PHP

Programando com Orientação a Objetos

Pablo Dall'Oglio

Novatec

Editor: Rubens Prates
Assistente editorial: Priscila A. Yoshimatsu
Editoração eletrônica: Carolina Kuwabata
Revisão gramatical: Jussara Rodrigues Gomes
Capa: Pablo Dall'Oglio e Rodolpho Lopes

ISBN: 978-85-7522-465-6

Histórico de impressões:

Novembro/2015	Terceira edição
Abril/2009	Segunda edição (ISBN: 978-85-7522-200-3)
Setembro/2007	Primeira edição (ISBN: 978-85-7522-137-2)

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
Email: novatec@novatec.com.br
Site: www.novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec
LinkedIn: linkedin.com/in/novatec

CAPÍTULO 1

Introdução ao PHP

A vida é uma peça de teatro que não permite ensaios...
Por isso, cante, ria, dance, chore e viva intensamente cada momento de sua vida,
antes que a cortina se feche e a peça termine sem aplausos...

Charles Chaplin

Ao longo deste livro utilizaremos diversas funções, comandos e estruturas de controle básicos da linguagem PHP, que apresentaremos neste capítulo. Conheceremos as estruturas básicas da linguagem, suas variáveis e seus operadores e também um conjunto de funções para manipulação de strings, arquivos, arrays, bancos de dados, entre outros.

1.1 O que é o PHP?

A linguagem de programação PHP, que no início significava Personal Home Page Tools, foi criada no outono de 1994 por Rasmus Lerdorf. Essa linguagem era formada por um conjunto de scripts escritos em linguagem C, voltados à criação de páginas dinâmicas que Rasmus utilizava para monitorar o acesso ao seu currículo na internet. Com o tempo, mais pessoas passaram a utilizá-la e Rasmus adicionou vários recursos, como a interação com bancos de dados. Em 1995, o código-fonte do PHP foi liberado, e com isso mais desenvolvedores puderam se juntar ao projeto. Naquela época, por um breve período de tempo, o PHP foi chamado de FI (Forms Interpreter).

O PHP passou por várias reescritas de código ao longo do tempo e nunca parou de conquistar novos adeptos. Uma segunda versão foi lançada em novembro de 1997, sob o nome PHP/FI 2.0. Naquele momento, aproximadamente 60 mil domínios, ou 1% da internet, já utilizavam PHP, que era mantido principalmente por Rasmus.

No mesmo ano Andi Gutmans e Zeev Suraski, dois estudantes que utilizavam essa linguagem em um projeto acadêmico de comércio eletrônico, resolveram cooperar com Rasmus para aprimorar o PHP. Para tanto, reescreveram todo o código-fonte, com base no PHP/FI 2, dando início assim ao PHP 3, disponibilizado oficialmente em junho de 1998. Entre as principais características do PHP 3 estavam a extensibilidade, a possibilidade de conexão com vários bancos de dados, novos protocolos, uma sintaxe mais consistente, suporte à orientação a objetos e uma nova API, que possibilitava a criação de novos módulos e que acabou por atrair vários desenvolvedores ao PHP. No final de 1998, o PHP já estava presente em cerca de 10% dos domínios da internet. Nessa época o significado da sigla PHP mudou para PHP: Hypertext Preprocessor, retratando, assim, a nova realidade de uma linguagem com propósitos mais amplos.

No inverno de 1998, Zeev e Andi começaram a trabalhar em uma reescrita do núcleo do PHP, tendo em vista melhorar seu desempenho e sua modularidade em aplicações complexas. O nome foi rebatizado para Zend Engine (Zeev + Andi). O PHP 4, já baseado nesse mecanismo, foi lançado em maio de 2000, trazendo melhorias como seções, suporte a diversos servidores web, além da abstração de sua API, permitindo inclusive ser utilizado como linguagem para shell script.

Apesar de todos os esforços, o PHP ainda necessitava de maior suporte à orientação a objetos. Esses recursos foram trazidos pelo PHP 5, após um longo período de desenvolvimento que culminou com sua disponibilização oficial em julho de 2004. Ao longo de mais de uma década, o PHP vem adicionando mais e mais recursos e se consolida ano após ano como uma das linguagens de programação orientadas a objetos que mais crescem no mundo. Estima-se que o PHP seja utilizado em mais de 80% dos servidores web existentes, tornando-a disparadamente a linguagem mais utilizada para desenvolvimento web. Ao longo do livro, veremos esses recursos empregados em exemplos práticos.

Fonte: PHP Group (http://php.net/history.php)

1.2 Um programa PHP

1.2.1 Estrutura do código-fonte

Normalmente um programa PHP tem a extensão *.php*. Entretanto não é incomum encontrarmos extensões como *.class.php* para armazenar classes ou *.inc.php,* em projetos mais antigos, para representar simplesmente arquivos a serem incluídos.

O código de um programa escrito em PHP deve iniciar com `<?php`, veja:

```
<?php
// código;
// código;
// código;
```

Observação: a finalização da maioria dos comandos se dá por ponto e vírgula (;).

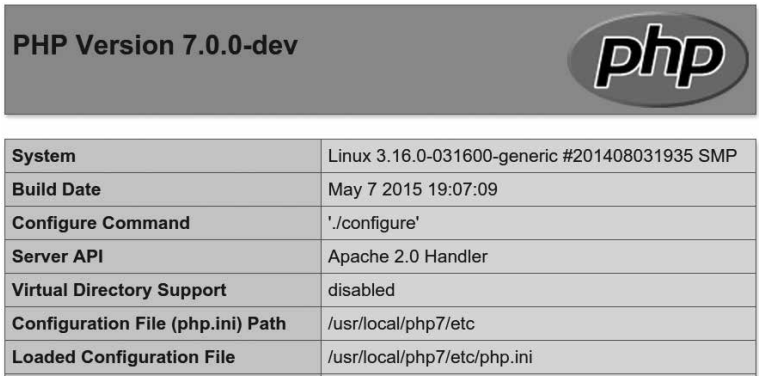
1.2.2 Configuração

Ao iniciarmos com o PHP, é importante sabermos configurar o ambiente. Nesse sentido a função `phpinfo()` é muito importante, pois ela apresenta uma tabela com a configuração atual do PHP, como níveis de erro, extensões instaladas, entre outros.

```
<?php
phpinfo();
```

O arquivo de configuração do PHP é o *php.ini*, cuja localização varia conforme a instalação utilizada. Mas sua localização pode ser descoberta pela função `phpinfo()`, conforme visto na figura 1.1. A partir da localização do *php.ini* podemos realizar algumas configurações iniciais. No exemplo a seguir definimos o `timezone` (usado em funções com cálculo de tempo) habilitando o `display_errors` (para que todos os erros que ocorrerem sejam exibidos), o `log_errors` (para que os erros sejam registrados em um arquivo), definindo o arquivo de registro de erros com o `error_log`, bem como ligando simplesmente todos os níveis de erro, por meio do `error_reporting`. Essas definições são voltadas mais para um ambiente de desenvolvimento. Em produção, geralmente desligamos o `display_errors`.

```
date.timezone = America/Sao_Paulo
display_errors = On
log_errors = On
error_log = /tmp/php_errors.log
error_reporting = E_ALL
```



System	Linux 3.16.0-031600-generic #201408031935 SMP
Build Date	May 7 2015 19:07:09
Configure Command	'./configure'
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/usr/local/php7/etc
Loaded Configuration File	/usr/local/php7/etc/php.ini

Figura 1.1 – Página de configuração.

1.2.3 Comentários

Para comentar uma única linha:

```
// echo "a";
# echo "a";
```

Para comentar muitas linhas:

```
/* echo "a";
   echo "b"; */
```

1.2.4 Comandos de saída (output)

São os comandos utilizados para gerar uma saída em tela (output). Se o programa PHP for executado na linha de comando (prompt do sistema), a saída será no próprio console. No entanto se o programa for executado via servidor de páginas web (Apache, Nginx), a saída será exibida na própria página HTML.

echo

É um comando que imprime uma ou mais variáveis no console. Exemplo:

```
echo 'a' . '<br>' . PHP_EOL;
echo 'b' . '<br>' . PHP_EOL;
```

Observação: utilizaremos o separador `'<br>'.PHP_EOL` para formarmos uma quebra de linha. `PHP_EOL` representa uma quebra de linha (*end of line*). Ex.: "\n".

💻 **Resultado:**

```
a
b
```

print

É um comando que imprime uma string no console. Exemplo:

```
print 'abc';
```

💻 **Resultado:**

```
abc
```

var_dump

É uma função que imprime o conteúdo de uma variável de forma detalhada, muito comum para se realizar debug. Se o parâmetro for um objeto, ele imprimirá todos os seus atributos; se for um array de várias dimensões, imprimirá todas elas, com seus respectivos conteúdos e tipos de dados. Exemplo:

```
$vetor = array('Palio', 'Gol', 'Fiesta', 'Corsa');
var_dump($vetor);
```

💻 **Resultado:**

```
array(4) {
   [0]=>
   string(5) "Palio"
   [1]=>
   string(3) "Gol"
   [2]=>
   string(6) "Fiesta"
   [3]=>
   string(5) "Corsa"
}
```

print_r

É uma função que imprime o conteúdo de uma variável de forma detalhada, assim como a `var_dump()`, mas em um formato mais legível para o programador, com os conteúdos alinhados e suprimindo os tipos de dados. Exemplo:

```
$vetor = array('Palio', 'Gol', 'Fiesta', 'Corsa');
print_r($vetor);
```

💻 **Resultado:**

```
Array
(
    [0] => Palio
    [1] => Gol
    [2] => Fiesta
    [3] => Corsa
)
```

1.3 Variáveis

Variáveis são identificadores utilizados para representar valores mutáveis e voláteis, que só existem durante a execução do programa. Elas são armazenadas na memória RAM e seu conteúdo é destruído após a execução do programa. Para criar uma variável em PHP, precisamos atribuir-lhe um nome de identificação, sempre precedido pelo caractere cifrão ($). Veja os exemplos a seguir:

```
<?php
$nome = "João";
$sobrenome = "da Silva";
echo "$sobrenome, $nome";
```

💻 **Resultado:**

```
da Silva, João
```

Algumas dicas:

- Nunca inicie a nomenclatura de variáveis com números.
- Nunca utilize espaços em branco no meio do identificador da variável.
- Nunca utilize caracteres especiais (! @ # % ^ & * / | [ ] { }) na nomenclatura das variáveis.
- Evite criar variáveis com mais de 15 caracteres em virtude da clareza do código-fonte.
- Nomes de variáveis devem ser significativos e transmitir a ideia de seu conteúdo dentro do contexto no qual a variável está inserida.
- Utilize preferencialmente palavras em letras minúsculas separadas por "_" ou somente as primeiras letras em maiúsculas quando da ocorrência de mais palavras.

```
<?php
$codigo_cliente  // exemplo de variável
$codigoCliente  // exemplo de variável
```

Com exceção de nomes de classes e funções, o PHP é *case sensitive*, ou seja, é sensível a letras maiúsculas e minúsculas. Tome cuidado ao declarar variáveis. Por exemplo, a variável `$codigo` é tratada de forma totalmente diferente da variável `$Codigo`.

Em alguns casos, precisamos ter em nosso código-fonte nomes de variáveis que podem mudar de acordo com determinada situação. Neste caso não só o conteúdo da variável é mutável, mas também seu nome. Para isso o PHP implementa o conceito de variáveis variantes (*variable variables*). Sempre que utilizarmos dois sinais de cifrão (`$$`) precedendo o nome de uma variável, o PHP irá referenciar uma variável representada pelo conteúdo da primeira. Neste exemplo utilizamos esse recurso quando declaramos a variável `$nome` (conteúdo de `$variavel`) contendo `'maria'`.

```
<?php
// define o nome da variável
$variavel = 'nome';

// cria variável identificada pelo conteúdo de $variavel
$$variavel = 'maria';

// exibe variável $nome na tela
echo $nome; // resultado = maria
```

Quando uma variável é atribuída a outra, é criada uma nova área de armazenamento na memória (exceto quando a variável representar um objeto). Veja neste exemplo que, apesar de `$b` receber o mesmo conteúdo de `$a`, após qualquer modificação em `$b`, `$a` continua com o mesmo valor.

```
<?php
$a = 5;
$b = $a;
$b = 10;
echo $a;  // resultado = 5
echo ' '; // espaço
echo $b;  // resultado = 10
```

Para criar referência entre variáveis, ou seja, duas variáveis apontando para a mesma região da memória, a atribuição deve ser precedida pelo operador `&`. Assim, qualquer alteração em qualquer uma das variáveis reflete na outra.

```
<?php
$a = 5;
$b = &$a;
$b = 10;
echo $a;  // resultado = 10
echo ' '; // espaço
echo $b;  // resultado = 10
```

Objetos são sempre copiados por referência, independentemente de utilizarmos o operador `&` na operação de atribuição. Objetos planos podem ser criados no PHP a partir da classe `stdClass`. Neste exemplo demonstramos que objetos são copiados por referência. Assim alterações em um objeto implicam em alterações em sua réplica.

```
<?php
$a = new stdClass;   // cria objeto
$a->nome = 'Maria';  // define atributo
$b = $a;  // cria réplica
$b->nome = 'Joana'; // define atributo
print $a->nome;  // resultado = Joana
echo ' '; // espaço
print $b->nome;   // resultado = Joana
```

1.3.1 Tipo booleano

Um booleano expressa um valor lógico que pode ser verdadeiro ou falso. Para especificar um valor booleano, utilize as palavras-chave `TRUE` ou `FALSE`. No exemplo a seguir declaramos a variável booleana `$exibir_nome`, cujo conteúdo é `TRUE` (verdadeiro). Em seguida testaremos o conteúdo dessa variável para verificar se ela é realmente verdadeira. Em caso positivo, será exibido na tela o nome "José da Silva".

```
<?php
$exibir_nome = TRUE;  // declara variável com valor TRUE

// testa se $exibir_nome é TRUE
if ($exibir_nome) {
   echo 'José da Silva';
}
```

Resultado:

```
José da Silva
```

No programa que segue criamos uma variável numérica contendo o valor 91. Em seguida testamos se a variável é maior que 90. Tal comparação também retorna um valor booleano (`TRUE` ou `FALSE`). O conteúdo da variável `$vai_chover` é um booleano que será testado logo em seguida para a impressão da string "Vai chover".

```
<?php
$umidade = 91;  // declara variável numérica

// testa se é maior que 90. Retorna um booleano
$vai_chover = ($umidade > 90);

// testa se $vai_chover é verdadeiro
if ($vai_chover) {
   echo 'Vai chover';
}
```

💻 **Resultado:**

```
Vai chover
```

Também são considerados valores falsos em comparações booleanas:

- Inteiro 0.
- Ponto flutuante 0.0.
- Uma string vazia "" ou "0".
- Um array vazio.
- Um objeto sem elementos.
- Tipo `NULL`.

1.3.2 Tipo numérico

Números podem ser especificados em notação decimal (base 10), hexadecimal (base 16) ou octal (base 8), sendo opcionalmente precedidos de sinal.

```
<?php
$a = 1234;  // número decimal
$a = -123;  // um número negativo
$a = 0123;  // número octal (equivalente a 83 em decimal)
$a = 0x1A;  // número hexadecimal (equivalente a 26 em decimal)
$a = 1.234;  // ponto flutuante
$a = 4e23;  // notação científica
```

1.3.3 Tipo string

Uma string é uma cadeia de caracteres alfanuméricos. Para declará-la, podemos utilizar aspas simples (' ') ou aspas duplas (" "). Veja com mais detalhes como manipular strings na seção 1.10, "Manipulação de strings".

```
<?php
$variavel = 'Isto é um teste';
$variavel = "Isto é um teste";
```

1.3.4 Tipo array

Array é uma lista de valores armazenados na memória que podem ser de tipos diferentes (números, strings, objetos) e podem ser acessados a qualquer momento, pois cada valor é relacionado a uma chave. Um array também pode crescer dinamicamente com a adição de novos itens. Veja na seção 1.11, "Manipulação de arrays", como manipular arrays.

```
<?php
$carros = array('Palio', 'Corsa', 'Gol');
echo $carros[1];  // resultado = 'Corsa'
```

1.3.5 Tipo objeto

Um objeto é uma entidade com um determinado comportamento definido por seus métodos (ações) e propriedades (dados). Para criar um objeto deve-se utilizar o operador `new`. Neste exemplo criamos um objeto plano (`stdClass`) e atribuimos algumas propriedades para ele. No capítulo 2 veremos como manipular classes e objetos.

```
<?php
$carro = new stdClass;
$carro->modelo = 'Palio';
$carro->ano    = 2002;
$carro->cor    = 'Azul';

print_r($carro);
print $carro->modelo . ' ';
print $carro->ano . ' ';
print $carro->cor . ' ';
```

Resultado:

```
stdClass Object
(
    [modelo] => Palio
    [ano] => 2002
    [cor] => Azul
)
Palio 2002 Azul
```

1.3.6 Tipo recurso

Recurso (`resource`) é uma variável que mantém uma referência de recurso externo. Recursos são criados e utilizados por funções como aquelas que criam conexões de banco de dados. As funções `mysql_connect()` e `pg_connect()`, por exemplo, ao conectarem-se ao banco de dados, retornam uma variável de referência do tipo recurso.

```
resource mysql_connect ([ string $server [, string $username [, string $password
    [, bool $new_link [, int $client_flags ]]]]] )
resource pg_connect ( string $connection_string )
```

Observação: uma variável do tipo recurso não pode ser serializada.

1.3.7 Tipo misto

O tipo misto (`mixed`) é uma identificação que representa diversos (não necessariamente todos) tipos de dados que podem ser usados em um mesmo parâmetro. Um parâmetro do tipo `mixed` indica que a função aceita diversos tipos de dados como parâmetro. Um exemplo é a função `gettype()`, quel obtém o tipo da variável passada como parâmetro (que pode ser integer, string, array, objeto, entre outros).

```
string gettype (mixed var)
```

Veja alguns resultados possíveis:

```
"boolean"  "integer"  "double"  "string"  "array"  "object"  "resource"  "NULL"
```

1.3.8 Tipo callback

Algumas funções como `call_user_func()` aceitam um parâmetro que significa uma função a ser executada. Esse tipo de dado é chamado de `callback`. Um parâmetro

callback pode ser o nome de uma função representada por uma string ou o método de um objeto a ser executado representado por um array. Veja os exemplos na documentação da função call_user_func().

1.3.9 Tipo NULL

A utilização do valor especial NULL significa que a variável não tem valor. NULL é o único valor possível do tipo NULL.

1.4 Constantes

Uma constante é um valor que não sofre modificações durante a execução do programa. Ela é representada por um identificador, assim como as variáveis, com a exceção de que só pode conter valores escalares (booleano, inteiro, ponto flutuante e string) ou arrays. As regras de nomenclatura de constantes seguem as mesmas regras das variáveis, com a exceção de que as constantes não são precedidas pelo sinal de cifrão ($) e geralmente utilizam nomes em letras maiúsculas.

```
MAXIMO_CLIENTES  // exemplo de constante
```

Você pode definir uma constante utilizando a função define(). Quando uma constante é definida, ela não pode mais ser modificada ou anulada. Exemplo:

```
<?php
define('MAXIMO_CLIENTES', 100);
echo MAXIMO_CLIENTES;
```

Resultado:

```
100
```

1.5 Operadores

1.5.1 Atribuição

Um operador de atribuição é utilizado para definir o valor de uma variável. O operador básico de atribuição é =, mas outros operadores podem ser utilizados:

```
<?php
$var = 100;
```

```
$var += 5;  // Soma 5 em $var;
$var -= 5;  // Subtrai 5 em $var;
$var *=5;   // Multiplica $var por 5;
$var /= 5;  // Divide $var por 5;
print $var; // resultado: 100
```

Operadores	Descrição
`++$var`	Pré-incremento. Incrementa `$a` em um e, então, retorna `$a`.
`$var++`	Pós-incremento. Retorna `$a` e, então, incrementa `$a` em um.
`--$var`	Pré-decremento. Decrementa `$a` em um e, então, retorna `$a`.
`$var--`	Pós-decremento. Retorna `$a` e, então, decrementa `$a` em um.

```
<?php
$var = 100;
print $var++ . ' '; // retorna 100 e incrementa para 101
print ++$var . ' '; // incrementa para 102 e retorna
print $var-- . ' '; // retorna 102 e decrementa para 101
print --$var . ' '; // decrementa para 100 e retorna
```

Resultado:

```
100  102  102  100
```

1.5.2 Aritméticos

Operadores aritméticos são utilizados para realização de cálculos matemáticos.

Operadores	Descrição
`+`	Adição.
`-`	Subtração.
`*`	Multiplicação.
`/`	Divisão.
`%`	Módulo (resto da divisão).

Em cálculos complexos, procure utilizar parênteses, sempre observando as prioridades aritméticas. Por exemplo:

```
<?php
$a = 2;
$b = 4;
echo $a+3*4+5*$b . ' ';  // resultado = 34
echo ($a+3)*4+(5*$b) . ' ';  // resultado = 40
```

O PHP realiza automaticamente a conversão de tipos em operações:

```
<?php
// declaração de uma string contendo 10
$a = '10';
// soma + 5
echo $a + 5;
```

Resultado:

```
15
```

1.5.3 Relacionais

Operadores relacionais são utilizados para realizar comparações entre valores ou expressões, resultando sempre um valor boolean (TRUE ou FALSE).

Comparadores	Descrição
==	Igual. Resulta verdadeiro (TRUE) se as expressões forem iguais.
===	Idêntico. Resulta verdadeiro (TRUE) se as expressões forem iguais e do mesmo tipo de dados.
!= ou <>	Diferente. Resulta verdadeiro (TRUE) se as variáveis forem diferentes.
<	Menor.
>	Maior que.
<=	Menor ou igual.
>=	Maior ou igual.

Veja a seguir alguns testes lógicos e seus respectivos resultados. No primeiro caso vemos a utilização errada do operador de atribuição = para realizar uma comparação: o operador sempre retorna o valor atribuído.

```
<?php
if ($a = 5) {
   echo 'essa operação atribui 5 à variável $a';
}
```

Resultado:

```
essa operação atribui 5 à variável $a
```

No exemplo que segue declaramos duas variáveis, uma integer e outra string. Neste caso vemos a utilização dos operadores de comparação == e !=.

```
<?php
$a =  1234;
$b = '1234';

if ($a == $b) {
   echo '$a e $b são iguais';
}
else if ($a != $b) {
   echo '$a e $b são diferentes';
}
```

Resultado:

```
$a e $b são iguais
```

No próximo caso além da comparação entre as variáveis, comparamos também os tipos de dados das variáveis.

```
<?php
$c =  1234;
$d = '1234';

if ($c === $d) {
   echo '$c e $d são iguais e do mesmo tipo';
}
if ($c !== $d) {
   echo '$c e $d são de tipos diferentes';
}
```

Resultado:

```
$c e $d são de tipos diferentes
```

O PHP considera o valor zero como sendo falso em comparações lógicas. Para evitar erros semânticos em retorno de valores calculados por funções que podem retornar tanto um valor booleano quanto um inteiro, podemos utilizar as seguintes comparações:

```
<?php
$e = 0;
// testa a variável é FALSE
if ($e == FALSE) {
   echo '$e é falso ';
}
```

```
// testa se a variável é um FALSE e do tipo boolean
if ($e === FALSE) {
   echo '$e é FALSE e do tipo boolean ';
}

// testa se $e é igual a zero e do mesmo tipo que zero
if ($e === 0) {
   echo '$e é zero mesmo ';
}
```

Resultado:

```
$e é falso  $e é zero mesmo
```

1.5.4 Lógicos

Operadores lógicos são utilizados para combinar expressões lógicas entre si, agrupando testes condicionais.

Operador	Descrição
`($a and $b)`	E: Verdadeiro (`TRUE`) se tanto `$a` quanto `$b` forem verdadeiros.
`($a or $b)`	OU: Verdadeiro (`TRUE`) se `$a` ou `$b` forem verdadeiros.
`($a xor $b)`	XOR: Verdadeiro (`TRUE`) se `$a` ou `$b` forem verdadeiros, de forma exclusiva.
`(! $a)`	NOT: Verdadeiro (`TRUE`) se `$a` for `FALSE`.
`($a && $b)`	E: Verdadeiro (`TRUE`) se tanto `$a` quanto `$b` forem verdadeiros.
`($a || $b)`	OU: Verdadeiro (`TRUE`) se `$a` ou `$b` forem verdadeiros.

Observação: `or` e `and` têm precedência menor que `&&` ou `||`.

No programa a seguir, se as variáveis `$vai_chover` e `$esta_frio` forem verdadeiras ao mesmo tempo, será impresso no console "Não vou sair de casa".

```
<?php
$vai_chover = TRUE;
$esta_frio  = TRUE;

if ($vai_chover and $esta_frio) {
   echo "Não vou sair de casa";
}
```

Resultado:

```
Não vou sair de casa
```

Já neste outro programa, caso uma variável seja TRUE e a outra seja FALSE (OU exclusivo), será impresso no console "Vou pensar duas vezes antes de sair".

```
<?php
$vai_chover = FALSE;
$esta_frio  = TRUE;

if ($vai_chover xor $esta_frio) {
   echo "Vou pensar duas vezes antes de sair";
}
```

Resultado:

```
Vou pensar duas vezes antes de sair
```

1.6 Estruturas de controle

1.6.1 IF

O IF é uma estrutura de controle que introduz um desvio condicional, ou seja, um desvio na execução natural do programa. Caso a condição dada pela expressão seja satisfeita, então serão executadas as instruções do bloco de comandos. Caso a condição não seja satisfeita, o bloco de comandos será simplesmente ignorado. O comando IF pode ser lido como "SE (*expressão*) ENTÃO { *comandos*... }".

ELSE é utilizado para indicar um novo bloco de comandos delimitado por { }, caso a condição do IF não seja satisfeita. Pode ser lido como "caso contrário". A utilização do ELSE é opcional.

Veja na figura 1.2 um fluxograma explicando o funcionamento do comando IF.

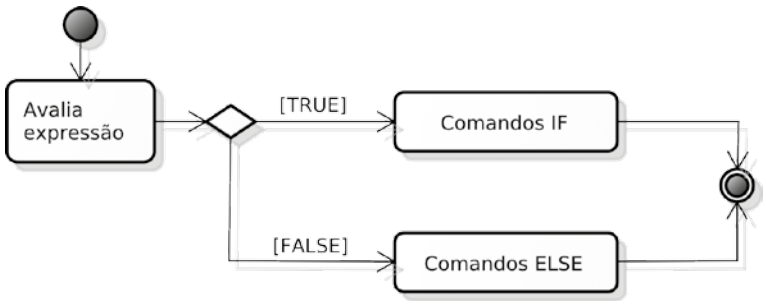


Figura 1.2 – Fluxo do comando IF.

Caso a avaliação da expressão seja verdadeira, o programa parte para a execução de um bloco de comandos; caso seja falsa, parte para a execução do bloco de comandos dada pelo ELSE.

```
if (expressão) {
   // comandos se expressão é verdadeira;
}
else {
   // comandos se expressão é falsa;
}
```

Exemplo:

```
<?php
$a = 1;
if ($a==5) {
   echo "é igual";
}
else {
   echo "não é igual";
}
```

Resultado:

```
não é igual
```

Quando não explicitamos o operador lógico em testes por meio do IF, o comportamento-padrão do PHP é retornar TRUE sempre que a variável tiver conteúdo válido.

```
<?php
$a = 'conteúdo';
if ($a) {
   echo '$a tem conteúdo';
}
if ($b) {
   echo '$b tem conteúdo';
}
```

Resultado:

```
$a tem conteúdo
```

Observação: atualmente se o PHP estiver com o nível de erro NOTICE ligado, o teste `if ($b)` emitirá a mensagem de erro "Notice: Undefined variable: b". A forma mais correta para testar se uma variável está definida é utilizar a função `if (isset($b))`.

Para realizar testes encadeados, basta colocar um IF dentro do outro ou utilizar o operador AND da seguinte forma:

```
<?php
$salario = 1020;
$tempo_servico = 12;
$tem_reclamacoes = false;
if ($salario > 1000) {
    if ($tempo_servico >= 12)  {
        if ($tem_reclamacoes != true) {
            echo 'parabéns, você foi promovido<br>' . PHP_EOL;
        }
    }
}
if (($salario > 1000) and ($tempo_servico >= 12) and ($tem_reclamacoes != true)) {
   echo 'parabéns, você foi promovido<br>' . PHP_EOL;
}
```

Observação: quando os exemplos forem executados no ambiente web, é importante utilizarmos o marcador
 para quebrar as linhas. De outra forma, quando o PHP for executado pela linha de comando, tal marcador será desnecessário.

Resultado:

```
parabéns, você foi promovido
parabéns, você foi promovido
```

O PHP nos oferece facilidades quando desejamos realizar tarefas simples, como realizar uma atribuição condicional a uma variável. A seguir você confere um código tradicional que verifica o estado de uma variável antes de atribuir o resultado.

```
if ($valor_venda > 100) {
   $resultado = 'muito caro';
}
else {
   $resultado = 'pode comprar';
}
```

O mesmo código poderia ser escrito em uma única linha da seguinte forma:

```
$resultado = ($valor_venda > 100) ? 'muito caro' : 'pode comprar';
```

A primeira expressão é a condição a ser avaliada; a segunda é o valor atribuído caso ela seja verdadeira; e a terceira é o valor atribuído caso ela seja falsa.

1.6.2 WHILE

O WHILE é uma estrutura de controle similar ao IF. Da mesma forma, contém uma condição para executar um bloco de comandos. A diferença primordial é que o WHILE estabelece um laço de repetição, ou seja, o bloco de comandos será executado repetitivamente enquanto a condição de entrada dada pela expressão for verdadeira. Esse comando pode ser interpretado como "ENQUANTO (*expressão*) FAÇA {*comandos...* }.".

A figura 1.3 procura explicar o fluxo de funcionamento do comando WHILE. Quando a expressão é avaliada como TRUE, o programa parte para a execução de um bloco de comandos. Ao fim da execução desse bloco de comandos, o programa retorna ao ponto inicial da avaliação e, se a expressão continuar verdadeira, o programa continua também com a execução do bloco de comandos, constituindo um laço de repetições, o qual só é interrompido quando a expressão avaliada retornar um valor falso (FALSE).

```
while (expressão) {
   comandos;
}
```

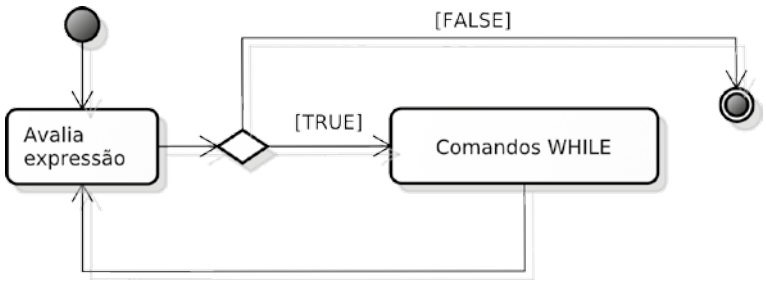


Figura 1.3 – Fluxo do comando WHILE.

No exemplo a seguir, o comando WHILE está avaliando a expressão "se $a é menor que 5" como ponto de entrada do laço de repetições. Na primeira vez que é executada essa comparação é retornado TRUE, visto que o valor de $a é 1. Logo o programa entra no laço de repetições executando os comandos entre { }. Observe que, dentro do bloco de comandos, a variável $a é incrementada. Assim, essa execução perdurará por mais algumas iterações. Quando seu valor for igual a 5, a comparação retornará FALSE e não mais entrará no WHILE, deixando de executar o bloco de comandos.

```
<?php
$a = 1;
while ($a < 5) {
```

```
    print $a;
    $a ++;
}
```

Resultado:

```
1234
```

1.6.3 FOR

O FOR é uma estrutura de controle que estabelece um laço de repetição baseado em um contador; é muito similar ao comando WHILE. O FOR é controlado por um bloco de três comandos que estabelecem uma contagem, ou seja, o bloco de comandos será executado um certo número de vezes.

```
for (expr1; expr2; expr3) {
    comandos
}
```

Parâmetros	Descrição
expr1	Valor inicial da variável contadora.
expr2	Condição de execução. Enquanto TRUE, o bloco de comandos será executado.
expr3	Valor a ser incrementado após cada execução.

Exemplo:

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

Resultado:

```
12345678910
```

Procure utilizar nomes sugestivos para variáveis, mas, em alguns casos específicos, como em contadores, permita-se utilizar variáveis de apenas uma letra, como no exemplo a seguir:

```
<?php
for ( $i = 0; $i < 5; $i++ ) {
    for ( $j = 0; $j < 4; $j++ ) {
        for ( $k = 0; $k < 3; $k++ ) {
```

```
            // comandos...
        }
    }
}
```

Evite laços de repetição com muitos níveis de iteração. Como o próprio Linus Torvalds já disse certa vez, se você está utilizando três níveis encadeados ou mais, considere a possibilidade de revisar a lógica de seu programa.

1.6.4 SWITCH

O comando `switch` é uma estrutura que simula uma bateria de testes sobre uma variável. É similar a uma série de comandos `IF` sobre a mesma expressão. Frequentemente é necessário comparar a mesma variável com valores diferentes e executar uma ação específica em cada um desses casos.

No fluxograma da figura 1.4 vemos que, para cada teste condicional (`CASE`), existe um bloco de comandos a ser executado caso a expressão avaliada retorne verdadeiro (`TRUE`). Caso a expressão retorne falso (`FALSE`), o `switch` parte para a próxima expressão a ser avaliada, até que não tenha mais expressões para avaliar. Caso todas as expressões sejam falsas, o `switch` executará o bloco de códigos representado pelo identificador `default`.

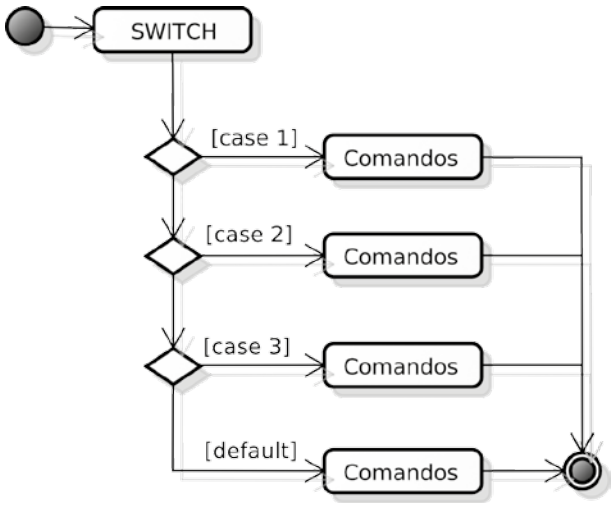


Figura 1.4 – Fluxo do comando `SWITCH`.

Sintaxe do comando:

```
<?php
switch ($variavel) {
    case valor1:
        // comandos
        break;
    case valor2:
        // comandos
        break;
    default:
        // comandos
}
```

Os exemplos seguintes representam duas formas diferentes de se atingir o mesmo resultado. Primeiro, por meio de uma série de comandos IF e, logo em seguida, utilizando a estrutura switch.

Observação: se você tem um switch dentro de um loop e deseja continuar para a próxima iteração do laço de repetição, utilize o comando continue 2, que escapará dois níveis acima.

```
<?php
$i = 1;

if ($i == 0) {
   print "i é igual a 0";
}
elseif ($i == 1) {
   print "i é igual a 1";
}
elseif ($i == 2) {
   print "i é igual a 2";
}
else {
   print "i não é igual a 0, 1 ou 2";
}
```

Resultado:

```
i é igual a 1
```

O switch executa linha por linha até encontrar a ocorrência de break. Por isso a importância do comando break para evitar que os blocos de comando seguintes

sejam executados por engano. A cláusula `default` será executada caso nenhuma das expressões anteriores tenha sido verificada.

```
<?php
$i = 1;
switch ($i) {
    case 0:
        print "i é igual a 0";
        break;
    case 1:
        print "i é igual a 1";
        break;
    case 2:
        print "i é igual a 2";
        break;
    default:
        print "i não é igual a 0, 1 ou 2";
}
```

💻 **Resultado:**

```
i é igual a 1
```

1.6.5 FOREACH

O `foreach` é um laço de repetição para iterações em arrays ou matrizes. É um `FOR` simplificado que decompõe um vetor ou uma matriz em cada um de seus elementos por meio de sua cláusula `AS`.

```
foreach ($array as $valor) {
   instruções
}
```

Exemplo:

```
<?php
$fruta = array("maçã", "laranja", "pera", "banana");
foreach ($fruta as $valor) {
   print "$valor -";
}
```

💻 **Resultado:**

```
maçã - laranja - pera - banana -
```

1.6.6 CONTINUE

A instrução `continue`, quando executada em um bloco de comandos `FOR`/`WHILE`, ignora as instruções restantes até o fechamento em }. Dessa forma o programa segue para a próxima verificação da condição de entrada do laço de repetição.

1.6.7 BREAK

O comando `break` aborta a execução de blocos de comandos, como `IF`, `WHILE`, `FOR`. Quando estamos em uma execução com muitos níveis de iteração e desejamos abortar *n* níveis, a sintaxe é a seguinte:

```
while...
    for...
        break <quantidade de níveis>
```

1.7 Requisição de arquivos

Em linguagens de script como o PHP, frequentemente precisamos incluir dentro de nossos programas outros arquivos com definições de funções, constantes, configurações ou mesmo carregar um arquivo contendo a definição de uma classe. Para atingir esse objetivo no PHP, podemos fazer uso de um dos seguintes comandos:

include <*arquivo*>

A instrução `include()` inclui e avalia o arquivo informado. Seu código (variáveis, objetos e arrays) entra no escopo do programa, tornando-se disponível a partir da linha em que a inclusão ocorre. Se o arquivo não existir, produzirá uma mensagem de advertência (*warning*).

Exemplo:

tools.php

```
<?php
function quadrado($numero) {
    return $numero * $numero;
}
```

teste.php

```
<?php
include 'tools.php'; // carrega arquivo com a função necessária
echo quadrado(4); // imprime o quadrado do número 4
```

Resultado:

```
16
```

require <*arquivo*>

Similar ao `include`. Difere somente na manipulação de erros. Enquanto o `include` produz uma mensagem de advertência, o `require` produz um erro fatal caso o arquivo não exista.

include_once <*arquivo*>

Funciona da mesma maneira que o comando `include`, a não ser que o arquivo informado já tenha sido incluído. Neste caso, a operação não é refeita (o arquivo é incluído apenas uma vez). É útil em casos em que o programa passa mais de uma vez pela mesma instrução. Assim evitará sobreposições, redeclarações etc.

require_once <*arquivo*>

Funciona da mesma maneira que o comando `require`, a não ser que o arquivo informado já tenha sido incluído. Neste caso, a operação não é refeita (o arquivo é incluído apenas uma vez). É útil em casos em que o programa passa mais de uma vez pela mesma instrução. Assim podem-se evitar sobreposições, redeclarações etc.

1.8 Manipulação de 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.

1.8.1 Criação

Para declarar uma função em PHP, utiliza-se o operador `function` seguido do nome que desejamos definir, 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
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 massa corporal. A função recebe dois parâmetros (`$peso` e `$altura`) e retorna o valor definido pela fórmula.

```
<?php
function calcula_imc($peso, $altura) {
   return $peso / ($altura * $altura);
}
echo calcula_imc(75, 1.85);
```

Resultado:

```
21.913805697589
```

1.8.2 Variáveis globais

Todas as variáveis declaradas dentro do escopo de uma função são locais. Para acessar uma variável externa ao contexto de uma função sem passá-la como parâmetro, é necessário declará-la como `global`. Uma variável global é acessada a partir de qualquer ponto da aplicação. No exemplo a seguir, a função criada converte quilômetros para milhas, enquanto acumula a quantidade de quilômetros percorridos em uma variável global (`$total`).

```
<?php
$total = 0;
function km2mi($quilometros) {
   global $total;
   $total += $quilometros;
   return $quilometros * 0.6;
}
echo 'percorreu ' . km2mi(100) . " milhas <br>\n";
echo 'percorreu ' . km2mi(200) . " milhas <br>\n";
echo 'percorreu no total ' . $total . " quilometros <br>\n";
```

Resultado:

```
percorreu 60 milhas
percorreu 120 milhas
percorreu no total 300 quilometros
```

Observação: a utilização de variáveis globais não é considerada uma boa prática de programação, pois uma variável global pode ser alterada a partir de qualquer parte da aplicação. Dessa forma valores inconsistentes podem ser armazenados em uma variável global, podendo resultar em um comportamento inesperado da aplicação.

1.8.3 Variáveis estáticas

Dentro do escopo de uma função, podemos armazenar variáveis de forma estática. Assim elas mantêm o valor que lhes foi atribuído na última execução. Declaramos uma variável estática com o operador `static`.

```
<?php
function percorre($quilometros) {
   static $total;
   $total += $quilometros;
   echo "percorreu mais $quilometros do total de $total<br>\n";
}
percorre(100);
percorre(200);
percorre(50);
```

Resultado:

```
percorreu mais 100 do total de 100
percorreu mais 200 do total de 300
percorreu mais 50 do total de 350
```

1.8.4 Passagem de parâmetros

Existem dois tipos de passagem de parâmetros: por valor (*by value*) e por referência (*by reference*). Por padrão os valores são passados *by value* para as funções. Assim o parâmetro que a função recebe é tratado como variável local dentro do contexto da função, não alterando o seu valor externo. Os objetos são uma exceção, pois são tratados por referência na passagem de parâmetros.

```
<?php
function incrementa($variavel, $valor) {
   $variavel += $valor;
}
$a = 10;
incrementa($a, 20);
echo $a;
```

💻 **Resultado:**

```
10
```

Para efetuar a passagem de parâmetros *by reference* para as funções, basta utilizar o operador `&` na frente do parâmetro; isso faz com que as transformações realizadas pela função sobre a variável sejam válidas no contexto externo à função.

```
<?php
function incrementa(&$variavel, $valor) {
   $variavel += $valor;
}
$a = 10;
incrementa($a, 20);
echo $a;
```

💻 **Resultado:**

```
30
```

O PHP permite definir valores default para parâmetros. Reescreveremos a função anterior, declarando o default de `$valor` como sendo 40. Assim se o programador executar a função sem especificar o parâmetro, será assumido o valor 40.

```
<?php
function incrementa(&$variavel, $valor = 40) {
   $variavel += $valor;
}
$a = 10;
incrementa($a);
echo $a;
```

💻 **Resultado:**

```
50
```

O PHP também permite definir uma função com o número de argumentos variáveis, ou seja, permite obtê-los de forma dinâmica, mesmo sem saber quais são ou quantos são. Para obter quais são, utilizamos a função `func_get_args()`; para obter a quantidade de argumentos, utilizamos a função `func_num_args()`.

```
<?php
function ola() {
    $argumentos = func_get_args();
    $quantidade = func_num_args();

    for($n=0; $n<$quantidade; $n++) {
        echo 'Olá ' . $argumentos[$n] . ', ';
    }
}
ola('João', 'Maria', 'José', 'Pedro');
```

💻 **Resultado:**

```
Olá João, Olá Maria, Olá José, Olá Pedro,
```

1.8.5 Recursão

O PHP permite chamada de funções recursivamente. No caso a seguir, criaremos uma função para calcular o fatorial de um número.

```
<?php
function fatorial($numero) {
    if ($numero == 1)
        return $numero;
    else
        return $numero * fatorial($numero -1);
}
echo fatorial(5) . "<br>\n";
echo fatorial(7) . "<br>\n";
```

💻 **Resultado:**

```
120
5040
```

1.8.6 Funções anônimas

Funções anônimas, ou *lambda functions*, são funções que podem ser definidas em qualquer instante e, diferentemente das funções tradicionais, não têm um nome definido. Funções anônimas podem ficar atreladas à uma variável. Nesse caso a variável é utilizada para fazer a chamada imediata da função, bem como passá-la como parâmetro de alguma função. Caso a variável não esteja visível no escopo, a função fica fora de alcance. Uma função anônima pode ser vista como uma função descartável, que se aplica ao contexto no qual é criada.

Observação: no PHP, funções anônimas são instâncias de uma classe interna chamada Closure.

Funções anônimas são úteis para várias coisas, como passagem de parâmetros e uso como *callback* de funções. Para iniciar a explicação de funções anônimas, vamos criar uma função anônima que remove o acento de alguns caracteres e atribuir essa função à variável `$remove_acento`. Veja que a função não tem nome. A única referência para esse código criado é a variável para a qual a função foi atribuída.

A partir da atribuição, a função pode ser atribuída e passada como parâmetro para funções e métodos de objetos. Sempre que quisermos utilizar a função anônima, basta utilizar a variável na qual ela está definida, como estamos fazendo neste exemplo ao remover os acentos de "José da Conceição". Por fim, estamos definindo um vetor com vários termos acentuados e chamando a função `array_map()`, que recebe em seu primeiro parâmetro uma função a ser aplicada (`Callback`) e como segundo parâmetro um vetor a ser percorrido. Neste caso estamos aplicando a função `$remove_acento` ao vetor e exibindo seu resultado.

```
<?php
$remove_acento = function($str) {
    $a = array('À', 'Á', 'Â', 'Ã', 'Ä', 'Ç', 'È', 'É', 'Ê',
               'Ì', 'Í', 'Ò', 'Ó', 'Ô', 'Õ', 'Ù', 'Ú', 'à',
               'á', 'â', 'ã', 'ç', 'è', 'é', 'ê', 'í', 'î',
               'ò', 'ó', 'ô', 'õ', 'ù', 'ú', 'û', 'ü');

    $b = array('A', 'A', 'A', 'A', 'A', 'C', 'E', 'E', 'E',
               'I', 'I', 'O', 'O', 'O', 'O', 'U', 'U', 'a',
               'a', 'a', 'a', 'c', 'e', 'e', 'e', 'i', 'i',
               'o', 'o', 'o', 'o', 'u', 'u', 'u', 'u');

    return str_replace($a, $b, $str);
};
```

```
print $remove_acento('José da Conceição');
print '<br>' . PHP_EOL;

$palavras = array();
$palavras[] = 'José da Conceição';
$palavras[] = 'Jéferson Araújo';
$palavras[] = 'Félix Júnior';
$palavras[] = 'Ênio Müller';
$palavras[] = 'Ângelo Ônix';

// array array_map ( callback $callback , array $arr1 [, array $... ] )
$r = array_map( $remove_acento, $palavras );
print_r($r);
```

Resultado:

```
Jose da Conceicao
Array
(
    [0] => Jose da Conceicao
    [1] => Jeferson Araujo
    [2] => Felix Junior
    [3] => Enio Muller
    [4] => Angelo Onix
)
```

1.9 Manipulação de arquivos e diretórios

A seguir veremos uma série de funções utilizadas exclusivamente para manipulação de arquivos, como sua abertura, leitura, escrita e seu fechamento.

fopen

Abre um arquivo e retorna um identificador. Aceita arquivos no formato "http://...".

```
resource fopen ( string $filename , string $mode [, ...] )
```

feof

Testa se determinado identificador de arquivo (criado pela função `fopen()`) está no fim de arquivo (End Of File).

```
bool feof ( resource $handle )
```

fclose

Fecha o arquivo aberto apontado pela `fopen()`. Retorna `TRUE` em caso de sucesso ou `FALSE` em caso de falha.

```
bool fclose ( resource $handle )
```

fgets

Lê uma linha de um arquivo aberto pela `fopen()` no formato de string. Recebe opcionalmente um tamanho máximo a ser lido. Se ocorrer um erro, retorna `FALSE`.

```
string fgets ( resource $handle [, int $length ] )
```

Observação: neste exemplo estamos abrindo o próprio arquivo de código-fonte (identificado pela constante mágica `__FILE__`).

Exemplo:

```
<?php
$fd = fopen ( __FILE__, "r");
$linha = 1;
while (!feof ($fd)) {
    $buffer = fgets($fd, 4096); // lê uma linha do arquivo
    if ($linha > 1) {
        echo $buffer; // imprime a linha
    }
    $linha ++;
}
fclose ($fd);
```

Resultado:

```
O próprio código-fonte
```

fwrite

Grava uma string no arquivo apontado pelo identificador (*handle*) de arquivo. Se o argumento *comprimento* (*length*) é informado, a gravação será limitada à esse tamanho. Retorna o número de bytes gravados ou `FALSE` em caso de erro.

```
int fwrite ( resource $handle , string $string [, int $length ] )
```

Exemplo:

```
<?php
$fp = fopen ("/tmp/file.txt", "w");  // abre o arquivo para gravação
```

```
// escreve no arquivo
fwrite ($fp, "linha 1" . PHP_EOL);
fwrite ($fp, "linha 2" . PHP_EOL);
fwrite ($fp, "linha 3" . PHP_EOL);
fclose ($fp); // fecha o arquivo
```

file_put_contents

Grava uma string (*data*) em um arquivo (*filename*). Retorna a quantidade de bytes gravados.

```
int file_put_contents ( string $filename , mixed $data [, ...] )
```

Exemplo:

```
<?php
echo file_put_contents('/tmp/teste.txt', "este \n é o conteúdo \n do arquivo");
```

file_get_contents

Lê o conteúdo de um arquivo e retorna o conteúdo em forma de string.

```
string file_get_contents ( string $filename [, ...] )
```

Exemplo:

```
<?php
echo file_get_contents('/etc/mtab');
```

Resultado:

```
/dev/hda3 / ext3 rw 0 0
/dev/hda1 /windows ntfs rw 0 0
proc /proc proc rw 0 0
usbfs /proc/bus/usb usbfs rw 0 0
```

file

Lê um arquivo e retorna um array com seu conteúdo, de modo que cada posição do array represente uma linha do arquivo. O nome do arquivo pode conter o protocolo, como no caso *http://www.server.com/page.html*. Assim, o arquivo remoto será lido.

```
array file ( string $filename [, ...] )
```

Exemplo:

```
<?php
$arquivo = file ("/tmp/file.txt");
foreach ($arquivo as $linha) {
```

```
    print $linha;
}
```

Resultado:

```
linha 1  linha 2  linha 3
```

copy

Copia um arquivo para outro local/nome. Retorna `TRUE` caso tenha sucesso e `FALSE` em caso de falhas.

```
bool copy ( string $source , string $dest [, ... ] )
```

Exemplo:

```
<?php
$origem  = "/tmp/file.txt";
$destino = "/tmp/file2.txt";
if (copy($origem, $destino))
   echo "Cópia efetuada";
else
   echo "Cópia não efetuada";
```

Resultado:

```
Cópia efetuada
```

rename

Altera a nomenclatura de um arquivo ou diretório.

```
bool rename ( string $oldname , string $newname [, .. ] )
```

Exemplo:

```
<?php
$origem  = "/tmp/file2.txt";
$destino = "/tmp/file3.txt";
if (rename($origem, $destino))
   echo "Renomeação efetuada";
else
   echo "Renomeação não efetuada";
```

Resultado:

```
Renomeação efetuada
```

unlink

Apaga um arquivo passado como parâmetro. Retorna `TRUE` em caso de sucesso e `FALSE` em caso de falhas.

```
bool unlink ( string $filename [, resource $context ] )
```

Exemplo:

```
<?php
$arquivo = "/tmp/file3.txt";
if (unlink($arquivo))
   echo "Arquivo apagado";
else
   echo "Arquivo não apagado";
```

Resultado:

```
Arquivo apagado
```

file_exists

Verifica a existência de um arquivo ou de um diretório.

```
bool file_exists ( string $filename )
```

Exemplo:

```
<?php
$arquivo   = '/tmp/file2.txt';
if (file_exists($arquivo))
   echo "Arquivo existente";
else
   echo "Arquivo não existente";
```

Resultado:

```
Arquivo não existente
```

is_file

Verifica se a localização dada corresponde ou não a um arquivo.

```
bool is_file ( string $filename )
```

Exemplo:

```
<?php
$arquivo = '/tmp/file.txt';
```

```
if (is_file($arquivo))
   echo "É um arquivo";
else
   echo "Não é um arquivo";
```

Resultado:

```
É um arquivo
```

mkdir

Cria um diretório informando, entre outras opções, a sua permissão de acesso.

```
bool mkdir ( string $pathname [, int $mode = 0777 [, bool $recursive = false
    [, resource $context ]]] )
```

Exemplo:

```
<?php
$dir = '/tmp/diretorio';
if (mkdir($dir, 0777))
    echo "Diretório criado com sucesso";
else
    echo "Diretório não criado";
```

Resultado:

```
Diretório criado com sucesso
```

rmdir

Apaga um diretório.

```
bool rmdir ( string $dirname [, resource $context ] )
```

Exemplo:

```
<?php
$dir = '/tmp/diretorio';
if (rmdir($dir))
   echo "Diretório apagado com sucesso";
else
   echo "Diretório não apagado";
```

Resultado:

```
Diretório apagado com sucesso
```

scandir

Lê o conteúdo de um diretório (arquivos e diretórios), retornando-o como um array.

```
array scandir ( string $directory [, int $sorting_order = SCANDIR_SORT_ASCENDING
    [, resource $context ]] )
```

Exemplo:

```
<?php
$diretorio = '/';
if (is_dir($diretorio)) {
    $linhas = scandir($diretorio);
    foreach ($linhas as $linha) {
        print $linha . '<br>' . PHP_EOL;
    }
}
```

💻 **Resultado:**

```
.
..
bin
boot
dev
etc
...
```

1.10 Manipulação de strings

1.10.1 Declaração

Uma string é uma cadeia de caracteres alfanuméricos. Para declarar uma string, podemos utilizar aspas simples ‘ ‘ ou aspas duplas “ “.

```
$variavel = 'Isto é um teste';
$variavel = "Isto é um teste";
```

A diferença é que todo conteúdo dentro de aspas duplas é avaliado pelo PHP. Assim, se a string contém uma variável, essa variável será traduzida pelo seu valor.

```
<?php
$fruta = 'maçã';
```

```
print "como $fruta<br>" . PHP_EOL;  // resultado: como maçã
print 'como $fruta<br>' . PHP_EOL;  // resultado: como $fruta
```

Também podemos declarar uma string literal com muitas linhas observando a sintaxe a seguir, na qual escolhemos uma palavra-chave (neste caso escolhemos `CHAVE`) para delimitar o início e o fim da string.

```
<?php
$texto = <<<CHAVE
Aqui nesta área
você poderá escrever
textos com múltiplas linhas
CHAVE;
echo $texto;
```

Resultado:

```
Aqui nesta área
você poderá escrever
textos com múltiplas linhas.
```

1.10.2 Concatenação

Para concatenar strings, pode-se utilizar o operador "." ou colocar múltiplas variáveis dentro de strings duplas "", visto que seu conteúdo é interpretado.

```
<?php
$fruta = 'maçã';
echo $fruta .' é a fruta de adão<br>' . PHP_EOL; // resultado: maçã é a fruta de adão
echo "{$fruta} é a fruta de adão<br>" . PHP_EOL; // resultado: maçã é a fruta de adão
```

O PHP realiza automaticamente a conversão entre tipos, como neste exemplo de concatenação entre uma string e um número:

```
<?php
$a = 1234;

echo 'O salário é ' . $a . '<br>' . PHP_EOL;
echo "O salário é $a" . '<br>' . PHP_EOL;
```

Resultado:

```
O salário é 1234
O salário é 1234
```

1.10.3 Caracteres de escape

Dentro de aspas duplas "" podemos utilizar caracteres de escape (\) – controles especiais interpretados diferentemente pelo PHP. Veja a seguir os mais utilizados:

Caractere	Descrição
\n	Nova linha (proporciona uma quebra de linha).
\r	Retorno de carro.
\t	Tabulação.
\\	Barra invertida "\\" (o mesmo que '\').
\"	Aspas duplas.
\$	Símbolo de $.

Exemplo:

```
<?php
echo "seu nome é \"Paulo\".<br>" . PHP_EOL;  // resultado: seu nome é "Paulo".
echo 'seu nome é "Paulo".<br>' . PHP_EOL;   // resultado: seu nome é "Paulo".
echo 'seu salário é $650,00<br>' . PHP_EOL;  // seu salário é $650,00
echo "seu salário é \$650,00<br>" . PHP_EOL;  // seu salário é $650,00
```

Observação: utilize aspas duplas para declarar strings somente quando for necessário avaliar seu conteúdo, evitando, assim, tempo de processamento desnecessário.

1.10.4 Funções para manipulação de strings

As funções a seguir formam um grupo cuja característica comum é a manipulação de cadeias de caracteres (strings), como conversões, transformações etc.

strtoupper

Retorna a string com todos os caracteres alfabéticos convertidos para letras maiúsculas

```
string strtoupper ( string $string )
```

strtolower

Retorna a string com todos os caracteres alfabéticos convertidos para letras minúsculas.

```
string strtolower ( string $string )
```

Exemplo:

```
<?php
echo strtoupper('Convertendo para maiusculo') . '<br>' . PHP_EOL;
echo strtolower('CONVERTENDO PARA MINUSCULO') . '<br>' . PHP_EOL;
```

Resultado:

```
CONVERTENDO PARA MAIUSCULO
convertendo para minusculo
```

substr

Retorna uma parte de uma string. O primeiro parâmetro representa a string original, o segundo representa o início do corte, e o terceiro, o tamanho do corte. Se o *comprimento* (*length*) for negativo, conta *n* caracteres antes do final.

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

Exemplo:

```
<?php
print substr("Americana", 1) . '<br>'. PHP_EOL;
print substr("Americana", 1, 3) . '<br>'. PHP_EOL;
print substr("Americana", 0, -1) . '<br>'. PHP_EOL;
print substr("Americana", -2) . '<br>'. PHP_EOL;
```

Resultado:

```
mericana
mer
American
na
```

strpad

Preenche uma string com uma outra string, dentro de um tamanho específico, podendo preencher com caracteres à esquerda, direita ou centralizado.

```
string str_pad ( string $input , int $pad_length [, string $pad_string = " "
    [, int $pad_type = STR_PAD_RIGHT ]] )
```

Exemplo:

```
<?php
$texto = "The Beatles";
print str_pad($texto, 20, "*", STR_PAD_LEFT) . "<br>\n";
print str_pad($texto, 20, "*", STR_PAD_BOTH) . "<br>\n";
print str_pad($texto, 20, "*") . "<br>\n";
```

Resultado:

```
*********The Beatles
****The Beatles*****
The Beatles*********
```

str_repeat

Repete uma string certa quantidade de vezes.

```
string str_repeat ( string $input , int $multiplier )
```

Exemplo:

```
<?php
$txt = ".oOOOo.";
print str_repeat($txt, 5) . "\n";
```

Resultado:

```
.oOOOo..oOOOo..oOOOo..oOOOo..oOOOo.
```

strlen

Retorna o comprimento de uma string.

```
int strlen ( string $string )
```

Exemplo:

```
<?php
$txt = "O Rato roeu a roupa do rei de roma";
print 'O comprimento é: ' . strlen($txt) . "\n";
```

Resultado:

```
O comprimento é: 34
```

str_replace

Substitui uma string (primeiro parâmetro) por outra (segundo parâmetro) dentro de um dado contexto (terceiro parâmetro).

```
mixed str_replace (mixed $search , mixed $replace , mixed $subject [, int &$count ])
```

Exemplo:

```
<?php
print str_replace('Rato', 'Leão', 'O Rato roeu a roupa do rei de Roma');
```

Resultado:

```
O Leão roeu a roupa do rei de Roma
```

1.11 Manipulação de arrays

A manipulação de arrays no PHP é, sem dúvida, um dos recursos mais poderosos dessa linguagem. O programador que assimilar bem esta parte terá muito mais produtividade no seu dia a dia. Isso porque os arrays no PHP servem como verdadeiros contêineres, armazenando números, strings, objetos, entre outros, de forma dinâmica. Além disso, o PHP nos oferece uma gama de funções para manipulá-los, as quais serão vistas a seguir.

1.11.1 Criando um array

Arrays são acessados mediante uma posição, como um índice numérico. Para criar um array, pode-se utilizar a função `array([chave =>] valor , ... )` ou a sintaxe resumida entre [].

```
$cores = array(0=>'vermelho', 1=>'azul', 2=>'verde', 3=>'amarelo');
```

ou

```
$cores = array('vermelho', 'azul', 'verde', 'amarelo');
```

ou

```
$cores = [ 'vermelho', 'azul', 'verde', 'amarelo' ];
```

Outra forma de criar um array é simplesmente adicionando-lhe valores com a seguinte sintaxe:

```
$nomes[] = 'maria';
$nomes[] = 'joão';
$nomes[] = 'carlos';
$nomes[] = 'josé';
```

De qualquer forma, para acessar o array indexado, basta indicar o seu índice entre colchetes:

```
echo $cores[0];  // resultado = vermelho
echo $cores[1];  // resultado = azul
echo $cores[2];  // resultado = verde
echo $cores[3];  // resultado = amarelo
```

```
echo $nomes[0];  // resultado = maria
echo $nomes[1];  // resultado = joão
echo $nomes[2];  // resultado = carlos
echo $nomes[3];  // resultado = josé
```

1.11.2 Arrays associativos

Os arrays no PHP são associativos, pois contêm uma chave de acesso para cada posição. Para criar um array, pode-se utilizar a função `array([chave =>] valor , ... )`.

```
$cores = array('vermelho' => 'FF0000', 'azul' => '0000FF', 'verde' => '00FF00');
```

Outra forma de criar um array associativo é simplesmente adicionando-lhe valores com a seguinte sintaxe:

```
$pessoa = array();

$pessoa['nome'] = 'Maria da Silva';
$pessoa['rua'] = 'São João';
$pessoa['bairro'] = 'Cidade Alta';
$pessoa['cidade'] = 'Porto Alegre';
```

De qualquer forma, para acessar o array, basta indicar a sua chave entre colchetes:

```
echo $cores['vermelho'];  // resultado = FF0000
echo $cores['azul'];  // resultado = 0000FF
echo $cores['verde'];  // resultado = 00FF00

echo $pessoa['nome'];  // resultado = Maria da Silva
echo $pessoa['rua'];  // resultado = São João
echo $pessoa['bairro'];  // resultado = Cidade Alta
echo $pessoa['cidade'];  // resultado = Porto Alegre
```

Observação: a chave pode ser string ou integer não negativo; o valor pode ser de qualquer tipo.

1.11.3 Iterações

Os arrays podem ser iterados no PHP pelo operador `FOREACH`, que percorre cada uma das posições do array. Exemplo:

```
$frutas = array();
$frutas['cor']     = 'vermelha';
```

```
$frutas['sabor']   = 'doce';
$frutas['formato'] = 'redondo';
$frutas['nome']    = 'maçã';
foreach ($frutas as $chave => $fruta) {
    echo "$chave => $fruta <br>\n";
}
```

Resultado:

```
cor => vermelha
sabor => doce
formato => redondo
nome => maçã
```

1.11.4 Acessando arrays

As posições de um array podem ser acessadas a qualquer momento, e sobre elas operações podem ser realizadas.

```
<?php
$contato = array();
$contato['nome'] = 'Pablo';
$contato['empresa'] = 'Adianti';
$contato['peso'] = 73;
// alterações
$contato['nome'] .= ' Dall Oglio';
$contato['empresa'] .= ' Solutions';
$contato['peso'] += 2;
// debug
print_r($contato);
```

```
$comidas = array();
$comidas[] = 'Lasanha';
$comidas[] = 'Pizza';
$comidas[] = 'Macarrão';
// alterações
$comidas[1] = 'Pizza Calabresa';
// debug
print_r($comidas);
```

Resultado:

```
Array
(
    [nome] => Pablo Dall Oglio
    [empresa] => Adianti Solutions
    [peso] => 75
)
Array
(
    [0] => Lasanha
    [1] => Pizza Calabresa
    [2] => Macarrão
)
```

1.11.5 Arrays multidimensionais

Arrays multidimensionais ou matrizes são arrays nos quais algumas de suas posições podem conter outros arrays de forma recursiva. Um array multidimensional pode ser criado pela função `array()`:

```
<?php
$carros = array('Palio' => array('cor'=>'azul',
                                 'potência'=>'1.0',
                                 'opcionais'=>'Ar Cond.'),
                'Corsa' => array('cor'=>'cinza',
                                 'potência'=>'1.3',
                                 'opcionais'=>'MP3'),
                'Gol'   => array('cor'=>'branco',
                                 'potência' => '1.0',
                                 'opcionais' => 'Metalica')
                );
echo $carros['Palio']['opcionais']; // resultado = Ar Cond.
```

Outra forma de criar um array multidimensional é simplesmente atribuindo-lhe valores:

```
<?php
$carros = array();
$carros['Palio']['cor']       = 'azul';
$carros['Palio']['potência']  = '1.0';
$carros['Palio']['opcionais'] = 'Ar Cond.';
$carros['Corsa']['cor']       = 'cinza';
```

```
$carros['Corsa']['potência']  = '1.3';
$carros['Corsa']['opcionais'] = 'MP3';
$carros['Gol']['cor']         = 'branco';
$carros['Gol']['potência']    = '1.0';
$carros['Gol']['opcionais']   = 'Metalica';

echo $carros['Palio']['opcionais']; // resultado = Ar Cond.
```

Para realizar iterações em um array multidimensional, é preciso observar quantos níveis ele tem. No exemplo a seguir, realizamos uma iteração para o primeiro nível do array (veículos) e, para cada iteração, realizamos uma nova iteração para imprimir suas características.

```
foreach ($carros as $modelo => $caracteristicas) {
    echo "=> modelo $modelo<br>\n";
    foreach ($caracteristicas as $caracteristica => $valor) {
        echo " - característica $caracteristica => $valor<br>\n";
    }
}
```

Resultado:

```
=> modelo Palio
   - característica cor => azul
   - característica potência => 1.0
   - característica opcionais => Ar Cond.
=> modelo Corsa
   - característica cor => cinza
   - característica potência => 1.3
   - característica opcionais => MP3
=> modelo Gol
   - característica cor => branco
   - característica potência => 1.0
   - característica opcionais => Metalica
```

1.11.6 Funções para manipulação de arrays

A seguir, veremos uma série de funções utilizadas exclusivamente para manipulação de arrays; funções de ordenação, intersecção, acesso, entre outras.

array_unshift

Adiciona elemento(s) ao início de um array.

```
int array_unshift ( array &$array , mixed $var [, mixed $... ] )
```

array_push

Adiciona elemento(s) ao final de um array. Tem o mesmo efeito que `$array[] = $valor`.

```
int array_push ( array &$array , mixed $var [, mixed $... ] )
```

Exemplo:

```
<?php
$a = array("verde", "azul", "vermelho");
array_unshift($a, "ciano");
array_push($a, "amarelo");
print_r($a);
```

Resultado:

```
Array
(
    [0] => ciano
    [1] => verde
    [2] => azul
    [3] => vermelho
    [4] => amarelo
)
```

array_shift

Remove um elemento do início de um array.

```
mixed array_shift ( array &$array )
```

array_pop

Remove um valor do final de um array.

```
mixed array_pop ( array &$array )
```

Exemplo:

```
<?php
$a = array("ciano", "verde", "azul", "vermelho", "amarelo");
array_shift($a);
array_pop($a);
print_r($a);
```

Resultado:

```
Array
(
```

```
    [0] => verde
    [1] => azul
    [2] => vermelho
)
```

array_reverse

Recebe um array e retorna-o na ordem inversa.

```
array array_reverse ( array $array [, bool $preserve_keys ] )
```

Exemplo:

```
<?php
$a[0] = 'verde';
$a[1] = 'amarelo';
$a[2] = 'vermelho';
$a[3] = 'azul';
$b = array_reverse($a, true);
print_r($b);
```

Resultado:

```
Array
(
    [3] => azul
    [2] => vermelho
    [1] => amarelo
    [0] => verde
)
```

array_merge

Mescla dois ou mais arrays. Um array é adicionado ao final do outro. O resultado é um novo array. Se ambos os arrays tiverem conteúdo indexado pela mesma chave, o segundo irá se sobrepor ao primeiro.

```
array array_merge (array nome_array1, array nome_array2 [, array ...])
```

Exemplo:

```
<?php
$a = array("verde", "azul");
$b = array("vermelho", "amarelo");
$c = array_merge($a, $b);
print_r($c);
```

Resultado:

```
Array
(
    [0] => verde
    [1] => azul
    [2] => vermelho
    [3] => amarelo
)
```

array_keys

Retorna as chaves (índices) de um array. Se o segundo parâmetro for indicado, a função retornará apenas índices que apontam para um conteúdo igual ao parâmetro.

```
array array_keys ( array $input [, mixed $search_value [, bool $strict ]] )
```

array_values

Retorna somente os valores de um array, ignorando suas chaves.

```
array array_values ( array $input )
```

count

Retorna a quantidade de elementos de um array.

```
int count (array nome_array)
```

Exemplo:

```
<?php
$exemplo = array('cor' => 'vermelho', 'volume' => 5, 'animal'=>'cachorro');
print_r(array_keys($exemplo));
print_r(array_values($exemplo));
print 'Quantidade: ' . count($exemplo);
```

Resultado:

```
Array
(
    [0] => cor
    [1] => volume
    [2] => animal
)
Array
(
    [0] => vermelho
```

```
    [1] => 5
    [2] => cachorro
)
Quantidade: 3
```

in_array

Verifica se um array contém um determinado valor.

```
bool in_array ( mixed $needle , array $haystack [, bool $strict ] )
```

Exemplo:

```
<?php
$a = array('refrigerante', 'cerveja', 'vodca', 'suco natural');
if (in_array('suco natural', $a)) {
   echo 'suco natural encontrado';
}
```

💻 Resultado:

```
suco natural encontrado
```

sort

Ordena um array pelo seu valor, sem manter a associação de índices. Para ordem reversa, utilize `rsort()`.

```
bool sort ( array &$array [, int $sort_flags ] )
```

Exemplo:

```
<?php
$a = array('refrigerante', 'cerveja', 'vodca', 'suco natural');
sort($a);
print_r($a);
```

💻 Resultado:

```
Array
(
    [0] => cerveja
    [1] => refrigerante
    [2] => suco natural
    [3] => vodca
)
```

asort

Ordena um array pelo seu valor, mantendo a associação de índices. Para ordenar de forma reversa, use `arsort()`.

```
bool asort ( array &$array [, int $sort_flags ] )
```

Exemplo:

```
<?php
$a[0] = 'verde';
$a[1] = 'amarelo';
$a[2] = 'vermelho';
$a[3] = 'azul';
asort($a);
print_r($a);
```

💻 **Resultado:**

```
Array
(
    [1] => amarelo
    [3] => azul
    [0] => verde
    [2] => vermelho
)
```

ksort

Ordena um array pelos seus índices. Para ordem reversa, utilize `krsort()`.

```
int ksort ( array &$array [, int $sort_flags ] )
```

Exemplo:

```
<?php
$carro['potência']  = '1.0';
$carro['cor']       = 'branco';
$carro['modelo']    = 'celta';
$carro['opcionais'] = 'ar quente';
ksort($carro);
print_r($carro);
```

💻 **Resultado:**

```
Array
(
```

```
    [cor] => branco
    [modelo] => celta
    [opcionais] => ar quente
    [potência] => 1.0
)
```

explode

Converte uma string em um array, quebrando os elementos por meio de um separador (*delimiter*).

```
array explode ( string $delimiter , string $string [, int $limit ] )
```

implode

Converte um array em uma string, agrupando os elementos do array por meio de um elemento cola (*glue*).

```
string implode ( string $glue , array $pieces )
```

Exemplo:

```
<?php
$string = "10/05/2015";
print_r( explode("/", $string) ); // string  array
$array = [ 10, 05, 2015 ];
print implode('/', $array); // array  string
```

💻 **Resultado:**

```
Array
(
    [0] => 10
    [1] => 05
    [2] => 2015
)
10/5/2015
```

1.12 Acesso a bases de dados

1.12.1 Introdução

No atual mundo das aplicações de negócios, o armazenamento de informações de uma organização é implementado por diferentes mecanismos de bancos de dados em decorrência, muitas vezes, da utilização de sistemas de diferentes fornecedores

e tecnologias. Alguns dados relativos a recursos humanos podem estar armazenados em um banco de dados DB2, ao passo que os dados financeiros podem estar armazenados em um Oracle, as estatísticas de acesso ao site da empresa podem estar em um MySQL e o Data Warehouse, que consiste de várias informações estratégicas, pode estar em um PostgreSQL. A diversidade reside nos mais diversos fornecedores de bancos de dados, cada qual com métodos diferentes (interfaces) de acesso aos seus serviços.

Observação: aprenderemos primeiramente a utilizar bancos de dados no PHP de maneira procedural para, ao longo do livro, construir classes que tornarão o acesso totalmente orientado a objetos.

O PHP conecta-se a uma enorme variedade de gerenciadores de bancos de dados disponíveis no mercado. Para cada tipo de banco de dados, existe uma gama de funções associadas para realizar operações como conexão, consulta, retorno, desconexão, entre outras. Nesta seção abordaremos resumidamente as principais funções nativas para conexão. Se você deseja desenvolver uma aplicação utilizando qualquer um dos bancos de dados listados a seguir, é imprescindível visitar o site do PHP para obter a lista completa de funções de acesso.

Banco	Site
Sybase	*www.php.net/sybase*
SQL Server	*www.php.net/mssql*
MySQL	*www.php.net/mysqli*
Frontbase	*www.php.net/fbsql*
Interbase	*www.php.net/ibase*
ODBC	*www.php.net/odbc*
Informix	*www.php.net/ifx*
Oracle	*www.php.net/oci8*
PostgreSQL	*www.php.net/pgsql*
SQLite	*www.php.net/sqlite*

Observação: neste primeiro momento entenderemos como funciona a conexão a bases de dados de maneira procedural. No capítulo 3 será abordada a biblioteca PDO, que permite uma conexão ao banco de dados de maneira totalmente orientada a objetos.

1.12.2 Exemplos

Para exemplificar o acesso ao banco de dados, criaremos dois programas. O primeiro deles será responsável por inserir dados em um banco de dados PostgreSQL; o segundo irá listar os resultados inseridos pelo primeiro programa. Para criar o banco de dados utilizado nos exemplos, certifique-se de que você tem o PostgreSQL instalado em sua máquina e utilize os seguintes comandos:

```
# createdb livro
# psql livro
livro=# CREATE TABLE famosos (codigo integer, nome varchar(40));
CREATE TABLE
livro=# \q
```

Observação: é possível criar a base de dados utilizando o software PgAdmin (www.pgadmin.org), que apresenta uma interface totalmente gráfica para gerenciar o banco de dados.

Neste primeiro exemplo o programa se conectará ao banco de dados local `livro`, localizado em `localhost`, com o usuário `postgres`. Em seguida irá inserir dados de algumas pessoas famosas na base de dados. Por fim, ele fechará a conexão ao banco de dados.

pgsql_insere.php

```
<?php
// abre conexão com Postgres
$conn = pg_connect("host=localhost port=5432 dbname=livro user=postgres password=");

// insere vários registros
pg_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (1, 'Érico Veríssimo')");
pg_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (2, 'John Lennon')");
pg_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (3, 'Mahatma Gandhi')");
pg_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (4, 'Ayrton Senna')");
pg_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (5, 'Charlie Chaplin')");
pg_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (6, 'Anita Garibaldi')");
pg_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (7, 'Mário Quintana')");

// fecha a conexão
pg_close($conn);
```

No próximo exemplo, o programa se conectará ao mesmo banco de dados do exemplo anterior, chamado `livro`, localizado em `localhost`. Em seguida ele irá selecionar código e nome de famosos existentes nesse banco de dados, exibindo-os em tela.

pgsql_lista.php

```
<?php
// abre conexão com Postgres
$conn = pg_connect("host=localhost port=5432 dbname=livro user=postgres password=");

// define consulta que será realizada
$query = 'SELECT codigo, nome FROM famosos';

// envia consulta ao banco de dados
$result = pg_query($conn, $query);

if ($result) {
   // percorre resultados da pesquisa
   while ($row = pg_fetch_assoc($result)) {
        echo $row['codigo'] . ' - ' . $row['nome'] . "<br>\n";
   }
}
// fecha a conexão
pg_close($conn);
```

Resultado:

```
1 - Érico Veríssimo
2 - John Lennon
3 - Mahatma Gandhi
4 - Ayrton Senna
5 - Charlie Chaplin
6 - Anita Garibaldi
7 - Mário Quintana
```

Nos exemplos anteriores, você deve ter notado quais funções precisamos utilizar para nos conectarmos a um banco de dados PostgreSQL e executar comandos SQL. Basicamente temos o `pg_connect()` para abrir uma conexão, `pg_query()` para executar uma instrução SQL e `pg_close()` para fechar uma conexão, entre outros comandos. No PHP, as funções nativas de acesso a banco de dados são específicas para cada sistema gerenciador de banco de dados (SGBD). Isso quer dizer que as funções de acesso e consulta a um banco de dados MySQL são diferentes de um Oracle, um DB2, entre outros. Esse cenário começa a mudar com a biblioteca PDO, apresentada no capítulo 3, que fornece uma interface unificada para acesso a diversos bancos de dados.

No exemplo seguinte, procuramos repetir o mesmo exemplo de inserção de dados em um banco, anteriormente em PostgreSQL, agora em MySQL. Veja que o

exemplo é bastante similar, porém a nomenclatura das funções utilizadas e a ordem dos parâmetros são outras. Para o MySQL, utilizamos basicamente as funções `mysqli_connect()` para conectar ao banco, `mysqli_query()` para enviar a instrução SQL para o banco e `mysqli_close()` para fechar a conexão com o banco de dados. Neste exemplo estamos conectando o programa a um banco de dados `livro` localizado localmente (`127.0.0.1`), com o usuário `root` e a senha `mysql`. Primeiro criaremos a base de dados `livro` e posteriormente criaremos a tabela de famosos.

```
# mysql
mysql> create database livro;
Query OK, 1 row affected (0.00 sec)

mysql> use livro
Database changed

mysql> CREATE TABLE famosos (codigo integer, nome varchar(40));
Query OK, 0 rows affected (0.01 sec)
```

mysql_insere.php

```
<?php
// abre conexão com o MySQL
$conn = mysqli_connect('127.0.0.1', 'root', 'mysql', 'livro');

// insere vários registros
mysqli_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (1, 'Érico Veríssimo')");
mysqli_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (2, 'John Lennon')");
mysqli_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (3, 'Mahatma Gandhi')");
mysqli_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (4, 'Ayrton Senna')");
mysqli_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (5, 'Charlie Chaplin')");
mysqli_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (6, 'Anita Garibaldi')");
mysqli_query($conn, "INSERT INTO famosos (codigo, nome) VALUES (7, 'Mário Quintana')");

// fecha a conexão
mysqli_close($conn);
```

Assim como reescrevemos o exemplo para inserção de dados de PostgreSQL para MySQL, iremos agora reescrever o programa de listagem de dados, responsável por obter os registros do banco de dados e exibi-los em tela.

mysql_lista.php

```
<?php
// abre conexão com o MySQL
$conn = mysqli_connect('127.0.0.1', 'root', 'mysql', 'livro');
```

```
// define a consulta que será realizada
$query = 'SELECT codigo, nome FROM famosos';

// envia consulta ao banco de dados
$result = mysqli_query($conn, $query);

if ($result) {
    // percorre resultados da pesquisa
    while ($row = mysqli_fetch_assoc($result)) {
        echo $row['codigo'] . ' - ' . $row['nome'] . "<br>\n";
    }
}
mysqli_close($conn); // fecha a conexão
```

Resultado:

```
1 - Érico Veríssimo
2 - John Lennon
3 - Mahatma Gandhi
4 - Ayrton Senna
5 - Charlie Chaplin
6 - Anita Garibaldi
7 - Mário Quintana
```

1.13 Formulários

Um formulário é um conjunto de campos dispostos ao usuário de forma agrupada para que este os preencha com informações requisitadas pela aplicação. Um formulário é composto de campos de diversos tipos, como caixas de digitação, radio buttons e combo-boxes, além de ter botões de ação, que definem o programa que processará os dados. Em uma aplicação temos interfaces de entrada de dados, rotinas de processamento e interfaces para apresentação de dados. Podemos dizer que os formulários dominam amplamente as interfaces de entradas de dados em aplicações, portanto é imprescindível dominarmos o seu uso e também simplificarmos ao máximo para ganharmos maior produtividade no desenvolvimento.

Observação: aprenderemos primeiramente a utilizar formulários no PHP de maneira estruturada para, ao longo do livro, construir classes que tornarão esse processo totalmente orientado a objetos.

1.13.1 Elementos de um formulário

Para criarmos um formulário, utilizamos a tag `<form>`. Dentro dela podemos dispor diversos elementos, que chamamos também de widgets em linguagens visuais como o Delphi ou o Gtk. Cada campo é descrito por uma tag. A seguir explicaremos brevemente cada um dos elementos que podem compor um formulário, tendo em vista que o objetivo principal deste livro não é explicar a linguagem HTML, a respeito da qual já existem excelentes livros.

Um formulário está contido entre as tags `<form>` e `</form>`. Destacamos a propriedade `name`, que indicará o nome do formulário, a propriedade `method`, que indicará se os dados serão submetidos por meio de um `post` ou um `get`, vistos a seguir, a propriedade `action`, que indica o script que será acionado, recebendo os dados do formulário e também o tipo de codificação utilizada para o transporte dos dados (`enctype`).

```
<form name="myform" method="post" action="form_gravar.php" enctype="multipart/form-data">
... elementos ...
</form>
```

A seguir veremos quais elementos podem fazer parte de um formulário, ou seja, quais elementos poderão estar contidos entre as tags `<form>` e `</form>`.

O input do tipo `text` é o elemento mais utilizado. Trata-se de uma caixa de digitação de texto simples, composta de uma única linha. É utilizado para digitação de números e strings.

```
<input name='nome' value='teste' type='text' size='40' maxlength=40>
```

O input do tipo `password` é utilizado para a digitação de senhas. Seu comportamento é muito parecido com o input tipo `text`. A diferença está no momento da digitação, quando são exibidos `*` no lugar dos caracteres inseridos pelo usuário.

```
<input name='senha' type='password' size='40'>
```

O input do tipo `checkbox` é utilizado para exibir caixas de verificação. Uma caixa de verificação tem dois estados: clicada ou não. É útil para realizar perguntas "booleanas" ao usuário.

```
<input name='tem_filhos' value='1' type='checkbox' size='40'>
<input name='tem_renda'  value='1' type='checkbox' size='40' checked>
```

O input do tipo `radio` é utilizado para exibir botões de seleção exclusiva, que permitem a seleção de apenas um item entre vários exibidos. É utilizado para permitir ao usuário selecionar uma opção a partir de um conjunto de itens.

```
<input name='sexo' value='F' type='radio'>
<input name='sexo' value='M' type='radio' checked>
```

O input do tipo `file` é utilizado para que o usuário possa selecionar um arquivo e realizar seu upload, enviando-o para processamento do programa.

```
<input name='foto' type='file' size='40'>
```

O input do tipo `hidden` é utilizado para armazenar um campo escondido dentro do formulário. Um campo escondido não é visível ao usuário e é utilizado, na maioria das vezes, para enviar informações que serão processadas sem que o usuário tome conhecimento.

```
<input name='id' type='hidden' size='40'>
```

O input do tipo `button` é utilizado para exibir um botão na tela. Podemos definir ações para esse botão usando funções JavaScript, por exemplo.

```
<input type='button' size='40' value='botão' onclick="alert('teste123')">
```

O input do tipo `submit` é utilizado para exibir o botão de submissão do formulário. Esse botão irá submeter os dados para processamento no servidor, passando todas as informações preenchidas pelo usuário para o script identificado pelo atributo `action` da tag `<form>`.

```
<input type='submit' value='Gravar'>
```

O input do tipo `reset` é utilizado para exibir um botão de limpar o formulário. Botões de `reset` fazem o formulário voltar ao seu estado inicial, ou seja, aos valores iniciais que ele continha quando a página foi carregada pelo navegador.

```
<input type='reset' value='Limpar'>
```

O input do tipo `select` é utilizado para exibir listas de seleção ao usuário. Uma lista de seleção pode permitir ao usuário a seleção de apenas um item (comportando-se como uma combo-box) ou de vários itens ao mesmo tempo (como um campo de múltipla seleção).

```
<select name='cor' size=4 multiple>
   <option value='red'>Vermelho</option>
   <option value='green'>Verde</option>
   <option value='blue' selected>Azul</option></select>
</select>
```

O input do tipo `textarea` disponibiliza ao usuário uma área de digitação de texto com múltiplas linhas e colunas, dentro de uma área delimitada que apresentará barras de rolagem verticais e horizontais quando necessário.

```
<textarea name='resumo' rows='7' cols='40'>digite o resumo aqui</textarea>
```

1.13.2 Exemplo de formulário

No exemplo a seguir temos uma demonstração de vários elementos que compõem um formulário HTML. Uma das formas mais comuns de construir um formulário em tela é utilizar algum contêiner, responsável por distribuir os elementos de um formulário conforme um layout. O contêiner mais utilizado para formulários é a tabela. Utilizando uma estrutura de tabela, podemos alinhar os rótulos de texto e os campos do formulário de forma que fiquem bem dispostos visualmente.

Também temos um formulário para digitação de dados de pessoas. Nele temos um campo de digitação de texto para o nome, um campo de senha (`password`), um grupo de radio buttons para seleção do sexo, uma combo-box para a seleção da religião, um grupo de check buttons para seleção de idiomas, uma caixa de múltipla seleção de estilo, um campo do tipo arquivo para upload da foto e um campo do tipo `textarea` para digitação do currículo, além de botões de ação para gravar (`submit`) e limpar (`reset`) e um botão personalizado que executa uma função JavaScript. Na figura 1.5 temos um formulário HTML.

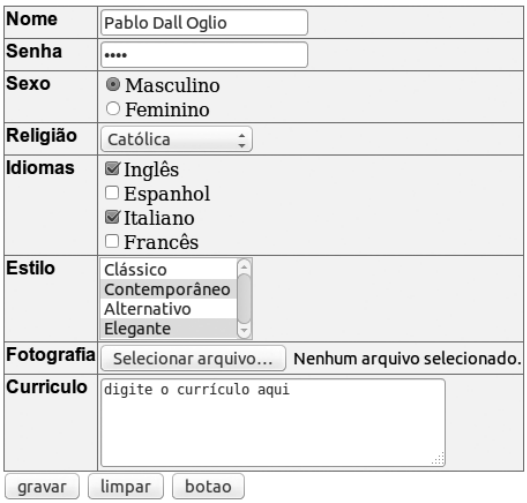


Figura 1.5 – Formulário HTML.

form.html

```
<html>
    <body>
      <form name=myform method=post action="form_gravar.php" enctype="multipart/form-data">
         <table border="1" width="400px" bgcolor="whiteSmoke"
               style="border-collapse:collapse">
            <tr>
               <td valign=top width="200px"><b><font face="Arial">Nome</font></b></td>
               <td><input maxlength="40" name="nome" value="Pablo Dall Oglio" type="text"
                  size="14"></td>
            </tr>
            <tr>
               <td valign=top width="140px"><b><font face="Arial">Senha</font></b></td>
               <td><input name="senha" value="1234" type="password" size="14"></td>
            </tr>
            <tr>
               <td valign=top width="140px"><b><font face="Arial">Sexo</font></b></td>
               <td>
                  <input name="sexo" value="M" type="radio" checked="1">Masculino</input><br>
                  <input name="sexo" value="F" type="radio">Feminino</input><br>
               </td>
            </tr>
            <tr>
               <td valign=top width="140px"><b><font face="Arial">Religião</font></b></td>
               <td>
                  <select name="religiao">
                     <option value="0">Clique Aqui</option>
                     <option value="C">Católica</option>
                     <option value="E">Evangélica</option>
                     <option value="U">Igreja Universal</option>
                  </select>
               </td>
            </tr>
            <tr>
               <td valign=top width="140px"><b><font face="Arial">Idiomas</font></b></td>
               <td>
                  <input name="idiomas[]" value="E" type="checkbox">Inglês<br>
                  <input name="idiomas[]" value="S" type="checkbox">Espanhol<br>
                  <input name="idiomas[]" value="I" type="checkbox">Italiano<br>
                  <input name="idiomas[]" value="F" type="checkbox">Francês<br>
               </td>
```

```
            </tr>
            <tr>
               <td valign=top width="140px"><b><font face="Arial">Estilo</font></b></td>
               <td>
                  <select name="estilo[]" multiple>
                     <option value="C">Clássico</option>
                     <option value="T">Contemporâneo</option>
                     <option value="A">Alternativo</option>
                     <option value="E">Elegante</option>
                  </select>
               </td>
            </tr>
            <tr>
               <td valign=top width="140px"><b><font face="Arial">Fotografia</font></b></td>
               <td><input name="fotografia" type="file"></td>
            </tr>
            <tr>
               <td valign=top width="140px"><b><font face="Arial">Curriculo</font></b></td>
               <td><textarea name="curriculo" rows="4" cols="40">digite o currículo aqui
                   </textarea></td>
            </tr>
         </table>
         <input type="submit" value="gravar">
         <input type="reset"  value="limpar">
         <input type="button" value="botao" onclick="alert('teste123')">
      </form>
   </body>
</html>
```

1.13.3 Método POST

Formulários podem ser submetidos para processamento por meio dos métodos `POST` ou `GET`. Tecnicamente a diferença básica entre os dois métodos é a forma como o transporte dos dados é efetuado. O método `GET` transfere os dados do formulário via URL, ao passo que o método `POST` encapsula os dados dentro do corpo da mensagem a ser transmitida, não sendo visível ao usuário.

O método `GET` pode ser utilizado sempre que o processamento do formulário for "idempotente", ou seja, o formulário poderá ser submetido diversas vezes sendo que o mesmo resultado é obtido, ou seja, não causa efeitos colaterais. Um exemplo disso é um formulário de consultas. Se optarmos pelo método `GET`, os campos que

utilizarmos como filtro da consulta serão expostos na URL e poderemos inclusive enviar a URL por email para um amigo e ele, ao acessá-la, visualizará os mesmos resultados da busca.

Em se tratando de gravar dados em bancos de dados, no qual um formulário poderá inserir ou mesmo alterar o estado de um registro do banco de dados cada vez que é acionado, causando efeitos colaterais, é recomendável utilizarmos o método `POST`, inclusive por questões de segurança, visto que um usuário mal-intencionado poderia passar dados inconsistentes via URL para causar algum problema no funcionamento do programa (se utilizarmos `GET`).

Como ação do formulário anterior, definimos o script *form_gravar.php*, cujo código é apresentado a seguir. O programa que receberá a ação do formulário já recebe por padrão o array `$_POST` contendo os dados do formulário. Para exibi-lo, utilizaremos a função `print_r()`. Você perceberá que os índices do array são exatamente os nomes dos campos.

No caso de arquivos enviados pelo campo `<input type=file>` o PHP disponibiliza o array `$_FILES`. Esses arquivos são enviados para o servidor e lá recebem um nome e uma localização temporária (veja [`tmp_name`]). Demais informações, como o tipo de arquivo e seu nome original, também são disponibilizadas por esse array.

form_gravar.php

```
<?php
echo "<pre>";
print_r($_POST);
print_r($_FILES);
echo "</pre>";
```

Resultado:

```
Array
(
    [nome] => Pablo Dall Oglio
    [senha] => 1234
    [sexo] => M
    [religiao] => C
    [idiomas] => Array
    (
        [0] => E
        [1] => I
    )
```

```
    [estilo] => Array
    (
        [0] => T
        [1] => E
    )
    [curriculo] => digite o currículo aqui
)

Array
(
    [fotografia] => Array
    (
        [name] => foto.png
        [type] => image/jpeg
        [tmp_name] => /tmp/phptlin9t
        [error] => 0
        [size] => 1722
    )
)
```

1.14 Listagens

Uma listagem geralmente é utilizada para apresentar um conjunto de registros de um banco de dados, apesar de se poder listar informações de praticamente qualquer fonte de dados, como arquivos XML, por exemplo. A apresentação de uma listagem segue uma forma tabular e utiliza-se com frequência de uma tabela para organizar visualmente a listagem. Uma listagem é composta de linhas, colunas e ações. Nas colunas distribuímos as informações que desejamos apresentar (Código, Nome, Telefone); nas linhas distribuímos cada um dos itens a exibir. Uma listagem normalmente apresenta ações ao usuário (editar, visualizar, excluir, imprimir), sendo essas ações executadas sobre cada um dos itens apresentados na listagem.

Observação: aprenderemos primeiramente a utilizar listagens no PHP de maneira estruturada para ao longo do livro construir classes que tornarão esse processo totalmente orientado a objetos.

1.14.1 Exemplo de listagem

No exemplo a seguir construiremos uma listagem que apresentará alguns dados de maneira estruturada. Para tal, mesclaremos algumas informações estáticas, como o cabeçalho da listagem (`table, tr`) com informações provindas do banco de dados. Para isso, abriremos uma conexão com o banco de dados (`pg_connect`), executaremos uma consulta (`pg_query`) e, em seguida, percorreremos o seu retorno (`pg_fetch_assoc`), exibindo uma linha de tabela `<tr>` para cada registro retornado, inserindo as variáveis dentro de cada `string`. Veja na figura 1.6 o resultado deste exemplo.

> **Observação:** nunca é demais ressaltar que essa técnica é apresentada aqui apenas por motivos didáticos. Não é uma boa prática escrever código-fonte que mescla PHP e HTML. Ao longo do livro, aprenderemos técnicas que permitem separar esses dois aspectos (lógica e apresentação).

	Código	Nome
Editar	1	Érico Veríssimo
Editar	2	John Lennon
Editar	3	Mahatma Gandhi
Editar	4	Ayrton Senna
Editar	5	Charlie Chaplin
Editar	6	Anita Garibaldi
Editar	7	Mário Quintana

Figura 1.6 – Listagem de dados.

list.php

```
<?php
// exibe início da tabela
echo '<table border=1 style="min-width:500px">
      <tr bgcolor=#c0c0c0>
        <td align="center"></td>
        <td align="center">Código</td>
        <td align="center">Nome</td>
      </tr>';

// abre conexão com Postgres
$conn = pg_connect("host=localhost port=5432 dbname=livro user=postgres");

// define consulta que será realizada
$query = 'select codigo, nome from famosos';
```

```
// envia consulta ao banco de dados
$result = pg_query($conn, $query);

if ($result) {
    // percorre resultados da pesquisa
    while ($row = pg_fetch_assoc($result)) {
        $codigo = $row['codigo'];
        $nome   = $row['nome'];

        // exibe uma linha de resultados
        echo "<tr>
                <td align='center'><a href='edit.php?codigo={$codigo}'> Editar </a></td>
                <td align='center'>{$codigo}</td>
                <td align='left'>{$nome}</td>
              </tr>";
    }
}
// fecha a conexão
pg_close($conn);

// imprime fechamento da tabela
echo '</table>';
```