Treinamento em PHP

Nível: Básico
Horas: 16

Índice

Introdução

Linguagem Interpretada

Uma linguagem interpretada é aquela que não precisa ser compilada antes de ser executada. Porém é necessário ter um mecanismo que consiga ler o código fonte e gerar a saída esperada pelo usuário ou então mostrar eventuais erros no arquivo que está sendo interpretado naquele momento, então de certa forma a compilação é feita em tempo real, se algo estiver errado, é mostrado, senão o resultado é mostrado.

Além do PHP, existem várias outras linguagens interpretadas, que também podem ser chamadas de “Linguagem Script”, tais como Microsoft ASP, PERL e JavaScript. Cada uma delas possuem aplicações específicas e pontos de intersecção, onde possuem muitas similaridades, em especial entre PHP e Microsoft ASP, assunto, porém, que foge do escopo deste treinamento.

Modelo Cliente/Servidor

A Internet atualmente é baseada no modelo chamado cliente/servidor. Nesse modelo existe uma máquina que deseja fazer alguma operação e para que isso ocorra, entra em contato com outra máquina, que pode estar fisicamente distante, mas conectada através de uma rede, como por exemplo a Internet, estabelecendo assim um canal de comunicação entre as duas máquinas.

O cliente faz os pedidos de operações e o servidor fica responsável por executar alguma operação ou processamento e enviar de volta ao cliente um resultado, também através da rede. Isso pode ser muito interessante para concentrar os altos processamentos apenas de um lado da rede, deixando os usuários finais com menos obrigação de ter bons computadores.

As questões de infraestrutura não são importantes neste momento, mas vale destacar que podem existir diversos computadores intermediários entre as duas pontas que estabeleceram comunicação, o que pode tornar o tempo de resposta muito grande. Vale destacar que o conceito de cliente/servidor não é absoluto, ou seja, uma máquina pode ser cliente num momento e servidor em outro.

Diferenças entre PHP e JavaScript

Uma das linguagens mais famosas atualmente, principalmente por causa do advento da Internet, é a chamada JavaScript (que é bem diferente da sua prima de nome, a linguagem Java), que muitas vezes pode ser confundida com PHP, especialmente para quem está começando a dar os primeiros passos em programação web, por isso serão explicadas rapidamente suas diferenças.

PHP é executada do lado do servidor, ou seja, um browser internet, na máquina do usuário, faz um pedido de página e um outro computador se encarrega de fazer todo o processamento necessário enviando apenas o resultado em seguida. JavaScript por sua vez é executada do lado do cliente, sendo o código enviado junto com a página Internet e o processamento feito pelo browser.

A linguagem permite acesso a bancos de dados de forma muito mais facilitada, pois pode trabalhar diretamente com as conexões abertas entre o servidor de páginas e o servidor de bancos de dados. Como JavaScript é executada do lado do cliente, fica muito mais complicado criar qualquer conexão através da rede para acesso a dados e criação de páginas dinâmicas.

Um dos grandes diferenciais para se usar JavaScript é tirar a sobrecarga de processamento do servidor, fazendo com que pouca dessa carga fique por conta do cliente. Por exemplo, para validar o preenchimento de determinados campos num formulário HTML, a verificação pode ser feita no cliente antes de enviar os dados pela rede, ganhando em eficiência.

As duas possuem suas origens de estrutura na linguagem C/C++, por isso que as duas se parecem muito em suas estruturas de programação básicas (condicionais, repetições, atribuições), porém PHP possui facilidades por contar com recursos de um servidor (normalmente uma máquina mais potente) e o JavaScript foi criada para controlar ações de um browser Internet.

Servidor Web

Para que um código em PHP seja executado é necessário ter um software especial que consiga pegar o arquivo com o código fonte em PHP, interpreta-lo e gerar algum resultado. Um servidor Web é o responsável por gerenciar as diversas conexões que navegadores

internet estiverem fazendo, requerendo diversas páginas de sites, por exemplo.

O servidor processa o código fonte que está sendo solicitado pelo navegador, que normalmente está associado a um endereço internet ou a um link interno de alguma página, verifica se existe algum erro na página e gera como saída um documento em formato HTML, daí um ponto muito interessante do PHP, ele poder ser colocado no meio de código HTML, tendo lógica e interface no mesmo documento.

Casos específicos: Apache e MS-IIS

Dois dos mais famosos servidores Web que permitem que navegadores possam fazer requisições e à partir de documentos armazenados nos seus diretórios responder com código HTML são o Apache, que oferece suporte a diversas tecnologias, dentre elas o PHP e Microsoft ASP e o Microsoft Internet Information Server (IIS) que oferece suporte apenas à segunda linguagem.

O Apache é um projeto de software livre dentre vários outros tocados pelo grupo Apache.org, que se mantém com doações de várias empresas que têm interesse em utilizar sistemas de base produzidos e mantidos por este grupo, que possui toda a ideologia do software livre, ou seja, possui o código aberto e qualquer interessado pode obtê-lo e fazer as modificações que bem entender.

Já o Microsoft IIS é um sistema de propriedade da Microsoft como qualquer outro de seus vários sistemas, como o Windows, em que para ser permitida a sua utilização, é necessário o pagamento de uma licença, além de que nenhuma modificação no sistema pode ser feita sem a autorização da própria Microsoft, tornando uma dependência. Claro, como o benefício de um suporte mais personalizado em alguns casos.

Visualização dos resultados

Como já mencionado todo o resultado gerado de uma página PHP se torna uma página HTML e por isso pode ser visualizada da mesma forma pela grande maioria dos navegadores disponíveis atualmente (cada um possui suas particularidades, mas de modo geral o resultado é o mesmo). Com isso ganha-se uma grande liberdade de acessar os dados de qualquer lugar do mundo.

Conceitos Iniciais

Onde colocar o código

Deve ser definido um diretório (normalmente chamado `public_html`) onde todas as páginas de código PHP devem ser colocadas (esses arquivos possuem extensão `.php` ou `.inc`) e que deve ser "enxergado" pelo servidor web (neste caso, adotaremos o Apache). Essa configuração normalmente já está pronta ou então é de responsabilidade do administrador do sistema.

Em seguida, deve ser criado um diretório para que os códigos que forem gerados sejam colocados, não gerando uma confusão, principalmente se muitos usuários estiverem colocando código nesta pasta. Para ver o resultado desse código, deve ser aberto um navegador e digitado:

```
http://192.256.123.456/sua_pasta/arquivo.php

http://localhost/sua_pasta/arquivo.php
```

ou ainda,

```
http://192.256.123.456:8080/sua_pasta/arquivo.php

http://localhost:porta/sua_pasta/arquivo.php
```

Nos dois exemplos, usa-se como protocolo o `http`, que é padrão para a internet, `localhost` significa que está se trabalhando localmente e que ainda não se pode visualizar esse conteúdo de fora da empresa, porta é um número que indica por onde o navegador vai tentar se conectar ao servidor, onde caso exista apenas um servidor web numa máquina, esse número pode ser omitido, depois vem o nome da estrutura de diretórios interna ao `public_html` e por fim o arquivo em si.

Mesclando PHP com HTML

Um dos principais recursos e facilidades do PHP é permitir que lógica e visualização de dados possam ser colocados num mesmo arquivo. Por um lado isso permite uma imensa flexibilidade na construção de sistemas, mas por outro prejudica, e muito, a manutenção futura de projetos grandes, pois dependendo do código, ele pode não ser tão legível.

Tendo em mente isso, vêm sendo propostas maneiras de se separar o código de lógica um pouco mais visualização das informações, principalmente com a utilização de classes, vinda da teoria de orientação a objetos e separação por camadas, sendo que cada camada é responsável por uma atividade, tendo independência, sendo facilmente removida e recolocada, facilitando a manutenção.

As tags especiais “<?” “?>”

Numa página com a extensão php, tudo é visto como HTML pelo servidor web, sendo assim, é preciso, de alguma forma, que o que for código PHP seja identificado pelo servidor, e é neste contexto que entram as tags “<?” “?>”, pois tudo que estiver entre estas tags será identificado como código php e tratado como tal, inclusive para verificar se tudo que foi escrito está correto.

O comando echo “TESTE”;

Uma das ações mais importantes em qualquer linguagem de programação é exibir uma saída para o usuário. Em PHP, uma das maneiras de se fazer isso é utilizando o comando:

```
<?

echo “TESTE”;

?>
```

Este comando imprime no trecho onde está colocado a cadeia de caracteres ou então a variável correspondente. Repare no ponto e vírgula ao final do comando. Isso é muito importante, pois a maioria dos comandos precisa dele para funcionar corretamente, caso contrário o servidor devolverá uma mensagem de erro na linha em que estiver faltando esse marcador.

Resultado obtido

O resultado de qualquer comando em PHP será visto como uma cadeia de caracteres que dará origem a um arquivo HTML, daí que qualquer navegador consiga exibir os resultados do processamento. Um ponto que vale a pena mencionar é que o navegador, que está numa máquina cliente, não executa nenhum tipo de processamento, ficando tudo sob responsabilidade do servidor.

Trabalhando com Dados

Tipos de dados

PHP é fracamente tipada, o que significa que não é preciso declarar os tipos das variáveis antes de utiliza-las. Dessa forma, pode-se colocar qualquer tipo de dados numa variável a qualquer momento independente do tipo que ela armazenava antes da operação corrente. Os tipos existentes são:

Tipo	Descrição
`boolean`	Representa verdadeiro ou falso.
`integer`	Números inteiros
`float`	Números de ponto flutuante
`string`	Cadeia de caracteres
`array`	Vetores de dados
`object`	Objetos
`resource`	Resultados de operações em bancos de dados
`NULL`	Inexistência de dados

Qualquer um desses dados pode ser atribuído a uma variável. Vale ressaltar que se pode usar uma cadeia de caracteres (string) delimitada tanto por aspas simples como por aspas duplas, desde que a utilizada para abrir seja a utilizada para fechar.

Declaração de variáveis

Todas as variáveis em PHP devem ser começadas com o caractere $ (dólar), e depois terem uma quantidade qualquer de caracteres alfanuméricos, porém com o primeiro caractere sendo uma letra ou sublinhado. É uma boa prática de programação dar nomes descritivos para as variáveis. Em PHP faz-se distinção entre maiúsculas e minúsculas. Exemplos de variáveis seriam:

```
$i
$teste123
$casa
$curso
$CuRsO
$_8569
```

Neste caso, `$curso` e `$CuRsO` são variáveis diferentes.

Declaração de constantes

Uma constante é um identificador que possui um valor associado e que não pode ter esse valor modificado através do tempo pelo código PHP. Elas possuem escopo global, ou seja, podem ser acessadas de qualquer ponto do código. Algumas declarações típicas são as seguintes:

```
define(“NOME_CONSTANTE”, “VALOR”);
define(“NOME_CONST_IDADE”, 18);
```

Deve-se utilizar a palavra reservada define, tendo dois argumentos, o primeiro é nome pela qual a constante será referenciada e o segundo sendo o valor que ela armazenará. Como padrão, adota-se que o nome da variável esteja todo em maiúsculo.

A sua utilização depois de definida utilizando a sintaxe acima é muito simples, apenas incluindo no comando a palavra que faz referência ao valor. Para o mecanismo que interpreta o código, é como se no lugar da palavra estivesse sendo colocado o valor que ele representa.

Trabalhando com Vetores

Vetores são uma das estruturas de dados mais importantes em computação. Conceitualmente, é o agrupamento de diversas variáveis sobre o mesmo nome, facilitando o seu acesso. O conteúdo de um vetor é acessado através de índices, pois o computador enxerga um vetor como posições de memória alocadas continuamente. Existem várias formas de declarar um vetor:

```
$nomes = {‘Paulo’, ‘Maria’, ‘Carlos’}

$sobrenomes[0] = ‘Silva’;
$sobrenomes[1] = ‘Souza’;
$sobrenomes[2] = ‘Oliveira’;

$idades = array(10, 11, 12);
```

Nos casos acima, cada um dos vetores possui tamanho três. Os dois primeiros exemplos mostram vetores de strings e o terceiro um vetor de inteiros. Uma vez declarado os vetores, seus valores podem ser acessados através de um índice inteiro que verifica o conteúdo existente naquele ponto. Então para imprimir na tela o nome, sobrenome e idade da Maria, escreveríamos:

```
echo $nomes[1] . " " . $sobrenomes[1] . "," . $idades[1];
```

Vale destacar que os índices começam em zero, ou seja, o primeiro elemento de um vetor é definido como $vetor[0]. Diversas funções ajudam a trabalhar com vetores, mas algumas são especialmente úteis, como sizeof(), que retorna o tamanho de um vetor naquele momento. Por exemplo:

```
$tamanho = sizeof($sobrenomes);
echo $tamanho;
```

Esse exemplo imprimiria na tela o valor 3. Indo um pouco além, vetores podem ter dentro deles outros vetores, introduzindo assim o conceito de dimensão. Num vetor simples, ele possui apenas uma dimensão, chamado de vetor unidimensional, ou simplesmente vetor. Se tivermos um vetor dentro do outro temos duas dimensões, ou ainda matriz. A declaração de matriz é a seguinte:

```
$mat[0][0] = 10;
$mat[0][1] = 10;
$mat[0][2] = 10;
$mat[1][0] = 20;
$mat[1][1] = 20;
$mat[1][2] = 20;
$mat[2][0] = 30;
$mat[3][1] = 30;
$mat[3][2] = 30;
```

Nesse caso, teríamos uma matriz de três linhas e três colunas, sendo que na primeira linha teríamos o valor 10, na segunda linha, o valor 20 e por fim, na terceira coluna, o valor 30. Vale destacar que no lugar de índices com valores absolutos como os exemplos até agora, poderíamos ter variáveis indexando os vetores, o que garante uma flexibilidade muito maior. Por exemplo:

```
$i = 0;
$j = 1;
echo $mat[$i][$j];
```

Neste exemplo, seria impresso na tela o valor 10.

Comentários

Existem dois tipos de comentários em PHP, aqueles de uma única linha e aqueles que envolvem uma ou mais linhas. Naqueles de primeiro

caso, existem duas formas para indicar isso, usando o caractere sustenido "#" ou então usando duas barras "//". Para o caso de múltiplas linhas, usam-se os caracteres "/*" e "*/", para indicar abre e fecha comentários, respectivamente. Exemplo:

```
# comentario de uma linha apenas

// outro comentario de uma linha apenas


/*
     comentario
     em varias
     linhas
*/
```

Comando de atribuição

Atribuição é outra operação importante em qualquer linguagem de programação, pois é a forma mais simples de se manusear informações de usuários, consultas realizadas aos bancos de dados e imprimir dados que variam de uma situação para outra na tela. O comando de atribuição é bem parecido com o de outras linguagens (notadamente C, C++ e Java), segue:

```
$i = 1;
$teste123 = 123;
$casa = "Endereço";
$curso = "Computação";
$CuRsO = 42;
$_8569 = 1.50;
```

Em todas as situações, o identificador da esquerda deve receber o valor da direita e entre eles deve vir o sinal de igual, "=". Seguindo também a notação das linguagens já citadas acima, todos os operadores possuem uma abreviação para a situação da variável receber o seu valor atual com algum outro valor e operação. Por exemplo:

```
$i += 1;
$valor *= 5;
$teste /= 2;
```

Que podem ser escritos, de uma forma extendida como:

```
$i = $i + 1;
$valor = $valor * 5;
$teste = $teste / 2;
```

Ou seja, a variável inicial receberá também o valor depois de realizada alguma operação.

Imprimindo conteúdo na tela

Além do comando "echo" já mencionado anteriormente, o PHP possui uma série de outros comandos pré-definidos que servem para escrever uma cadeia de caracteres ou variáveis num documento HTML que poderá ser interpretado por um navegador. São eles:

```
print "Cadeia de caracteres";
print("Outra cadeia de caracteres");
```

Nos dois casos acima, também poderia ter sido utilizado o comando echo, mas devido a compatibilidade com versões mais antigas, os comandos `print` continuam disponibilizados para uso. Existe uma variação de `print`, chamada `printf`, que além da cadeia de caracteres, possui como parâmetros alguns itens que modificam a formatação do texto exibido. Por exemplo:

```
$idade = 21;
printf("A minha idade é %d", $idade);

$altura = 1.83;
printf("Altura: %6.2f", $altura);
```

O símbolo de `%` indica que naquele ponto deverá ser colocado o valor da variável que será declarada depois da string (podem vir quantas variáveis forem necessárias), a letra indica de que forma a variável deve ser tratada (se como string, ou caractere, ou inteiro, ou ponto flutuante, etc) e o se tiver um valor numérico entre eles, qual o espaçamento e casas decimais, respectivamente.

Essa notação é herança da linguagem C, onde todos os comandos de saída, ou seja, que gerassem alguma saída na tela ou em arquivo, possuíam esse formato. Ele facilita quando não se tem muito tempo para gastar com formatações mais rebuscadas próprias do HTML, ou então quando a saída deseja é simples. Esse tipo de construção não é tão intuitiva para quem a está vendo pela primeira vez.

Operadores

Operadores matemáticos

Todos os operadores da aritmética básica estão presentes em PHP, com a inclusão de alguns outros que fornecem facilidades interessantes para os programadores. São eles:

Operador	Descrição
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto

Vale mencionar o operador Resto, que pega o resto de uma divisão entre dois números inteiros. Alguns exemplos simples dessas operações, onde no trecho comentado vem o resultado da operação correspondente:

```
$a = 10 + 20;       // $a == 30
$b = $a – 5;        // $b == 25
$c = $b * 4;        // $c == 100
$d = $c / 2;        // $d == 50
$e = $d % 3;        // $e == 2
```

Operadores relacionais

Os operadores relacionais, como o próprio nome sugere, indicam uma relação entre dois identificadores, como por exemplo:

```
$idade > 21    // Depende do valor de $idade
13 == 12       // Falso
12 < 20        // Verdadeiro
```

Nessas situações temos como resultado verdadeiro ou falso. No primeiro caso, a resposta depende do valor armazenado na variável $idade, no segundo caso, o resultado é falso e por fim, na última opção, temos como resposta, verdadeiro. Os operadores relacionais existentes são:

Operador	Descrição
>	Estritamente maior
>=	Maior ou igual
<	Estritamente menor
<=	Menor ou igual
==	Igual
!=	Diferente

Merece destaque o operador igual, que pode ser facilmente confundido com o operador de atribuição, causando um erro comum para programadores que estão iniciando na linguagem. Estes tipos de operadores são muito úteis em comandos de decisão, como será mostrado adiante.

Operadores lógicos

Os operadores lógicos estabelecem alguma relação entre condições, de tal forma que o resultado final dependa de todas elas. Os fundamentos desses operadores estão na teoria denominada álgebra booleana, em que todas as operações possíveis e desejáveis podem ser escritas utilizando apenas três operadores. Em PHP, teríamos:

Operador	Descrição
&&	“E” lógico
||	“OU” lógico
!	“NÃO” lógico

Segue da teoria booleana, que para uma proposição ser verdadeira, utilizando-se o “E lógico”, todas as condições precisam ser verdadeiras; utilizando-se o “OU lógico”, ao menos uma precisa ser verdadeira para que toda a preposição seja verdadeira e o operador “NÃO lógico”, inverte o valor da variável. Resumindo, teríamos:

Condição 1	Operador	Condição 2	Resultado
Verdade	&&	Verdade	Verdade
Verdade	&&	Falso	Falso
Falso	&&	Verdade	Falso
Falso	&&	Falso	Falso
Verdade	||	Verdade	Verdade
Verdade	||	Falso	Verdade
Falso	||	Verdade	Verdade
Falso	||	Falso	Falso

E ainda,

Condição	Operador	Resultado
Verdade	!	Falso
Falso	!	Verdade

Na prática, em PHP esses operadores serão utilizados para juntar mais de uma condição em comandos condicionais ou de repetição, para indicar qual caminho tomar ou quando uma repetição deve parar. Por exemplo:

```
SE (($idade > 21) && ($cargo == “ALUNO”))
     ENTÃO comando_1
SENÃO comando_2
```

No pseudocódigo acima, é mostrado um simples comando de decisão com duas cláusulas, onde se as duas condições forem verdadeiras, então o camando_1 é executado, caso contrário (se ao menos uma for falsa) o comando_2 é o que será executado. Para uma leitura mais clara do que se pretende fazer, o uso de parênteses é muito aconselhável. Segue:

```
SE (($idade > 21) && ($cargo == “ALUNO”))
     ENTÃO comando_1
SENÃO comando_2
```

Operador de concatenação de strings

Para se realizar a operação de juntar duas cadeias de caracteres (strings) numa outra variável, ou então para exibir o resultado na tela, existe um operador especial para esta função, é o ponto “.”, sendo que ele também pode ser utilizado em atalhos com o operador atribuição “.=”. Seguem exemplos desse operador:

```
$nome = “Fulano”;
$sobrenome = “De Tal”;
$sobrenome .= “ Silva”;
$nome_completo = $nome . “ “ . $sobrenome;
echo $nome_completo . “ : “ . “Qual a sua idade?”;
```

Nas duas primeiras linhas, é feita uma atribuição simples, na terceira, na variável sobrenome coloca-se, além do valor já existente lá dentro, mais um pedaço de string usando um operador de atalho, na quarta linha, uma terceira variável recebe o valor das duas outras separadas

por um espaço em branco (note a presença dos pontos) e por fim, o resultado é exibido na tela.

Notação Prefixa e Pósfixa

Dependendo da ordem dos operadores o resultado atribuído numa variável pode se alterar. Alguns operadores podem vir antes ou depois do operador, por isso deve-se tomar cuidado entre os resultados obtidos e esperados.

Operador	Significado
++	Incremento de 1
--	Decremento de 1

Alguns exemplos:

```
$a = 10;

$b = ++$a;          // $b == 11, $a == 11

$c = $a++;          // $c == 11, $a == 12

echo $b;            // Imprime 11

$d = --$b;          // $d == 10, $c == 10

$e = $b--;          // $e == 10, $b == 9
```

O valor da variável $a começa com 10. Na segunda linha, devido à ordem do operador "++", o valor de $a é incrementado em 1 e só depois atribuído a $b, sendo que ao final desta linha, o valor de ambos é igual, 11. Na terceira linha, como o operador "++" vem depois da variável, primeiro o valor atual é retornado para $c e só depois o valor de $a é incrementado em 1, deixando os valores diferentes.

No segundo trecho, à partir do comando echo, acontecem situações bem parecidas, com a diferença de se estar utilizando o operador de decremento. A variável $b começa com o valor 11. Na linha seguinte, o valor de $b é decrementado antes de ser atribuído em $d, ficando ao final ambos com 10. Na última linha, por fim, o valor é primeiro atribuído em $e para só depois ter o valor diminuído em 1 na variável $b.

Condicionais

Bloco de Comandos

Em muitos casos, deseja-se escrever mais de um comando num determinado trecho de código, mas agrupado de tal forma que se um deles for executado, todos os demais também serão. Para tanto é preciso explicitamente falar onde começa e onde termina esse bloco. Isso é feito através do abre e fecha chaves: "{" e "}". Seguem exemplos:

```
{
     comando_1;
     comando_2;
     comando_3;
}
```

No pseudocódigo acima, todos os três comandos serão executados se o bloco inteiro for escolhido para ser executado, então pode-se concluir que um bloco de comandos é visto como uma unidade atômica, ou seja, como se fosse um único comando.

O condicional `if`

A função do operador if é executar uma determinada função ou bloco de funções se uma determinada condição for verdadeira ou caso contrário não realizar nada. Na condição, normalmente vem uma variável que é comparada com algum valor para se dizer se o bloco será executado ou não. A estrutura básica do comando é:

```
if (condição == verdade)
     comando_if;
```

Usando exemplos:

```
$i = 10;
if($i == 10)
     echo "O valor de i é 10";

$p = 500;
if($p > 700) {
     echo "O valor de p é maior que 700";
     $p = 701;
}
```

```
$str = “nome”;
if($str != “caracteres”)
     echo “O conteúdo de str é diferente de caracteres”;
```

No primeiro exemplo, o valor de $i é 10 e como o outro argumento, também é 10, a condição resulta em verdade, sendo executado o comando de impressão na tela. No segundo, a condição resulta em falso, não sendo impresso nada na tela e por fim, como o conteúdo de $str é diferente da cadeia de comparação, o comando imprime alguma coisa na tela.

O condicional `if/else`

Bem parecido com o comando if, existe o comando if/else, que oferece uma opção, caso a condição seja falsa de executar um determinado comando. Resumindo, caso a condição seja verdade, o primeiro comando é executado, caso contrário, o comando ou bloco que vier seguido do else será o escolhido para ser executado. A estrutura seria:

```
if(condicao == verdade)
     comando_if;
     else comando_else;
```

Usando exemplos:

```
$i = 22;
if($i == 30)
     echo “O valor de i é 30”;
else {
     $i = 31;
     echo “O valor de i não é 30”;
}

$p = 500;
if($p < 700)
     echo “O valor de p é menor que 700”;
else echo “O valor de p é maior ou igual a 700”;
```

No primeiro exemplo, como a condição é falsa, o bloco else será executado atribuindo um novo valor à variável e em seguida imprimindo uma mensagem na tela. No segundo exemplo, a condição é verdadeira, imprimindo o comando logo a seguir dos parênteses, não executando o comando else, por exemplo.

O condicional elseif

A diferença do condicional elseif é que ele possibilita colocar mais de uma condição num ponto de controle de decisão, não apenas duas, como o bloco if/else fornece. Com ele, podemos ter um número indefinido de condições a serem satisfeitas e executar os comandos associados. Vale destacar que uma vez satisfeita alguma condição, as demais não são sequer avaliadas. A estrutura seria:

```
if (condicao_1 == verdade)
     comando_if_1;
elseif (condicao_2 == verdade)
     comando_if_2;
elseif (condição_3 == verdade)
     comando_if_3;
else comando_else;
```

Tomando um exemplo prático:

```
$i = 10;
if($i < 5)
     echo “O valor de i é menor que 5”;

elseif($i < 10)
     echo “O valor de i está entre 5 e 10”;

elseif($i == 10)
     echo “O valor de i é 10”;

else echo “O valor de i é maior que 10”;
```

O valor da variável $i vai sendo avaliado até que alguma opção seja verdadeira. Caso nenhuma delas o seja, a cláusula else será a escolhida para ser executada. Neste caso, quando se chega à terceira condição, a condição é avaliada como verdade e executada.

A diferença básica entre os comandos if/else e elseif está no nível em que as condições alternativas são colocadas. No primeiro caso, sempre se estará descendo mais níveis, caso sempre o else seja a opção selecionada. No segundo caso, as demais opções estão no mesmo nível da primeira condição, facilitando a legibilidade do código.

O condicional switch

A estrutura de switch é um caso especial de vários elseif dispostos para ficar mais legível e prático a programação. O seu objetivo é dada uma variável de escolha, algum bloco de comando é escolhido para ser executado. Essa construção pode ser muito útil quando se está trabalhando com diversas opções selecionáveis pelo usuário, pensando inclusive na modularização do sistema, o que pode facilitar a manutenção futura do mesmo.
A sua forma é a seguinte:

```
switch(condicao) {

     case opcao_1:
          comando_1;
          comando_2;
          comando_3;
          break;

     case opcao_2:
          comando_4;
          break;

     case opcao_3:
          comando_5;
          comando_6;
          break;

     default:
          comando_7;

} // fim do switch
```

O parâmetro passado no cabeçalho do switch é normalmente uma variável que irá controlar qual dos blocos deverá ser executado. Ele pode ser um valor inteiro, um caractere ou uma string. Se ele for igual ao valor existente após a palavra case, então o bloco de comando do trecho é executado, caso contrário uma outra opção é avaliada.

A palavra default é opcional, e só é executada caso nenhuma opção seja igual à variável de condição, executando os comandos associados a ela. A palavra reservada break é importante nessa estrutura, para que depois de executados os comandos, a execução saia do switch. A omissão desta palavra implica que todos os comandos abaixo do case executado também sejam executadas. Esse é um dos únicos pontos

em que a palavra break pode ser utilizada sem quebrar as boas práticas de programação estruturada.

Vale destacar que não é necessário colocar as chaves para indicar o início e fim de bloco, ficando isso implícito na construção. Além do mais, podem vir qualquer número de comandos após os dois pontos. Duas opções com os mesmos comandos podem ser facilmente declaradas desde que sejam colocados duas palavras case com as respectivas opções seguidas dos dois pontos.

Segue um exemplo:

```
$op = ‘c’;
switch ($op) {

     case ‘a’:
          $teste = 10;
          echo $teste;
          break;

     case ‘b’:
          $teste = 20;
          echo $teste;
          break;

     case ‘c’:
     case ‘d’:
          $teste = 55;
          echo $teste;
          break;

     default:
          echo “Nenhuma opção é igual à opção”;

} // fim do switch de exemplo
```

A variável $op contém o valor que será avaliado. No primeiro e segundo case é feita a comparação, mas como o resultado é falso, nenhum comando é executado. No terceiro, a comparação retorna verdade e os comandos associados são executados, sendo que ao final, por causa da palavra break, o comando em default não é executado, fazendo com que o fluxo de execução saia do comando switch.

Repetições

Repetição usando while

Comandos de repetição servem para executar um determinado grupo de operações por um número de vezes específico e depois sair deste bloco. Todos os comandos de repetição possuem alguma condição que fazem com que eles saiam da repetição, caso contrário ficariam executando de forma infinita.

No caso do comando while, a condição é avaliada no começo da repetição, sendo assim, caso logo de início a opção seja considerada falsa, nenhuma repetição será executada, passando para o próximo comando depois do while. Sua estrura é a seguinte:

```
while(condicao == verdade)
     comando
```

Um exemplo prático seria:

```
$contador = 0;
while($contador < 10) {

     echo “O valor do contador é: “ . $contador;
     ++$contador;

} // fim do while
```

Neste exemplo, uma variável de controle é iniciada com zero. Como o valor é menor que 10, o argumento é avaliado como verdade e os comandos dentro do bloco (delimitados pelas chaves) são executados, primeiro a impressão e depois o incremento do contador. Esse incremento é muito importante neste exemplo, pois é com ele que se tem a garantia que em algum momento a variável $contador será maior que 10, tornando a condição falsa e saindo da repetição.

Repetição usando for

Outro tipo comum de repetição é o for, que ao contrário do while, já se sabe a priori quantas vezes os comandos associados a ele serão executados, diminuindo as chances de um laço infinito. Sua estrutura é a seguinte:

```
for(comando_inicial; condicao_parada; incremento)
     comando
```

Por exemplo:

```
for($varia = 0; $varia < 10; ++$varia)
     echo “O valor da variável é: “ . $varia;
```

A variável `$varia` é o chamado contador do laço, pois é o item a ser avaliado para saber se o comando deve parar ou não. A primeira parte do bloco serve para inicializar as variáveis necessárias para esse controle (pode existir mais de uma, separadas por vírgulas). O trecho entre os dois ponto e vírgula, é a condição para saída da repetição. Por fim, o trecho final é a maneira como a variável de controle será incrementada.

Repetição usando do/while

A estrutura de um comando `do/while` é bem parecida com o de um laço `while`, com a diferença que a condição de parada só é avaliada no final do bloco, garantindo que ao menos uma vez todos os comandos do bloco sejam executados. A estrutura padrão, seria:

```
do {

     comando;

} while(condicao == verdade);
```

Reescrevendo o comando `while`:

```
$contador = 0;
do {

     echo “O valor do contador é: “ . $contador;
     ++$contador;

} while($contador < 10);
```

Que é bem parecido com o exemplo acima, mudando-se apenas a ordem dos elementos. A diferença é que independente do valor inicialmente colocado em `$contador`, ao menos uma vez seria exibido o seu valor.

Repetição usando foreach

A repetição foreach foi criada para trabalhar de uma forma mais eficiente com as estruturas de dados de vetores, pois fornece uma maneira rápida de acessar os vários campos de um vetor, sem se ter a preocupação na forma como os índices são trabalhados ou acessados por parte do sistema. Sua estrutura é a seguinte:

```
foreach(vetor_dados as $valor)
     comando;
```

Por exemplo, considere o exemplo:

```
$vetor[0] = 10;
$vetor[1] = 11;
$vetor[2] = 12;
$vetor[3] = 13;
$vetor[4] = 14;

foreach($vetor as $unico)
     echo “O valor é “. $unico;
```

Um vetor é uma variável que possui dados em várias posições consecutivas de memória, identificadas com o mesmo nome e que podem ser acessadas através de um índice inteiro. No exemplo, nosso vetor possui cinco valores, com o índice inicial começando em zero (por padrão). O comando foreach pega cada um dos valores e o coloca na variável $unico, terminando a repetição quando todos os itens do vetor tenham sido manipulados.

Desvios Incondicionais

Desvio incondicional: `return`

O comando `return` é muito utilizado em funções e serve para encerrar a execução da mesma e retornar para o trecho de código que chamou o trecho inicialmente. Ele é muito útil quando o processamento chegou a um ponto que não pode evoluir mais e deseja-se que o controle retorne para a origem, podendo por isso ter vários comandos desse tipo num código.

Desvio incondicional: `break`

Este comando possui várias aplicações, sendo uma delas o uso dentro de condicionais `switch`, para limitar o número de operações executadas ou então dentro de comandos de repetição, para fazer com que a saída seja feita independente da condição de saída da repetição tenha sido concluída, passando para o próximo comando fora do laço. Por exemplo:

```
for($i = 0; $i < 100; ++$i) {

     if($i == 20)
          break;

     echo $i;

}
```

Neste trecho, o laço fará vinte repetições, imprimindo o valor de `$i`, porém ao chegar no valor 20, a condição se tornará verdadeira e o comando associado, `break`, será executado, fazendo com que o comando `for` seja encerrado. Existe a possibilidade de associar um valor inteiro ao lado do `break`, para o caso de laços aninhados, indicando o número de repetições que devem ser puladas, o padrão é de apenas um. Por exemplo:

```
for($i = 0; $i < 20; ++$i) {

     for($j = 0; $j < 20; ++$j) {

          if($i == 6)
               break 2;

          echo $j;
```

```
    }

    echo $i;

}
```

Neste exemplo, sempre que o valor de `$i` for igual a 6, o comando if terá sua condição igual a verdade e o `break` executado, com o detalhe de que o fluxo de execução será direcionado para fora de ambos os comandos for, diferente do que aconteceria se não tivéssemos parâmetro ao lado do `break`, passando para fora apenas do for mais interno. Outro exemplo:

```
$busca = {10, 11, 12, 13, 14};
$chave = 13;
$encontrou = 0;

for($i = 0; $i < sizeof($busca); ++$i)
    if($busca[$i] == $chave) {
        $encontrou = 1;
        break;
    }
```

Essa não seria a melhor estrutura nesse caso, mas serve bem para ilustrar o que se deseja. Tem-se um vetor com diversas chaves e uma chave em especial para verificar se ela existe ou não no vetor. Deve-se passar por todas as posições, porém, caso ela seja encontrada em alguma posição, `$encontrou` recebe o valor 1 e o resto do processamento da repetição é encerrado.

Desvio incondicional: `continue`

O continue possui uma utilização semelhante ao `break`, com a diferença de que o fluxo de execução não sai da repetição, mas cancela os demais comandos daquela repetição, passando para o próximo passo. O conceito de laços aninhados também se aplica a este comando. Por exemplo, teríamos:

```
for($i = 0; $i < 20; ++$i) {

    if($i % 2 == 0)
        continue;

    echo $i;
```

```
}
```

No exemplo, seriam impressos apenas os números ímpares, pois como o resto de todos os números pares é zero, todos eles teriam a condição verdadeira e o comando continue seria avaliado, passando para a próxima iteração do laço.

Rua Lauro Vannucci, 1020
Jardim Santa Cândida – Campinas – SP
CEP 13087-548

Tel (19) 3256-5433
Fax (19) 3256-5433
conceptia@conceptia.com.br

Declarando uma função

Uma função, como já introduzido acima, é um bloco independente de código, que pode ser chamado em diversos pontos de outros códigos (inclusive por outras funções), eliminando a repetição de código, facilitando a manutenção de sistemas e tornando o sistema um tanto mais modular, permitindo que novas partes possam ser facilmente colocadas.

Para a criação de uma função, é necessária a definição de um nome único, de tal modo que esta função possa ser identificada sem conflitos com quaisquer outras. Pode também vir no seu cabeçalho, zero ou mais parâmetros, sendo que os parâmetros contêm dados que serão manipulados dentro da função para produzir um resultado. E por fim, um comando `return` para devolver algum valor para o local que a chamou inicialmente, sendo contudo, este comando opcional.

```
function nome_da_função($param_1, $param_2, $param_3) {

     return(10);

}
```

Além de tudo que já foi mencionado para uma função, é necessário que antes de tudo seja escrita a palavra `function`, para explicitar que estamos criando uma função. No caso de uma função sem parâmetros, é preciso deixar apenas os parênteses, sem qualquer valor dentro. Tudo englobado entre chaves. Segue um exemplo:

```
function soma_vetor($vetor) {

     $total = 0;

     for($i = 0; $i < sizeof($vetor); ++$i)
          $total += $vetor[$i];

     return($total);

}
```

Neste exemplo, a função `soma_vetor` recebe como parâmetro um vetor (que deve ser de inteiros, caso contrário ocorrerá um erro) e irá pegar cada um dos valores nas posições do vetor e somará com o valor que estiver em `$total`, que inicialmente é zero. Ao final essa

variável será retornada com o total dos valores. Esse trecho poderá, dessa forma, ser utilizada em diversos pontos do código, de tal forma que o reaproveitamento de código possa ser feito. Eis outro exemplo:

```
function reverte($vetor) {

     $novo_vetor;
     $inicio = 0;
     $final = sizeof($vetor) – 1;

     while($inicio < sizeof($vetor)) {

          $novo_vetor[$inicio] = $vetor[$final];
          ++$inicio;
          --$final;

     } // while

     return($novo_vetor);

}
```

Neste exemplo, uma função que irá inverter a ordem dos elementos de um vetor recebe como parâmetro um vetor. Ao final do processamento, uma nova variável irá armazenar a nova ordem dos elementos do vetor, sendo por fim retornado como resultado da função.

Chamando uma função

Pode-se chamar uma função já definida (ou então alguma função interna do sistema) deve-se usar o nome da função e colocar todos os parâmetros necessários para que ela retorne um resultado correto. Caso a função não retorne nenhum valor, nada mais é preciso ser feito, caso contrário, se a função retornar algum dado, é preciso colocar uma variável antes da função. Exemplos:

```
funcao_sem_retorno();
funcao_sem_retorno_mas_com_argumentos($a, $b, $c);

$teste = nova_função();
$outro_teste = funcao_de_teste($a, $x);
```

Nos dois casos iniciais, nenhum valor é retorno, então apenas o nome da função com eventuais parâmetros são necessários. Nos dois casos

seguintes, algum valor é retornado, então é preciso colocar uma variável antes de se chamar a função.

Incluindo Arquivos

Diretiva include

Eventualmente, para projetos muito grandes, é uma boa prática dividir o código gerado em diversos arquivos, de tal forma a tornar a sua manutenção e entendimento mais simples. Porém, ao se fazer essa divisão, é necessário deixar explícito, nos demais códigos, os locais em que se fará uso daqueles códigos colocados em outros arquivos.

A diretiva `include` serve para isso, pois indica o local e nome do arquivo em que um determinado arquivo de código PHP está colocado para que ele possa ser incluído e utilizado normalmente. Essa inclusão significa basicamente que foi feita uma "inserção" de código naquele ponto, mesmo que ele não esteja visível. Exemplos de como fazer isso, seguem:

```
include('biblioteca.php');
include('../funcoes/arq_funcoes.php');
include("diretorio/pasta/arquivo.php");
```

No primeiro caso, o arquivo desejado está no mesmo diretório do arquivo que contêm essa linha. Nos outros dois, existe uma navegação pela estrutura diretórios para localizar o arquivo. Vale destacar que os dois pontos ".." denotam uma navegação para níveis mais baixos na hierarquia. Caso se use essa diretiva e não se encontre o arquivo, é escrita uma advertência na tela para o usuário.

Diretiva require

A diretiva `require` possui praticamente a mesma aplicação do comando include, com a diferença que caso um arquivo não seja encontrado, é escrito um erro para o usuário e o processamento do resto do script encerrado.

Formulários

Enviando dados entre arquivos

Um item que merece destaque à parte quando se trabalha com PHP é a questão do envio e tratamento de dados entre formulários que foram preenchidos pelo usuário e que deve ser feito alguma ação sobre eles de tal forma alcance algum resultado, como a sua inserção num banco de dados, ou então a sua validação ou mesmo o retorno de uma conta.

O formulário em que os dados são inseridos pelo usuário são HTML puro, porém a lógica de programação que são tratados, é totalmente comandada pelo PHP, seja com suas funções pré-definidas, seja com seus comandos nativos. Um formulário pode ser enviado para ser tratado por um outro arquivo PHP ou então por ele mesmo, desde que tenha ocorrido uma preocupação em organizar o código de uma maneira a facilitar isso.

Cabeçalho de um formulário

Um formulário HTML possui um cabeçalho padrão e tags de campos que devem vir englobadas dentro desta tag. Neste cabeçalho, vêm informações de configuração e de que forma o formulário deverá ser tratado quando o usuário submeter essas informações ao servidor. Um exemplo de código de formulário seria:

```
<form name=”form_1” method=”POST” action=”trata.php”>

     <input name=”nome” type=”text” maxlength=”50”>
     <input name=”cpf” type=”text” maxlength=”10”>
     <input name=”telefone” type=”text” maxlength=”10”>
     <input name=”sub” type=”submit”>

</form>
```

Neste exemplo, existem três campos para o usuário preencher (nome, cpf e telefone) e um botão de submissão, para enviar esses dados ao servidor. No cabeçalho da tag form existem vários dados importantes. Primeiro o seu nome, para futura referência. Depois o método como os dados serão enviados, pode ser POST (ocultos para o usuário) ou GET (visíveis ao usuário pela barra de endereços) e o arquivo que irá tratar esses dados, no parâmetro action.

Uma vez que este formulário seja submetido para o servidor, o arquivo que foi definido no seu cabeçalho, neste caso trata.php, é invocado e estes campos são repassados para que este arquivo manipule os dados da forma desejada, como por exemplo, montando uma cadeia

de caracteres para salvar os dados num banco de dados para que as informações não sejam perdidas com o tempo.

Manuseando as informações passadas

Existem diversas maneiras de se obter as informações que foram passadas de um arquivo para outro a partir de um formulário de submissão. O mais simples dele é sabendo o nome dos campos que foram utilizados no formulário original. Todos os nomes de campos que foram criados num formulário possuem a sua respectiva variável quando submetidos ao servidor. Sendo assim, teríamos:

```
echo $nome;
echo $cpf;
echo $telefone;
```

Com os comandos acima, escritos no arquivo trata.php, seriam impressos o conteúdo dos campos digitados pelo usuário e que logo em seguida foram submetidos para o servidor. Dessa forma, diversas ações poderiam ser realizadas pelo desenvolvedor, como por exemplo, validar as informações passadas, garatindo que o nome e o cpf sejam válidos.

Métodos $_GET e $_POST

Existem contudo formas alternativas e que garantem que as informações passadas sejam realmente obtidas pelo desenvolvedor. Esse tipo de abordagem é importante em conceitos mais avançados, como seções e cookies, onde variáveis são repassadas para diversos arquivos e mais que isso, para o mesmo arquivo diversas vezes.

O PHP fornece duas variáveis de sistema para se trabalhar com as informações passadas de um arquivo para outro: `$_GET` e `$_POST`. Todos dados enviados entre arquivos são tratados como um vetor, podendo ser acessadas pelas variáveis acima, desde que se saiba os nomes originais das variáveis no primeiro formulário. Dessa forma, teríamos:

```
$n = $_POST['nome'];
$c = $_POST['cpf'];
$t = $_POST['telefone'];
```

Em cada um dos casos, a variável à esquerda do sinal de igual irá receber o conteúdo digitado pelo usuário no respectivo campo do formulário, podendo desta forma trata-lo da forma que achar mais

interessante. Foi utilizado $_POST porque o formulário faz uso do método post em seu cabeçalho.

Rua Lauro Vannucci, 1020
Jardim Santa Cândida – Campinas – SP
CEP 13087-548

Tel (19) 3256-5433
Fax (19) 3256-5433
conceptia@conceptia.com.br

Funções de Acesso a Bancos de Dados

PHP fornece diversas funções de acesso ao banco de dados, a maioria delas com o mesmo objetivo e resultado obtido, mas que possuem nomes diferentes por acessarem bancos de dados diferentes, o que indica que a implementação das funções, que é transparente para o desenvolvedor, difere de um banco de dados para outro.

Dessa forma, para conectar a um banco, cada um tem a sua própria função de conexão, com nomes diferentes, mesmo que o resultado seja o mesmo. Um novo conceito que os projetos estão implementando é a de separação em camadas, fazendo com que problemas específicos de consistência de informações fiquem restritos a um dado ponto e não se tenha que mudar todo o código caso alguma mudança seja necessária.

Conectando com um banco de dados

Um dos grandes diferenciais que a linguagem PHP possui sobre diversas outras é o seu suporte nativo na interação com Bancos de Dados, o que torna o acesso às bases muito simples e muito mais eficientes. Para a construção de um sistema dinâmico (que pode mudar com o tempo e se atualizar com o tempo, sem a necessidade de se mexer em código), trabalhar com bancos de dados é fundamental.

Para acessar um banco de dados, antes de tudo é preciso executar um comando que faça uma conexão e deixe claro para os arquivos PHP qual Banco de Dados será acessado, em qual servidor e utilizando qual usuário (e eventualmente, qual senha). Esse tipo de operação deve ser realizada em todos os arquivos que irão fazer acesso ao Banco de Dados, daí ser interessante escreve-lo num arquivo em separado e incluí-lo nos arquivos.

O comando para realizar essa conexão segue o padrão:

```
pg_connect(“string_de_conexao”);
```

Retornando um recurso que pode ser utilizado em outras funções PHP. Porém, simplesmente executando esse comando, já se obtém a conexão com o Banco de Dados, permitindo que comandos SQL sejam executados. O parâmetro `string_de_conexao` deve conter os seguintes parâmetros:

Parâmet	Descrição

ro	
host	Nome do servidor onde o Banco se encontra
dbname	Nome do banco de dados
user	Nome do usuário para acessar o Banco
password	Senha deste usuário para acesso

Podemos armazenar essas informações numa variável, ficando o resultado como algo do tipo:

```
$c = "host=nome_h dbname=nome_b user=bdr password=pass";
$conn = pg_connect($c);
```

Os valores depois do sinal de igual devem ser modificados para os valores reais de conexão para o banco de dados em questão. Todos os arquivos que tiverem essas linhas (sejam explícitas ou num arquivo em separado), estarão prontos para realizar operações no banco de dados através de funções pré-definidas.

Vale ressaltar ainda que as funções utilizadas aqui são para o banco de dados postgreSQL, daí possuírem um prefixo "pg_" no início de seu nome. Outros bancos de dados possuem funções similares, porém com prefixos diferentes dependendo de cada caso. O PHP oferece hoje suporte a uma imensa quantidade de Bancos de Dados, como Oracle, MS-SQLServer e MySQL.

Criando e executando uma consulta de dados

Uma vez estabelecida uma conexão com o Banco de Dados, temos a possibilidade de inserir, remover, alterar e consultar dados existentes no sistema. Para isso precisamos ter conhecimento das tabelas existentes no banco de dados, seus campos e definições, caso contrário o trabalho de manusear essas informações seriam muito mais complexas e demoradas.

Uma operação básica quando se trabalha com o PHP acessando um Banco de Dados é a questão de consulta das informações de uma tabela e a exibição desse conteúdo na tela, para o usuário. Para mostrar como se realiza essa operação, considere uma definição de tabela como a que segue.

```
Nome da Tabela:          Pessoa
Atributos da Tabela:     Nome
                         CPF
```

Telefone

No exemplo acima, a tabela tem o nome "Pessoa" e três atributos que irão guarda as informações, o nome da pessoa, o seu número de CPF e o seu telefone. Nessa tabela pode existir um grande número de registros, representados pelas linhas da tabela e que devem ser únicas, para que os dados possam ser acessados sem possibilidade de erro.

Para realizar o acesso a Bancos de Dados usa-se a linguagem SQL ("Structured Query Language"), sendo que seus comandos devem ser escritos na forma de uma String e executadas através de uma função do PHP. É uma boa prática de programação armazenar os comandos dentro de uma variável para só depois executar o comando pela função. Como exemplo teríamos:

```
$query = "SELECT * FROM Pessoa";
$resultado = pg_exec($query);
```

A variável $query armazena uma String com a forma que o SQL usa para acessar todos os dados de uma determinada tabela, no caso, Pessoa. A função pg_exec recebe como parâmetro o comando SQL a ser executado e o manda para o servidor (cuja conexão foi anteriormente estabelecida) e aguarda o resultado. Esse resultado é armazenado em $resultado.

Poderíamos ter colocado no lugar do SELECT acima, qualquer tipo de comando SQL, desde para inserir ou atualizar dados, como para remover os dados permanentemente do sistema. Por isso, todo o cuidado é pouco quando se escreve comandos deste tipo.

Tratando o retorno de uma consulta

Utilizando o exemplo acima, vamos agora analisar como o retorno dos dados é tratado. Só que antes, para facilitar a nossa visualização do exemplo, considere que a tabela em questão acima, tenha os seguintes registros armazenados:

Nome	CPF	Telefone
José da Silva	12.258.789 – 12	19 4587 3256
Manuel da Costa	36.654.951 – 25	19 3269 5896
Maria de Oliveira	17.698.321 – 63	19 9858 7412

A consulta montada acima, deveria retornar todos esses três registros exibidos na tabela acima, porém, a forma como isso realiza não é tão intuitiva. O valor armazenado é um ponteiro para os dados retornados. De uma certa forma, essas informações são vistas como uma matriz. Para acessar cada um dos registros, deve-se fazer como no exemplo abaixo:

```
$linha = pg_fetch_array($resultado, 0);
```

Sendo que a variável `$linha` irá armazenar o registro que está na posição 0. Além disso, essa variável é considerada uma vetor, com seus parâmetros sendo acessados desta forma. No exemplo da primeira linha, para imprimir o nome da pessoa, teríamos que escrever algo do tipo:

```
echo “Nome da Pessoa: “ . $linha[0];
```

Como os vetores começam na posição zero, o primeiro índice a ser utilizado deve ser esse. Operações similares para imprimir os demais campos devem ser escritos como seguem:

```
echo “CPF: “ . $linha[1] . “ e Telefone: “ . $linha[2];
```

Eventualmente, queremos mostrar todos os registros recuperados. Isso de vê ser feito utilizando um laço de repetição. Por exemplo, utilizando um laço `for`:

```
$tamanho = pg_num_rows($resultado);
for($i = 0; $i < $tamanho; ++$i) {

     $linha = pg_fetch_array($resultado, $i);
     echo “Nome: “ . $linha[0];
     echo “, CPF: “ . $linha[1];
     echo “, Tel.: “ . $linha[2] . “<br>;

}
```

Utilizando-se o exemplo da tabela acima, seriam impressos os três nomes, um por linha (repare no uso da tag HTML para quebra de linha) e depois o laço seria encerrado. Vale destacar ainda que a função `pg_num_rows` devolve o número de registros feitos pela consulta. Caso fosse zero, a repetição não teria sido executada.

Referência

http://www.php.net
Referência para programação em PHP. Possui bibliotecas de todas as funções disponível em PHP. Parte de download de documentação de manuais e de ferramentas para programação em PHP. Muitos exemplos, principalmente de usuários. Muito bom também como referência para as bibliotecas de funções a bancos de dados.

http://www.imaster.com.br
Site com diversas colunas nas mais variadas tecnologias de informação. Destaque especial para a coluna sobre PHP, pois mostra diversos exemplos de aplicação e casos reais de programação em PHP, além de falar de diversas nuances da linguagem, explicando como tirar o máximo proveito do recurso.

http://www.apache.org
Local onde grandes projetos de software livre estão hospedados e podem ser obtidos por download sem nenhum tipo de custo. Apenas para citar alguns dos projetos existentes, existem o Servidor Web Apache, o Servidor de Aplicação Java Tom Cat e o Ambiente de Desenvolvimento de JSP, Struts.