b foi configurado como 4:

```
<?php
$a=($b=4)+5;
?>
```

Há, ainda, o que chamamos de operadores combinados. Significa combinar operadores aritméticos e de concatenação com o operador de atribuição, da seguinte forma: “.=” para concatenar, “+=” para somar, “-=” para subtrair, “*=” para multiplicar e “/=” para dividir.

Exemplo: vamos verificar como concatenar duas strings, sem fazer uso dos operadores combinados, para exemplificar um concatenamento.

```
<?php
$nome='Jorge';
$cidade='Maranguape';
echo "Meu nome é ".$nome."e eu moro em ".$cidade.".";
// imprime "Meu nome é Jorge e eu moro em Maranguape." na tela
?>
```

No caso acima, o ponto significa concatenação, e da mesma forma podem ser utilizados os outros operadores aritméticos. Agora um exemplo de concatenamento com a mesma variável (lembrando que pode ser aplicada à soma, subtração, etc):

```
<?php
$a='Jorge';
$a.=' Versilo';
echo $a;// imprime "Jorge Versilo." na tela
?>
```

Operadores aritméticos

Lembra-se da aritmética básica da escola? Estes operadores funcionam exatamente como aqueles.

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

O operador de divisão ("/") sempre retorna um valor com ponto flutuante (fracionário), mesmo que os dois operandos sejam inteiros (ou strings convertidos para inteiros).

Operadores de comparação

Operadores de comparação, como os seus nomes implicam, permitem que você compare dois valores. A tabela abaixo lista os operadores de comparação existentes no PHP:

Exemplo	Nome	Resultado
$a==$b	Igual	Verdadeiro (true) se $a é igual a $b.
$a!=$b	Diferente	Verdadeiro se $a não é igual a $b.
$a<>$b	Diferente	Verdadeiro se $a não é igual a $b (para números).
$a<$b	Menor que	Verdadeiro se $a é estritamente menor que $b.
$a>$b	Maior que	Verdadeiro se $a é estritamente maior que $b.
$a<=$b	Menor ou igual	Verdadeiro se $a é menor ou igual a $b.
$a>=$b	Maior ou igual	Verdadeiro se $a é maior ou igual a $b.

Outro operador condicional é o operador “?:” (ou ternário), que opera como no C e em muitas outras linguagens. Vejamos um exemplo do operador “?:”:

```
<?php
$a=true;
$b=($b)?‘Verdadeiro’:‘Falso’;
echo $b;// imprime “Verdadeiro” na tela
?>
```

Explicando: o operador “?:” funciona como um IF e ELSE, mas numa mesma linha, para operações simples condicionais. Neste caso, se a variável $a for verdadeira (TRUE) então a variável $b assume a string “Verdadeiro”, caso contrário, a variável $b assume a string “Falso”. Da mesma forma, poderíamos fazer:

```
<?php
$a=true;
if($a){
	$b=“Verdadeiro”;
}else{
	$b=“Falso”;
}
echo $b;
?>
```

Operadores lógicos

Os operadores lógicos, assim como os de comparação, examina 2 variáveis e retorna um resultado binário Falso (FALSE) ou Verdadeiro (TRUE). A tabela abaixo lista os tipos de operadores lógicos:

Exemplo	Nome	Resultado
$a and $b	E Verdadeiro (true)	Se tanto $a quanto $b são verdadeiros.
$a or $b	OU Verdadeiro	Se $a ou $b são verdadeiros.
$a xor $b	XOR Verdadeiro	Se $a ou $b são verdadeiros, mas não ambos.
!$a	NÃO Verdadeiro	Se $a não é verdadeiro.
$a&&$b	E Verdadeiro	Se tanto $a quanto $b são verdadeiros.
$a\|\|$b	OU Verdadeiro	Se $a ou $b são verdadeiro.

A razão para as duas variantes dos operandos “and” e “or” é que eles operam com precedências diferentes.

Operadores de strings

Há dois operadores de string. O primeiro é o operador de concatenação “.”, que retorna a concatenação dos seus argumentos direito e esquerdo. O segundo é o operador de atribuição de concatenação “.=” (previamente explicado em Operadores de atribuição), que acrescenta o argumento do lado direito no argumento do lado esquerdo. Exemplo:

```
<?php
$a='Olá';
$b=$a.' mundo!';// agora $b contém "Olá mundo!"

$a='Olá';
$a.=' mundo!';// agora $a contém "Olá mundo!"
?>
```

Precedência de Operadores

A precedência de um operador especifica quem tem mais prioridade quando há duas delas juntas. Por exemplo, na expressão, 1 + 5 * 3, a resposta é 16 e não 18 porque o operador de multiplicação (“*”) tem mais prioridade de precedência que o operador de adição (“+”). Parênteses podem ser utilizados para forçar a precedência, se necessário. Assim, (1 + 5) * 3 é avaliado como 18.

Expressões

Expressões são as peças de construção mais importantes do PHP. No PHP, quase tudo o que você escreve são expressões. A maneira mais simples e ainda mais precisa de definir uma expressão é “tudo o que tem um valor”.

As formas mais básicas de expressões são constantes e variáveis. Quando você digita “$a = 5”, você está atribuindo ‘5’ para $a. ‘5’, obviamente, tem o valor 5 (neste caso, ‘5’ é uma

constante inteira). Depois desta atribuição, você pode esperar que o valor de $a seja 5. Assim se você escrever $b = $a, você pode esperar que $b se comporte da mesma forma que se você escrevesse $b = 5.

Há mais uma expressão que pode parecer estranha se você não a viu em outras linguagens, o operador condicional ternário:

```
<?php
$primeira?$segunda:$terceira;
?>
```

Explicando: se o valor da primeira sub-expressão é verdadeiro (TRUE, não-zero), então a segunda sub-expressão é avaliada, e este é o resultado da expressão condicional. Caso contrário, a terceira sub-expressão é avaliada e este é o valor. Poderia ser escrita também assim:

```
<?php
if($primeira){
	$segunda;
}else{
	$terceira;
}
?>
```

Outro tipo comum de expressão são as expressões de comparação. O PHP suporta:

Símbolo	Significado
>	Maior que
>=	Maior ou igual
==	Igual
!=	Diferente
<	Menor que
<=	Menor ou igual
===	Igual e do mesmo tipo
!==	Diferente ou não do mesmo tipo

Estas expressões são usadas mais frequentemente dentro de instruções condicionais, como em comandos if. Outro bom exemplo de orientação de expressão é o pré e o pós-incremento e decremento. Se você quer somar 1 à variável $a, pode fazer de 2 formas:

```
<?php
$a=$a+1;// soma 1 à variável $a
$a++;// pós-incremento = também soma 1 à $a
++$a;// pré-incremento = também soma 1 à $a
?>
```

Da mesma forma você pode usar para subtração. A diferença entre pré e pós-incremento, é de que no pré-incremento o valor é calculado antes e retornado depois. No pós-incremento, o valor é retornado, e só depois calculado.

Ainda em se tratando de expressões, outro caso importante é o de combinar operador de atribuição. Você já sabe que se você quer incrementar $a de 1, você só precisa escrever $a++ ou ++$a. Mas e se você quiser somar mais que um a ele, por exemplo 3? Você poderia escrever $a++ várias vezes, mas esta obviamente não é uma forma muito eficiente ou confortável.

Uma prática muito mais comum é escrever $a = $a + 3. $a + 3 é avaliada como o valor de $a mais 3, e é atribuído de volta a $a, que resulta em incrementar $a de 3. Em PHP, como em várias outras linguagens como o C, você pode escrever isto de uma forma mais curta, que com o tempo se torna mais limpa e rápida de se entender também. Somar 3 ao valor corrente de $a pode ser escrito $a += 3. Qualquer operador de dois parâmetros pode ser usado neste modo operador de atribuição, por exemplo, $a -= 5 (subtrai 5 do valor de $a), $b *= 7 (multiplica o valor de $b por 7), etc. O exemplo a seguir mostra os tipos de expressões explicados nesta parte:

```
<?php
function double($i){
return $i*2;
} /* atribui o valor 5 à $a e $b */
$b=$a=5;
/* pós-incremento, atribui o valor original de $a (5) para $c, e
depois incrementa $a (6) */
$c=$a++;
/* pós-incremento, atribui o valor incrementado de $b (6) para $d e $e
*/
$e=$d=++$b;
/* neste ponto, tanto $d quanto $e são iguais a 6 */
/* atribui o dobro do valor de $d antes do incremento, 2*6 = 12 a $f
*/
$f=double($d++);
/* atribui o dobro do valor de $e depois do incremento, 2*7 = 14 a $g
*/
$g=double(++$e);
/* primeiro, $g é incrementado de 10 e termina com o valor de 24. O
valor da atribuição (24) é então atribuído à $h, e $h termina com o
valor 24 também. */
$h=$g+=10;
?>
```


Trabalhando com Arrays

Já vimos que um Array, ou Vetor, é um mapa ordenado que relaciona valores por chaves. Neste capítulo vamos aprofundar mais os estudos neste tipo de dados, que é largamente utilizado em muitas operações no PHP.

O que é um array?

Uma variável escalar é uma localização identificada com um nome em que é armazenado um valor; de maneira semelhante, um array é um lugar identificado com um nome para armazenar um conjunto de valores, permitindo que você agrupe valores escalares.

Uma cesta de frutas será o array para nosso exemplo. Imagine uma lista de três frutas armazenadas em um formato de array e uma variável, chamada $cesta, que armazena os três valores. Os valores armazenados em um array são chamados elementos do array. Cada elemento do array tem um índice associado (também denominado chave) que é utilizado para acessar o elemento. Os arrays, na maioria das linguagens de programação, têm índices numéricos que geralmente iniciam em zero ou um. O PHP permite o uso de números e strings como índices de array.

É possível usar arrays da maneira tradicional, indexada numericamente, ou agrupar as chaves para que a indexação seja mais significativa e útil. A abordagem de programação pode variar um pouco dependendo se você está usando arrays indexados numericamente de forma padrão ou valores de índice mais interessantes.

Iniciaremos examinando arrays numericamente indexados e depois veremos as chaves definidas pelo usuário.

Arrays numericamente indexados

Esses arrays são suportados na maioria das linguagens de programação. Em PHP, os índices iniciam no zero por padrão, embora você possa alterar isso. Para criar o array mostrado na figura logo acima, em nossa cesta de frutas, utilize a seguinte linha de código de PHP:

```
<?php
$cesta=array("Pêra","Uva","Maçã");
?>
```

Isso criará um array chamado $cesta contendo os três valores dados – “Pêra”, “Uva” e “Maçã”. Note que, como echo, array( ) é na realidade uma construção da linguagem em vez de uma função. Dependendo do conteúdo que você precise no array, talvez não seja necessário inicializá-lo manualmente como no exemplo anterior. Se tiver os dados de que precisa em outro array, você simplesmente pode copiar um array para outro utilizando o operador de atribuição.

Acessando o conteúdo de array

Para acessar o conteúdo de uma variável, utilize o nome dela. Se a variável for um array, acesse o conteúdo utilizando o nome da variável e uma chave ou índice. A chave ou índice indica quais valores armazenados acessamos. O índice é colocado entre colchetes, depois do nome. Digite $cesta[0], $cesta[1], $cesta[2] para utilizar o conteúdo do array $cesta. Por padrão, o elemento zero é o primeiro elemento do array.

Como com outras variáveis, o conteúdo dos elementos do array é alterado utilizando o operador de atribuição. A próxima linha substituirá o primeiro elemento no array “Pêra” por “Banana”.

```
$cesta[0]=“Banana”;
```

A linha a seguir poderia ser utilizada para adicionar um novo elemento – “Banana” – ao final do array, fornecendo um total de quatro elementos:

```
$cesta[3]=“Banana”;
```

Para exibir o conteúdo, poderíamos digitar:

```
echo “$cesta[0], $cesta[1], $cesta[2], $cesta[3]”;
```

Observe que, embora a análise sintática da string de PHP seja relativamente inteligente, você pode confundi-la. Se você estiver tendo problemas com arrays ou outras variáveis que não são interpretadas corretamente quando são inseridas dentro de uma string entre aspas duplas, pode colocá-las fora das aspas.

A instrução echo anterior funcionará corretamente, mas, em muitos dos exemplos mais complexos, você notará que as variáveis estão fora das strings entre aspas. Como com outras variáveis de PHP, os arrays não precisam ser inicializados ou criados antecipadamente. Eles são criados de forma automática na primeira vez em que você os utiliza. O código a seguir criará o mesmo array $cesta criado anteriormente com a instrução array( ):

```
$cesta[0]=“Pêra”;
$cesta[1]=“Uva”;
$cesta[2]=“Maçã”;
```

Se $cesta não existir ainda, a primeira linha criará um novo array com apenas um elemento. As linhas subseqüentes adicionam valores ao array. O array é redimensionado dinamicamente à medida que você adiciona elementos a ele. Essa funcionalidade de redimensionamento não está presente na maioria das outras linguagens de programação.

Utilizando loops para acessar o array

Como o array está indexado por uma seqüência de números, podemos utilizar um loop for mais facilmente para exibir o conteúdo:

```
for($i=0;$i<3;$i++)
echo “$cesta[$i]”;
```

Esse loop fornecerá saída semelhante ao código anterior, mas exigirá menos digitação do que escrever o código manualmente para trabalhar com cada elemento em um array grande. A capacidade de utilizar um loop simples para acessar cada elemento é um excelente

recurso de arrays. Também podemos usar o loop foreach, visto anteriormente, especialmente projetado para usar com os arrays, poderíamos utilizá-lo como a seguir:

```
foreach($cesta as $cesta=>$valor)
echo "$chave : $valor";
```

Por sua vez, esse código armazena cada índice do array em $chave e cada valor em $valor e os imprime.

Arrays Associativos

No array $cesta, permitimos que o PHP ofereça um índice padrão a cada item. Isso significa que o primeiro item que adicionamos tornou-se o item 0, o segundo item, 1 e assim por diante. O PHP também suporta arrays em que podemos associar qualquer chave ou índice que quisermos a cada valor.

Inicializando um array

O código a seguir cria um array associativo com nome de frutas como chaves, e preços como valores:

```
preços=array("Pêra"=>7,"Uva"=>3,"Maçã"=>1);
```

O símbolo entre as chaves e os valores é simplesmente um sinal de igual seguido imediatamente de um sinal de maior que.

Acessando os elementos do array

Novamente, acessamos o conteúdo utilizando o nome da variável e uma chave, então podemos acessar as informações que armazenamos no array de preços como $precos["Pêra"], $precos["Uva"] e $precos["Maçã"]. O código a seguir criará o mesmo array $precos. Em vez de criar um array com três elementos, essa versão cria um array com somente um elemento e então acrescenta outros dois.

```
$preços=array("Pêra"=>7);
$preços=["Uva"]=>3;
$preços=["Maçã"]=>1;
```

Eis outra parte, ligeiramente diferente, mas equivalente de código. Nesta versão, não criamos explicitamente um array. O array é automaticamente criado quando adicionamos a ele o primeiro elemento.

```
$preços=("Pêra"=>7);
$preços=["Uva"]=>3;
$preços=["Maçã"]=>1;
```

Arrays multidimensionais

Os arrays não têm de ser uma lista simples de chaves e valores – cada localização no array pode armazenar outro array. Dessa maneira, podemos criar um array bidimensional. Você pode pensar em um array de duas dimensões como sendo uma matriz ou grade, com largura e altura ou linhas e colunas. Se quiséssemos armazenar mais de uma parte de dados sobre cada item da cesta, poderíamos utilizar um array bidimensional.

A figura abaixo mostra os itens da cesta representados como um array bidimensional com cada linha representando um produto individual e cada coluna representando um atributo do produto armazenado.

Utilizando PHP, escreveríamos o código a seguir para configurar os dados no array mostrado na figura acima.

```
$cesta=array(array("PER","Pêra",7),
             array("UVA","Uva" ,3),
             array("MAC","Maçã",7));
```

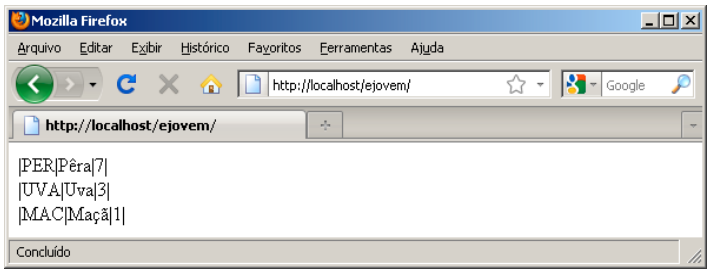
Você pode ver a partir dessa definição que nosso array de produtos agora contém três arrays. Para acessar os dados em um array dimensional, lembre-se de que precisamos do nome dos arrays e do índice dos elementos.

Um array bidimensional é semelhante, exceto que cada elemento tem dois índices – uma linha e uma coluna. (A linha superior é a linha 0 e a coluna na extremidade esquerda é a coluna 0.) Para exibir o conteúdo desse array, poderíamos acessar manualmente cada elemento no pedido deste modo:

```
echo '|'.$cesta[0][0].'|'.$cesta[0][1].'|'.$cesta[0][2].'|<br/>';
echo '|'.$cesta[1][0].'|'.$cesta[1][1].'|'.$cesta[1][2].'|<br/>';
echo '|'.$cesta[2][0].'|'.$cesta[2][1].'|'.$cesta[2][2].'|<br/>';

for($linha=0;$linha>3;$linha++){
      for($coluna=0;$coluna>3;$coluna++){
            echo '|'.$cesta[$linha][$coluna];
      }
      echo '|<br/>';
}
```

Alternativamente, poderíamos colocar um loop for dentro de outro para alcançar o mesmo resultado. As duas versões desse código produzem a mesma saída no navegador:



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.

IF

O IF é usado como estrutura condicional. Indica que se uma expressão for verdade, então faça tal ação.Sua sintaxe é simples. Se a ação a ser executada for de uma única linha, as chaves são opcionais, caso contrário, se a ação tiver vários comandos a serem executados, as chaves são obrigatórias.Veja um exemplo:

```
<?php
$a=3;
$b=1;
if($a>$b)
	echo "$a é maior do que $b"; // único comando
if($a>$b){
	// vários comandos, uso obrigatório das chaves
	$b=$a;
	$a++;
}
?>
```

ELSE

O ELSE é usado para que uma ação seja executada caso a expressão analisada no IF seja falsa. Em outras palavras, se uma expressão for verdade, faça a ação do IF, senão faça a ação do ELSE. Veja um exemplo:

```
<?php
$a=1;
$b=5;
if($a>$b){
	echo ”$a é maior do que $b”;
}else{
	echo ”$b é maior do que $a”;
}
?>
```

ELSEIF

A estrutura ELSEIF faz com que o usuário possa definir várias situações para que uma ação ocorra. Suponha que você quer executar uma ação para cada valor da uma variável. Por exemplo: se $var for igual a x, execute tal ação, se for igual a y, outra ação, se for igual a z, outra ação, e assim por diante. Exemplo:

```
<?php
$var='y';
if($var=='x'){
	echo ”Variável é igual a X.”;
}elseif($var == 'y'){
	echo “Variável é igual a Y.”;
}elseif($var == 'z'){
	echo “Variável é igual a Z.”;
}else{
	echo “Variável tem outro valor.”;
}
?>
```

SWITCH

A instrução switch é similar a uma série de instruções IF's seguidas. Observe o exemplo a seguir:

```
<?php
switch($i){
	case 0:
		echo “$i é igual a 0.”;
		break;
	case 1:
		echo “$i é igual a 1.”;
		break;
	case 2:
		echo “$i é igual a 2.”;
		break;
}
?>
```

É importante entender como a instrução switch funciona para evitar enganos. A instrução switch executa linha a linha. No início, nenhum código é executado. Quando uma

instrução case é encontrada com um valor que combina com a expressão do switch, o PHP executa as instruções a partir daí. O PHP continua executando as instruções até o fim do bloco switch ou finaliza o switch na primeira vez que encontrar uma instrução break.

Se você não escrever uma instrução break no fim das instruções case, o PHP continuará executando os cases seguintes. Um case especial é o default. Esse case é executado quando nenhum outro case combina. Ele precisa ser a última instrução case. Por exemplo:

```
<?php
switch($i){
	case 0:
		echo "$i é igual a 0.";
		break;
	case 1:
		echo "$i é igual a 1.";
		break;
	case 2:
		echo "$i é igual a 2.";
		break;
	default;
		echo "Não detectamos o valor de $i.";
}
?>
```

BREAK

BREAK cancela a execução do comando for, foreach, while, do...while ou switch atual. Break aceita um argumento numérico opcional que diz a ele quantas estruturas aninhadas englobadas devem ser quebradas.

CONTINUE

CONTINUE é usado dentro de estruturas de loops para saltar o resto da iteração do loop atual e continuar a execução no início da próxima iteração. Continue aceita um argumento numérico opcional que diz a ele de quantos níveis de loops aninhados ele deve saltar até o fim.

```
<?php
$i=0; while($i++ < 5){
		echo "Fora<br>\n";
		while(1){
			echo "  Meio<br>\n";
			while (1){
			echo "  Dentro<br>\n";
			continue 3;
			}
			echo "Isto nunca será exibido.<br>\n";
		}
		echo "Nem isso<br>\n";
	}
?>
```

FOR

A estrutura de laço (loop) FOR realiza repetidas ações a depender das expressões passadas como parâmetro. No total são 3 expressões: A primeira expressão (expr1) é avaliada (executada) uma vez no começo do loop. No começo de cada iteração, expr2 é avaliada. Se ela é avaliada como TRUE, o loop continua e o(s) comando(s) aninhado(s) é(são) executado(s). Se for avaliada como FALSE, a execução do 'loop' termina. No fim de cada iteração, expr3 é avaliada (executada). Veja um exemplo que imprime números de 1 a 10 na tela:

```
<?php
for($i=1;$i<=10;$i++){
	echo $i;
}
?>
```

Explicando: a variável $i é iniciada com valor 1, então é feita a verificação ($i <= 10): se for verdade, entra no loop, caso contrário, termina, e em cada iteração a variável $i é incrementada em 1 ($i++).

FOREACH

O construtor foreach permite interagir com arrays de forma mais simples, em força de laço, percorrendo todo o vetor. É útil caso você queira, por exemplo, usar os valores de todo o vetor para realizar alguma operação, ou imprimir na tela. Tem-se 2 tipos de sintaxe. A primeira não leva em consideração a chave do array, e a segunda utiliza as chaves. Sintaxe:

```
<?php
foreach(expressao_array as $valor)// instruções
foreach(expressao_array as $chave=>$valor)// instruções
?>
```

Vejamos um exemplo:

```
<?php
// exemplo usando apenas valores
$a=array(1, 2, 3, 17);
foreach($a as $v){
	echo "Valor atual: $v.<br/>";
}
// exemplo usando chaves e valores
$a=array("Um"=>1,"Dois"=>2,"Três"=>3,"Dezessete"=>17);
foreach($a as $k => $v){
	echo "$k=>$v.<br/>";
}
?>
```

WHILE

While é outra estrutura de laço (loop) mais simples. Enquanto a expressão (passada como parâmetro) for verdade, será executada ações dentro do bloco de chaves.

```
while{expressão}{
	//ações
}
```

Vejamos o mesmo exemplo de impressão de números de 1 a 10, mas agora utilizando a estrutura While:

```
<?php
$i=1;
while($i<=10){
	echo $i;
	$i++;// incrementa o $i para a próxima iteração
}
?>
```

Observe que o incremento é feito manualmente dentro do laço. Se você esquecer de incrementar, neste exemplo, o script ficará em loop infinito e será abortado pelo PHP.

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. Isso significa que a ação é feita no mínimo 1 vez. O laço do...while possui apenas uma sintaxe, que é a seguinte:

```
<?php
$i=0;
do{
	echo $i;
	$i++;
}
while($i<10);// incrementa o $i para a próxima iteração
?>
```

Funções

Uma função é um pedaço de código com um objetivo específico,encapsulado sob uma estrutura única que recebe um conjunto de parâmetros e retorna um dado. Uma função é declarada uma única vez, mas pode ser utilizada diversas vezes. É uma das estruturas mais básicas para prover reusabilidade.

Criação

Para declarar uma função em PHP, utiliza-se o operador function seguido do nome que desejamos lhe atribuir, sem espaços em branco e iniciando obrigatoriamente com uma letra. Na mesma linha, digitamos a lista de argumentos (parâmetros) que a função irá receber, separados por vírgula. Em seguida, encapsulado por chaves {}, vem o código da função. No final, utiliza-se a cláusula return para retornar o resultado da função (integer, string, array, objeto, etc.).

```
<?php
// exemplo de função
function nome_da_funcao($arg1, $arg2, $argN){
      $valor=$arg1+$arg2+$argN;
      return $valor;
}
?>
```

No exemplo a seguir, criamos uma função que calcula o índice de obesidade de alguém. A função recebe dois parâmetros ($peso e $altura) e retorna um valor definido por uma fórmula.

```
<?php
function calcula_obesidade($peso, $altura){
      return $peso/($altura*$altura);
}
echo calcula_obesidade(70, 1.85);
?>
```

Resultado: 20.45288531775

Retornando Valores de uma Função

É possível a uma função retornar um valor,o qual pode ser de qualquer tipo aceito pelo PHP (inteiro, ponto flutuante, booleano, string, array, etc.). Para isso, devemos utilizar a instrução return seguida do valor que desejamos retornar (ou expressão).

```
<?php
function cubo($valor){
      $cubo=$valor*$valor*$valor;
      return $cubo;
}
$valor=7;
echo "O cubo de $valor é:".cubo($valor);
?>
```

O resultado desse script é: O cubo de 7 é 343.

Valores são retornados pelo uso de comandos opcionais de retorno. Qualquer tipo pode ser retornado, incluindo listas e objetos.

```
<?php
function teste($numero){
      return $numero+$numero*2;
}
echo teste(7);// imprime '21'.
?>
```

Criando bloco de códigos reutilizáveis

Quando estiver projetando seu site da Web, você apontará alguns elementos encontrados em muitas páginas em todo o site. Tais elementos podem ser barras de

navegação ou uma linha de base com o endereço de e-mail do seu webmaster. Outros elementos podem ser fragmentos de códigos para exibir o dia, dados computados financeiros padrão e muitos outros códigos padronizados de HTML ou PHP, como por exemplo, constantes.

require()

A instrução require() e include() são idênticos em todas as formas exceto pela manipulação de erros. O include() produz Warning enquanto require() produzirá um Fatal Error. Em outras palavras, não hesite em utilizar require() se na falta de um arquivo quiser parar o processamento da página. O include() não se comporta da mesma maneira, e o script poderá continuar nessa situação. Veja os exemplos abaixo de sua utilização:

```
<?php
require 'teste.php';
require $arquivo;
require ('arquivo.txt');
?>
```

Nota: Até o PHP 4.0.2, havia o seguinte comportamento: require() ocorre mesmo que a linha onde ele está nunca seja executada. É por isso que instruções condicionais não afetam require(). Entretanto, se a linha onde ocorre o require() não for executada, nada do código incluído do arquivo também será. Similarmente, estruturas de loop não afetam o funcionamento do require(). Mas o código incluído pela função será submetido ao loop. A instrução require() apenas ocorre uma vez.

include( )

A instrução include() inclui e avalia o arquivo informado. Sua semelhança com o require dispensa maiores explicações.Qualquer variável disponível da linha onde a chamada da inclusão ocorre estará disponível para o arquivo incluído, daquele ponto em diante. Veja os exemplos de include():

```
<?php
$nome='Leonardo';
$apelido='Leo';
?>
```

Ilustração: includ.php

```
<?php
echo "O nome é $nome e seu apelido é $apelido.";
// As variáveis estão vazias
include 'includ.php';
echo "O nome é $nome e seu apelido é $apelido.";
// As variáveis, neste caso, contém as informações inclusas
?>
```

Se o include ocorre dentro de uma função do arquivo principal, então todo o código incluído será executado como se ele tivesse sido definido dentro daquela função. Da mesma forma, ele seguirá o escopo de variáveis da função.

```
<?php
function teste(){
      global $nome;
      include 'includ.php';
      echo "O nome é $nome e seu apelido é $apelido.";
}
/* includ.php está no escopo da função teste(), então NÃO
está disponível fora de seu escopo. $nome estará porque
ela foi declarada como global */
teste();
echo "O nome é $nome e seu apelido é $apelido.";
/* imprime somente $nome */
?>
```

Quando um arquivo é incluído, o interpretador sai do modo PHP e entra no modo HTML (no começo do arquivo incluído), e alterna novamente no seu fim. Por isso, qualquer código dentro do arquivo incluído que precisa ser executado como código PHP tem de ser delimitado por tags lidas de abertura e fechamento. Se "URL fopen wrappers" estão ativas no PHP (normalmente na configuração padrão), você pode especificar um arquivo utilizando uma URL (via HTTP) em vez de um caminho local.

Se o servidor apontado interpreta o arquivo informado como código PHP, variáveis podem ser passadas ao arquivo incluído na URL de requisição como num HTTP GET. Isto não é necessariamente a mesma coisa que incluir o arquivo e compartilhar o escopo de variável do arquivo principal: o script será executado no servidor remoto e apenas seu resultado será incluído no script local.

```
<?php
/* Este exemplo assume que WWW.exemplo.com está configurado para
interpretar arquivos .php, mas não .txt. Além, 'Funciona' aqui
significa que as variáveis $teste1 e $teste2 estão disponíveis no
arquivo incluído; Não funciona: arquivos .txt não são manipulados
WWW.exemplo.com como PHP */
include 'http://WWW.exemplo.com/arquivo.txt?teste=1&test2=2';
/* Não funciona: procura por um arquivo chamado
arquivo.php?teste=1&teste2=2 no sistema de arquivo local. */
include 'arquivo.php?teste=1&teste2=2';
// Funciona.
include 'http://WWW.exemplo.com/arquivo.txt?teste=1&test2=2';
$teste = 1;
$teste2 = 2;
include 'arquivo.txt'; // Funciona.
include 'arquivo.php'; // Funciona.
?>
```

Por serem include() e require() dois construtores de linguagem especiais, você precisa delimitá-los como um bloco de instruções quando utilizados dentro de instruções condicionais.

```
<?php
// Isto está errado e não funcionará como desejado.
if($condicao)
include $arquivo;
else
include $outro;
// E este está correto.
if($condicao){
      include $arquivo;
}else{
      include $outro;
}
?>
```

Também é possível executar uma instrução return() dentro de um arquivo incluído de maneira a finalizar o processamento daquele arquivo e retornar para o script que o chamou. Também é possível retornar valores de arquivos incluídos. Você pode pegar o valor de retorno de um include como faria com uma função normal. O return() se aplica apenas para a função e não para todo o arquivo.

```
<?php
$var='PHP';
return $var;
?>
```

Ilustração: exemplo1.php

```
<?php
$var='PHP';
?>
```

Ilustração: exemplo2.php

```
<?php
$teste=include 'exemplo1.php';
echo $teste;// Imprime 'PHP'.
$teste2=include 'exemplo2.php';
echo $teste2;// Imprime 1.
?>
```

$teste2 assimila o valor 1 porque a inclusão foi realizada com sucesso. Verifique a diferença entre os exemplos. O primeiro utiliza return( ) dentro do arquivo incluído enquanto que o outro não. Há outras maneiras de "incluir" arquivos dentro de variáveis, com fopen( ), file( ) ou utilizando include( ).

Funções de Data

Uma das tarefas mais comum, em se tratando de PHP, é o uso de funções para trabalhar com Datas. Aqui veremos as mais comuns.

Date

A função date() é utilizada para formatar data e hora locais (do servidor onde o PHP está rodando). Tabela com os caracteres utilizados junto com a função.

Caractere	Descrição	Exemplo de Valores
a	Retorna Ante meridiem e Post meridiem.	am or pm
A	Retorna Ante meridiem e Post meridiem.	AM or PM
d	Dia do mês, 2 dígitos com leading zeros.	01 to 31
D	Uma representação textual de um dia.	Mon through Sun
F	Uma representação textual de um mês.	Janeiro à Dezembro
g	12-hour formato de hora, sem zeros.	1 a 12
G	24-hour formato de hora, sem zeros.	0 a 23
h	12-hour formato de hora, com zeros.	01 a 12
H	24-hour formato de hora, com zeros.	00 a 23
i	Minutos com zeros.	00 a 59
I	Se a data está ou não em horário de verão.	1-Sim, 0-Não
j	Dia do mês sem leading zeros.	
l	Representação de um dia da semana.	Domingo à Sábado
L	Se é ano bissexto.	1-Sim, 0-Não
m	Representação numérica de um mês.	01 a 12
M	Uma representação textual curta de um mês.	Jan à Dec
n	Representação numérica de um mês.	1 a 12
O	Diferença ao horário de Greenwich em horas.	Exemplo: +0200
r	RFC 2822 formatted date.	Day, Date Time “0”
s	Segundos, com leading zeros.	00 a 59
S	Sufixo ordinal inglês para o dia do mês.	st, nd, rd ou th
t	Número de dias do dado mês.	28 a 31
T	Timezone setting of this machine.	Examples: EST, MDT...
U	Secondes sinces the Unix Epoch.	See also time()
w	Representação numérica do dia da semana.	0 (Dom) à 6 (Sáb)
W	Número da semana do ano ISO-8601.	Exemplo: 42
y	Representação do ano em dois dígitos.	Exemplo: 99
Y	Uma representação completa do ano.	Exemplo: 1999
z	O dia do ano (começando de 0).	0 a 365
Z	Timezone offset in seconds.	-43200 até 43200

Vejamos sua utilização:

```
<?php
$dataLocal=date(”d/m/Y H:i:s”);
echo $dataLocal;// mostra data e hora local
?>
```

Getdate

A função getdate() retorna a data e hora local, em formato de Array. Abaixo, a tabela com as respectivas chaves retornadas no array:

Chave	Descrição	Exemplos
“seconds”	Representação numérica dos segundos.	0 a 59
“minutes”	Representação numérica dos minutos.	0 a 59
“hours”	Representação numérica das horas.	0 a 23
“mday”	Representação numérica do dia do mês.	1 a 31
“wday”	Representação numérica do dia da semana.	0-Sun à 6-Sáb
“mon”	Representação numérica de um mês.	1 a 12
“year”	Representação numérica completa do ano.	199 ou 2003
“yday”	Representação numérica do dia do ano.	0 a 366
“weekday”	Representação numérica do dia da semana.	Sunday à Saturday
“month”	Representação textual completa de um mês.	January à December
0	Segundos desde a época UNIX, similar aos valores retornados por time() e usados por date().	System Dependent, de -2147483648 a 2147483647

Vejamos sua utilização:

```
<?php
$hoje=getdate();
echo $hoje['year'];// Imprime o ano atual
?>
```

Time

A função time() retorna o timestamp Unix atual.Um timestamp é a hora atual medida no número de segundos desde a Era Unix (01/01/1970). Exemplo:

```
<?php
$tsAtual=time();
echo $tsAtual;// Imprime timestamp atual na tela
?>
```

Mktime

A função mktime() é utilizada para se retornar um timestamp Unix (formato utilizado para datas no PHP) a partir de uma data qualquer maior que 01/01/1970. Sua sintaxe é simples:

```
<?php
mktime (hora, minuto, segundos, mês, dia, ano)
?>
```

Depois de obtido o timestamp, você pode utilizá-lo com as funções previamente citadas, a exemplo da função date. Veja um exemplo:

```
<?php
// Obtém timestamp de 01/04/2006 às 00:00:00
$timeStamp=mktime(0, 0, 0, 4, 1, 2006);
// Retorna o dia da semana em inglês, usando a função date
$data=date(”1”,$timeStamp);
// Imprime o dia da semana referente a data 01/04/2006
echo $data;
?>
```

Formulários

Os formulários são recursos oferecidos desde o HTML que, dentre outras funcionalidades, tem o intuito de interagir com o usuário. Seja em forma de enquetes, contato, pesquisas, etc.

Métodos GET e POST

Para se trabalhar com formulários existem 2 tipos de métodos utilizados para o tráfego dos dados: GET e POST.

O primeiro deles, o GET, trafega os dados de forma visível ao usuário, na barra de endereços do navegador (browser). É útil para fazer marcação de URLs, como exemplo de sites de busca, onde o usuário deseja salvar a página para consulta posterior.

Já o método POST trafega os dados de forma oculta, onde somente o navegador (browser) e o servidor de onde foi postado os dados, tem acesso à leitura destes. É utilizado para tráfego de dados de pesquisas, por exemplo, onde o usuário não usará novamente a página para favoritar, etc. Esses métodos são definidos na tag <form> do HTML:

```
<form method=“post” action=“arquivo.php”></form>
```

No lado do PHP, os dados são tratados utilizando-se de variáveis globais. Para o método GET é usado a variável $_GET, e no método POST, a variável $_POST. Existe ainda a variável $_REQUEST, que associa automaticamente o método recebido (GET ou POST). Todas estas variáveis são tratadas como arrays, onde a chave é o nome do campo do formulário. Assim se você tem um campo chamado nome, no PHP pode ser apontado como $_REQUEST[‘nome’] .

A diferença entre GET e POST está basicamente na capacidade do envio de dados, uma vez que GET é limitado à cerca de 2KB de dados e POST não possui limite. Outra diferença está na forma de envio:

- ***GET:*** envia os dados anexados ao nome do script informado (utiliza-se para isso o símbolo ? após o nome do script, seguido da relação de campos e seus respectivos valores separados por &).
- ***POST:*** os dados são enviados no corpo da mensagem que será enviada no servidor.

Formulários na prática

Nesta seção veremos na prática um exemplo de um formulário completo, com uso dos tipos texto, senha, radio, etc. Uma observação importante: se você tem vários campos com o mesmo nome e de múltipla escolha no formulário (como é o caso dos checkbox's), esses campos serão tratados como array do array global.

O exemplo a seguir é figurativo (uma pesquisa de gostos por livros), e será usado como base para você aprender a manusear dados de um formulário. Exemplo de um formulário HTML:

```
<form method="post" action="envia.php">
<fieldset>
<legend>Dados pessoais</legend>
<label>Nome:</label><br />
<input type="text" name="nome" size="30" /><br />
<label>Endereço:</label><br />
<input type="text" name="endereco" size="30" /><br />
</fieldset>
<fieldset>
<legend>Pesquisa</legend>
<label>Quantos livros você compra por ano?</label><br/>
<input type="radio" name="livros" size="0" />Nenhum<br />
<input type="radio" name="livros" size="1-5" />De 1 a 5<br />
<input type="radio" name="livros" size="5-10" />De 5 a 10<br />
<input type="radio" name="livros" size="10-20" />De 10 a 20<br />
<label>Qual seu estilo de livro favorito/</label><br/>
<input type="checkbox" name= "estilo[]" size="Romance" />Romance<br />
<input type="checkbox" name= "estilo[]" size="Drama" />Drama<br />
<input type="checkbox" name= "estilo[]" size="Suspense" />Suspense<br
/>
<input type="checkbox" name="estilo[]" size="Tecnico" />Técnico<br />
</fieldset>
<input type="submit" />
</form>
```

Exemplo do arquivo envia.php, que será submetido pelo formulário:

```
<?php
echo "Nome: ".$_POST['nome']."<br />";
echo "Endereço: ".$_POST['endereco']."<br />";
echo "Qt. Livros: ".$_POST['livros']."<br />";
echo "Estilo: ";
foreach ($_POST['nome'] as $estilo){echo $estilo. " - ";}
?>
```

Perceba que já fizemos uso do construtor foreach acima, pois o campo estilo[] do formulário é múltipla escolha, logo, ele é considerado um array no PHP. Repare ainda que o campo possui os colchetes []. Isso precisa ser determinado para que o PHP entenda como um Array.

Sessões e Cookies

Neste capítulo vamos estudar duas das funcionalidades mais utilizadas no PHP. Ambas são utilizadas para armazenar dados por um período e diferem no sentido de que as sessões expiram quando o usuário fecha o browser, e os Cookies podem perdurar um tempo definido pelo programador.

Uma das utilizações destas funcionalidades é, por exemplo, o carrinho de compras, que armazenam as compras feitas pelo usuário. Outro exemplo é o de criar função de login, área restrita de acesso, etc. Vamos estudar cada uma separadamente.

Sessões

As sessões são forma simples de armazenar dados temporários. Os dados são apagados assim que o usuário fechar o browser. Mas, é importante que você saiba, que você pode excluir sessões quando quiser. Imagine o seguinte problema: você tem um site, e quer que em algumas seções, apenas usuários que saibam uma determinada senha consigam acessar. Você pode utilizar sessões neste caso.

O ponto de partida para o uso de sessões é com sua inicialização, fazendo uso da função session_start(). Vale lembrar que até usá-la, não se pode imprimir nada na tela (seja via echo, print, avisos ou erros de algum trecho de código). Por estes motivos, recomenda-se que você insira a função logo na primeira linha do seu script. Sempre que for utilizar sessões, você vai precisar inicializar esta função. Exemplo:

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

Dentro da seção, fazemos uso da variável global $_SESSION para definir variáveis e valores.

Vamos restringir o acesso a algumas sessões utilizando uma variável de sessão para saber se o usuário passou pela tela de login e informou a senha corretamente. Nas seções onde a área será restrita, fazemos a verificação se o usuário logou. Veja o exemplo da pagina_secreta.php:

```
<?php
session_start();
if !$_SESSION[‘logado’]){
      header(“Location: login.htm”);
      die;
}
?>
```

Entendendo o código: primeiro inicializamos a sessão com a função session_start(). Em seguida fazemos a verificação: se não existir (uso da !) a variável de sessão “logado” ou se ela for igual a falso (booleano), redirecionamos para a página de login (função header) e finalizamos o script (função die). Caso contrário, o script prossegue, e a sessão é mostrada

normalmente. Vamos entender agora a página de login. Primeiro vamos criar o formulário do login.htm:

```
<form action="login.php" method="post">
      <fieldset>
            <legend>Informe a senha</legend>
            <label>Senha: </label>
            <input type="password" name="senha" size="10" />
      </fieldset>
</form>
```

Agora vejamos o arquivo login.php:

```
<?php
session_start();
if ($_POST['senha']=='secreta'){
      $_SESSION['senha'] = true;
      header("Location: pagina_secreta.php");
      die;
}else{
      echo "Senha incorreta";
}
?>
```

Entendendo o código: primeiro é feita a inicialização da sessão (função session_start). Em seguida é feita a comparação, se a senha digitada é igual a "secreta" (pode ser alterada), em caso verdadeiro é definida a variável de sessão "logado" como verdadeiro e redireciona o usuário para a página secreta. Caso contrário, imprime na tela "senha incorreta".

Cookies

Cookies, na livre tradução, são biscoitos. Cookies são mecanismos para armazenar e consultar informações nos browsers dos visitantes da página. O PHP atribui cookies utilizando a função setcookie, que deve ser utilizada antes de qualquer impressão na página (mesmo esquema das sessões).

O uso de cookies não é recomendado quando se trata de informações sigilosas. Os dados dos cookies são armazenados no diretório de arquivos temporários do visitante, sendo facilmente visualizado por pessoas mal intencionadas.

A diferença entre sessões e cookies, é que os cookies podem ser acessados mesmo depois de o usuário ter fechado seu browser. Outra questão importante, é que a opção "aceitar cookies" (do navegador – browser) pode ser desativada a qualquer momento pelo visitante. Para uma transmissão de dados segura é recomendável o uso de sessões.

Sintaxe básica:

```
setcookie("nome_do_cookie","seu_valor","tempo_de_vida","path","domínio
","conexão_segura")
```

- Nome_do_cookie = É o nome que, posteriormente, se tornará a variável e o que o servirá de referência para indicar o cookie.
- Seu_valor = É o valor que a variável possuirá. Esse valor pode ser de todos os tipos.
- Tempo_de_vida = É o tempo, em segundos, que o cookie existirá no computador do visitante. Uma vez excedido esse prazo o cookie se apaga de modo irrecuperável. Se esse argumento ficar vazio, o cookie se apagará quando o visitante fechar o browser.
- Path = endereço da página que gerou o cookie – automático
- Domínio = domínio ao qual pertence o cookie – automático
- Conexão_segura = Indica se o cookie deverá ser transmitido somente em uma conexão segura HTTPS.

Depois de definida um cookie, você pode acessá-lo através da variável global $_COOKIE[‘nome_do_cookie’].