PHP

Programando com Orientação a Objetos

Pablo Dall'Oglio

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 arquivos, arrays, bancos de dados, entre outros.

1.1 O que é o PHP?

A linguagem de programação PHP, cujo logotipo vemos na Figura 1.1, foi criada no outono de 1994 por Rasmus Lerdorf. No início era formada por um conjunto de scripts voltados à criação de páginas dinâmicas que Rasmus utilizava para monitorar o acesso ao seu currículo na internet. À medida que essa ferramenta foi crescendo em funcionalidades, Rasmus teve de escrever uma implementação em C, a qual permitia às pessoas desenvolverem de forma muito simples suas aplicações para web. Rasmus nomeou essa versão de PHP/FI (Personal Home Pages/Forms Interpreter) e decidiu disponibilizar seu código na web, em 1995, para compartilhar com outras pessoas, bem como receber ajuda e correção de bugs.

Em novembro de 1997 foi lançada a segunda versão do PHP. Naquele momento, aproximadamente 50 mil domínios ou 1% da internet já utilizava PHP. No mesmo ano, Andi Gutmans e Zeev Suraski, dois estudantes que utilizavam PHP 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. Dentre 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.

Figura 1.1 – Logo do PHP.

No inverno de 1998, após o lançamento do PHP 3, Zeev e Andi começaram a trabalhar em uma reescrita do núcleo do PHP, tendo em vista melhorar sua performance e modularidade em aplicações complexas. Para tanto, resolveram batizar este núcleo de *Zend Engine*, ou Mecanismo Zend (Zeev + Andi). O PHP 4, baseado neste mecanismo, foi lançado oficialmente em maio de 2000, trazendo muitas melhorias e recursos novos, como seções, suporte a diversos servidores web, além da abstração de sua API, permitindo inclusive ser utilizado como linguagem para shell script. Nesse momento, o PHP já estava presente em cerca de 20% dos domínios da internet, além de ser utilizado por milhares de desenvolvedores ao redor do mundo.

Apesar de todos os esforços, o PHP ainda necessitava maior suporte à orientação a objetos, tal qual existe em linguagens como C++ e Java. Tais recursos estão finalmente presentes no PHP 5, após um longo período de desenvolvimento que culminou com sua disponibilização oficial em julho de 2004. Ao longo do livro veremos esses recursos empregados em exemplos práticos.

Fonte: PHP Group

1.2 Um programa PHP

1.2.1 Extensão de arquivos

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

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

Entretanto, outras extensões podem ser encontradas principalmente em programas antigos:

Extensão	Significado
`.php3`	Arquivo PHP contendo um programa PHP versão 3.
`.php4`	Arquivo PHP contendo um programa PHP versão 4.
`.phtml`	Arquivo PHP contendo um programa PHP e HTML na mesma página.

1.2.2 Delimitadores de código

O código de um programa escrito em PHP deve estar contido entre os seguintes delimitadores:

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

Observação: os comandos sempre são delimitados por ponto-e-vírgula (;).

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)

Estes 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 ou IIS), a saída será exibida na própria página HTML gerada.

echo

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

```
echo 'a', 'b', 'c';
```

💻 **Resultado:**

```
abc
```

print

É uma função que imprime uma string no console. Exemplo:

```
print('abc');
```

💻 **Resultado:**

```
abc
```

var_dump

Imprime o conteúdo de uma variável de forma explanativa, 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

Imprime o conteúdo de uma variável de forma explanativa, 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 idéia de seu conteúdo dentro do contexto no qual a variável está inserida.
- Utilize preferencialmente palavras em minúsculo (separadas pelo caractere "_") ou somente as primeiras letras em maiúsculo quando da ocorrência de mais palavras.

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

O PHP é *case sensitive*, ou seja, é sensível a letras maiúsculas e minúsculas. Tome cuidado ao declarar variáveis e nomes de função. 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, sempre é criada uma nova área de armazenamento na memória. 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 $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 $b;   // resultado = 10
?>
```

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 desta variável para verificar se ela é realmente verdadeira. Caso positivo, será exibido na tela o nome "José da Silva".

```
<?php
// declara variável com valor TRUE
$exibir_nome = 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 boolean que será testado logo em seguida para a impressão da string "Está chovendo".

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

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

// testa se $vai_chover é verdadeiro
if ($vai_chover)
{
   echo 'Está chovendo';
}
?>
```

Resultado:

```
Está chovendo
```

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), opcionalmente precedido de sinal (- ou +).

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

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 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, os quais 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 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`. Veja o exemplo de um objeto do tipo `Computador` e aprenda, no Capítulo 2, como manipular classes e objetos.

```
<?php
class Computador
{
    var $cpu;
    function ligar()
    {
        echo "Ligando computador a {$this->cpu}...";
    }
}

$obj = new Computador;
$obj->cpu = "500Mhz";
$obj->ligar();
?>
```

Resultado:

```
Ligando computador a 500Mhz...
```

1.3.6 Tipo recurso

Recurso (`resource`) é uma variável especial que mantém uma referência de recurso externo. Recursos são criados e utilizados por funções especiais, como uma conexão ao banco de dados. Um exemplo é a função `mysql_connect()`, que, ao conectar-se ao banco de dados, retorna uma variável de referência do tipo recurso.

```
resource mysql_connect(...)
```

1.3.7 Tipo misto

O tipo misto (`mixed`) representa múltiplos (não necessariamente todos) tipos de dados 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()`, a qual 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. Este 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, representados 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 (boolean, inteiro, ponto flutuante e string). Um valor escalar não pode ser composto de outros valores, como vetores ou objetos. 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-se nomes em maiúsculo.

```
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 uma variável atribuindo-lhe um valor. O operador básico de atribuição é `=`.

```
<?php
$var += 5;     // Soma 5 em $var;
$var -= 5;     // Subtrai 5 em $var;
$var *=5;      // Multiplica $var por 5;
$var /= 5;     // Divide $var por 5;
?>
```

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

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 expressões forem iguais.
===	Idêntico. Resulta verdadeiro (`TRUE`) se as expressões forem iguais e do mesmo tipo de dados.
!= ou <>	Diferente. Resulta verdadeiro 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 e retorna verdadeiro';
}
?>
```

Resultado:

```
essa operação atribui 5 à variável $a e retorna verdadeiro
```

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:

```
<?
$e = 0;

// zero sempre é igual à FALSE
if ($e == FALSE)
{
   echo '$e é falso';
}

// testa se $e é do tipo FALSE
if ($e === FALSE)
{
   echo '$e é do tipo false';
}

// 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 maior 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 a seguir um fluxograma explicando o funcionamento do comando IF. Caso a avaliação da expressão seja verdadeira, o programa parte para execução de um bloco de comandos; caso seja falsa, parte para a execução do bloco de comandos dada pelo ELSE.

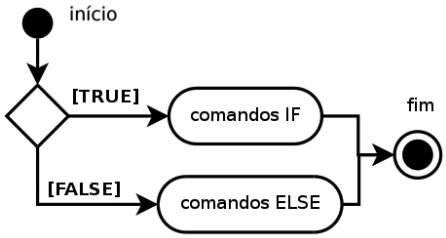

Figura 1.2 – Fluxo do comando IF.

```
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
```

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

```
<?php
$salario         = 1020;     // R$
$tempo_servico   = 12;       // meses
$tem_reclamacoes = false;    // boleano
if ($salario > 1000)
{
   if ($tempo_servico >= 12)
   {
       if ($tem_reclamacoes != true)
       {
           echo 'parabéns, você foi promovido';
       }
   }
}

if (($salario > 1000) and ($tempo_servico >= 12) and ($tem_reclamacoes != true))
{
   echo 'parabéns, você foi promovido';
}
?>
```

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, possui 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. Este 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. Quando do fim da execução deste 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`).

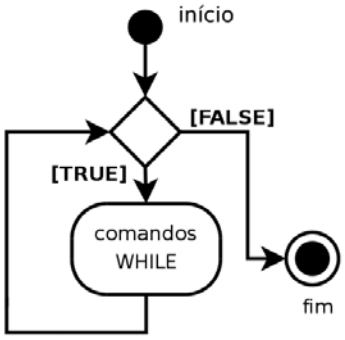

Figura 1.3 – Fluxo do comando `WHILE`.

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

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 esta 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, esta 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 for 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. Freqüentemente, é necessário comparar a mesma variável com valores diferentes e executar uma ação específica em cada um destes casos.

No fluxograma a seguir 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`.

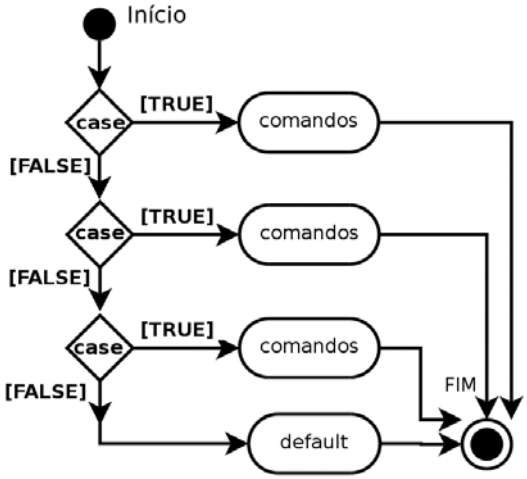

Figura 1.4 – Fluxo do comando SWITCH.

Sintaxe do comando:

```
switch ($expressão)
{
    case "valor 1":
        // comandos
        break;
    case "valor 2":
        // comandos
        break;
    case "valor n";
        // 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 matriz em cada um de seus elementos por meio de sua cláusula AS.

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

Exemplo:

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

Resultado:

```
maçã - laranja - pêra - 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 o 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, freqüentemente 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 este 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:

biblioteca.php

```
<?php
/*
 * função quadrado
 * retorna o quadrado de um número
 */
function quadrado($numero)
{
    return $numero * $numero;
}
?>
```

teste.php

```
<?php
// carrega arquivo com a função necessária
include 'biblioteca.php';

// imprime o quadrado do número 4
echo quadrado(4);
?>
```

Resultado:

```
16
```

require <*arquivo*>

Idêntico ao `include`. Difere somente na manipulação de erros. Enquanto o include produz uma warning, o require produz uma mensagem de Fatal Error 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, não refazendo a operação (o arquivo é incluído apenas uma vez). Este comando é útil em casos em que o programa pode passar 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, não refazendo a operação (o arquivo é incluído apenas uma vez). Este comando é útil em casos em que o programa pode passar mais de uma vez pela mesma instrução. Assim, poderá-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 lhe atribuir, sem espaços em branco e iniciando obrigatoriamente com uma letra. Na mesma linha, digitamos a lista de argumentos (parâmetros) que a função irá receber, separados por vírgula. Em seguida, encapsulado por chaves {}, vem o código da função. No final, utiliza-se a cláusula `return` para retornar o resultado da função (integer, string, array, objeto etc.).

```
<?php

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

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

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

Resultado:

```
20.45288531775
```

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 \n";
echo 'percorreu ' . km2mi(200) . " milhas \n";
echo 'percorreu no total ' . $total . " quilometros \n";
?>
```

Resultado:

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

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\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, vistos no Capítulo 2, são a exceção.

```
<?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, fazendo 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] . "\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) . "\n";
echo Fatorial(7) . "\n";
?>
```

Resultado:

```
120
5040
```

1.9 Manipulação de arquivos e diretórios

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

fopen

Abre um arquivo e retorna um identificador. Se o nome do arquivo está na forma "protocolo://...", o PHP irá procurar por um manipulador de protocolo, também conhecido como wrapper, conforme o prefixo.

```
int fopen (string arquivo, string modo [,int usar_path [, resource contexto]])
```

Parâmetros	Descrição
arquivo	String identificando o nome do arquivo a ser aberto.
modo	Forma de abertura do arquivo (r=read, w=write, a=append).
usar_path	Se 1 ou TRUE, vasculha a path pelo arquivo a ser aberto.
contexto	Opções de contexto; variam de acordo com o protocolo do arquivo.

Exemplo:

```
<?php
$fp = fopen ("/home/pablo/file.txt", "r");
$fp = fopen ("/home/pablo/file.gif", "wb");
$fp = fopen ("http://www.example.com/", "r");
$fp = fopen ("ftp://user:password@example.com/", "w");
?>
```

feof

Testa se um determinado identificador de arquivo (criado pela função `fopen()`) está no fim de arquivo (End Of File). Retorna `TRUE` se o ponteiro estiver no fim do arquivo (EOF); do contrário, retorna `FALSE`.

```
int feof (int identificador)
```

Parâmetro	Descrição
identificador	Parâmetro retornado pela `fopen()`.

fgets

Lê uma linha de um arquivo. Retorna uma string com até (tamanho – 1) bytes lidos do arquivo apontado pelo identificador de arquivo. Se nenhum tamanho for informado, o default é 1Kb ou 1024 bytes. Se um erro ocorrer, retorna `FALSE`.

```
string fgets (int identificador [, int tamanho])
```

Parâmetros	Descrição
identificador	Parâmetro retornado pela `fopen()`.
tamanho	Quantidade em bytes a retornar da leitura.

Exemplo:

```
<?php
$fd = fopen ("/etc/fstab", "r");
while (!feof ($fd))
{
    // lê uma linha do arquivo
    $buffer = fgets($fd, 4096);

    // imprime a linha.
    echo $buffer;
}
fclose ($fd);
?>
```

Resultado:

```
/dev/hda2       swap          swap       defaults          0   0
/dev/hda3       /             ext3       defaults          1   1
/dev/hda1       /windows      ntfs       defaults          1   0
/dev/cdrom      /mnt/cdrom    iso9660    noauto,owner,ro   0   0
/dev/fd0        /mnt/floppy   auto       noauto,owner      0   0
devpts          /dev/pts      devpts     gid=5,mode=620    0   0
proc            /proc         proc       defaults          0   0
```

fwrite

Grava uma string (*conteúdo*) no arquivo apontado pelo *identificador* de arquivo. Se o argumento *comprimento* é dado, a gravação irá parar depois que *comprimento* bytes for escrito ou o fim da string *conteúdo* for alcançado, o que ocorrer primeiro. Retorna o número de bytes gravados, ou `FALSE` em caso de erro.

```
int fwrite (int identificador, string conteúdo [, int comprimento])
```

Parâmetros	Descrição
identificador	Parâmetro retornado pela `fopen()`.
conteúdo	String a escrever no arquivo.
comprimento	Comprimento da string.

Exemplo:

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

// escreve no arquivo
fwrite ($fp, "linha 1\n");
fwrite ($fp, "linha 2\n");
fwrite ($fp, "linha 3\n");

// fecha o arquivo
fclose ($fp);
?>
```

fclose

Fecha o arquivo aberto apontado pelo identificador de arquivo. Retorna `TRUE` em caso de sucesso ou `FALSE` em caso de falha.

```
bool fclose (int identificador)
```

Parâmetro	Descrição
identificador	Parâmetro retornado pela `fopen()`.

file_put_contents

Grava uma string em um arquivo. Retorna a quantidade de bytes gravados.

```
int file_put_contents (string nome_arquivo, mixed conteúdo)
```

Parâmetros	Descrição
nome_arquivo	Arquivo a ser aberto.
conteúdo	Novo conteúdo.

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 nome_arquivo, ...)
```

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 todo o seu conteúdo, de modo que cada posição do array representa uma linha lida do arquivo. O nome do arquivo pode conter o protocolo, como no caso http://www.servidor.com.br/pagina.html. Assim, o arquivo remoto será lido para dentro do array.

```
array file (string nome_arquivo, [int usar_include_path])
```

Parâmetros	Descrição
nome_arquivo	Arquivo a ser lido.
usar_include_path	Se "1", procura também nos diretórios da constante PHP_INCLUDE_PATH.

Exemplo:

```
<?php
// lê o arquivo para o array $arquivo
$arquivo = file ("/home/pablo/file.txt");
// exibe o conteúdo
echo $arquivo[0];
echo $arquivo[1];
echo $arquivo[2];
?>
```

Resultado:

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

copy

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

```
bool copy (string arquivo_origem, string arquivo_destino)
```

Parâmetros	Descrição
arquivo_origem	Arquivo a ser copiado.
arquivo_destino	Arquivo destino.

Exemplo:

```
<?php
$origem  = "/home/pablo/file.txt";
$destino = "/home/pablo/file2.txt";
if (copy($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 arquivo_origem, string arquivo_destino)
```

Parâmetros	Descrição
arquivo_origem	Arquivo a ser renomeado.
arquivo_destino	Arquivo destino.

Exemplo:

```
<?php
$origem  = "/home/pablo/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 nome_arquivo)
```

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 nome_arquivo)
```

Parâmetros	Descrição
nome_arquivo	Localização de um arquivo ou diretório.

Exemplo:

```
<?php
$arquivo   = '/home/pablo/file2.txt';
if (file_exists($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 localização)
```

Parâmetros	Descrição
localização	Localização de um arquivo ou diretório.

Exemplo:

```
<?php
$arquivo = '/home/pablo/file.txt';
if (is_file($arquivo))
{
   echo "$arquivo é um arquivo";
}
else
{
```

```
    echo "$arquivo não é um arquivo";
}
?>
```

Resultado:

```
/home/pablo/file.txt é um arquivo
```

mkdir

Cria um diretório.

```
bool mkdir (string localização, [int modo])
```

Parâmetros	Descrição
localização	Localização de um diretório.
modo	Permissão de acesso.

Exemplo:

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

Resultado:

```
/tmp/diretorio criado com sucesso
```

getcwd

Retorna o diretório corrente.

```
string getcwd ()
```

Exemplo:

```
<?php
echo 'o diretório atual é ' . getcwd();
?>
```

Resultado:

```
o diretório atual é /tmp
```

chdir

Altera o diretório corrente. Retorna `TRUE` em caso de sucesso e `FALSE` em caso de falhas.

```
bool chdir (string localização)
```

Parâmetros	Descrição
localização	Localização de um diretório.

Exemplo:

```
<?php
echo 'o diretório atual é ' . getcwd();
chdir('/home/pablo');
echo 'o diretório atual é ' . getcwd();
?>
```

Resultado:

```
o diretório atual é /tmp
o diretório atual é /home/pablo
```

rmdir

Apaga um diretório.

```
bool rmdir (string localização)
```

Parâmetros	Descrição
localização	Localização de um diretório.

Exemplo:

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

Resultado:

```
/tmp/diretorio apagado com sucesso
```

opendir

Abre um diretório e retorna um identificador.

```
resource opendir (string nome_diretorio)
```

Parâmetros	Descrição
nome_arquivo	String identificando o nome do diretório a ser aberto.

closedir

Libera um recurso alocado pela função `opendir()`.

```
void closedir (resource identificador)
```

Parâmetros	Descrição
identificador	Identificador retornado pela função `opendir()`.

readdir

Realiza a leitura do conteúdo de um diretório por meio do identificador criado pela função `opendir()`.

```
string readdir (resource identificador)
```

Parâmetros	Descrição
identificador	Identificador retornado pela função `opendir()`.

Exemplo:

```
<?php
// exibe as entradas do diretório raiz
$diretorio = '/';
// verifica se é diretório.
if (is_dir($diretorio))
{
    $ident = opendir($diretorio);
    // laço de repetição para leitura.
    while ($arquivo = readdir($ident))
    {
        echo $arquivo . "\n";
    }
    closedir($ident);
}
?>
```

Resultado:

```
.
..
```

```
var
dev
bin
etc
lib
usr
boot
home
```

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 contido dentro de aspas duplas é avaliado pelo PHP. Assim, se a string contém uma variável, esta variável será traduzida pelo seu valor.

```
<?php
$fruta = 'maçã';
print "como $fruta";      // resultado 'como maçã'
print 'como $fruta';      // 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 "", uma vez que seu conteúdo é interpretado.

```
<?php
$fruta = 'maçã';

// primeira forma
echo $fruta .' é a fruta de adão';   // resultado = maçã é a fruta de adão

// segunda forma
echo "{$fruta} é a fruta de adão";   // 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 . "\n";
echo "O salário é $a \n";
?>
```

Resultado:

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

1.10.3 Caracteres de escape

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

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

Exemplo:

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

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

strtoupper

Transforma uma string (conteúdo) para maiúsculo. Retorna a string com todos os caracteres alfabéticos convertidos para maiúsculo.

```
string strtoupper (string conteúdo)
```

Exemplo:

```
<?php
echo strtoupper('Convertendo para maiúsculo');
?>
```

Resultado:

```
CONVERTENDO PARA MAIÚSCULO
```

strtolower

Transforma uma string (*conteúdo*) para minúsculo. Retorna a string com todos os caracteres alfabéticos convertidos para minúsculo.

```
string strtolower (string conteúdo)
```

Parâmetros	Descrição
conteúdo	String original a ser transformada.

Exemplo:

```
<?php
echo strtolower('CONVERTENDO PARA MINÚSCULO');
?>
```

Resultado:

```
convertendo para minúsculo
```

substr

Retorna parte de uma string (conteúdo). Retorna uma porção de conteúdo, começando em *início*, contendo *comprimento* caracteres. Se *comprimento* for negativo, conta *n* caracteres antes do final.

```
string substr (string conteúdo, int início [, int comprimento])
```

Parâmetros	Descrição
conteúdo	String original a ser percorrida.
início	Caractere inicial a ser lido.
comprimento	Comprimento da cadeia de caracteres a ser lida.

Exemplo:

```
<?php
$rest = substr("América", 1);
echo $rest . "\n";
$rest = substr("América", 1, 3);
echo $rest . "\n";
$rest = substr("América", 0, -1);
echo $rest . "\n";
$rest = substr("América", -2);
echo $rest . "\n";
?>
```

Resultado:

```
mérica
mér
América
ca
```

strpad

Preenche uma string com uma outra string, dentro de um tamanho específico.

```
string str_pad ( string entrada, int tamanho [, string complemento [, int tipo]])
```

Parâmetros	Descrição
entrada	String inicial a ser complementada.
tamanho	Comprimento da string a ser retornada.
complemento	String de preenchimento.
tipo	Tipo de preenchimento. Pode ser `STR_PAD_RIGHT` (preenche com caracteres à direita), `STR_PAD_LEFT` (preenche à esquerda) ou `STR_PAD_BOTH` (preenche em ambos os lados).

Exemplo:

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

Resultado:

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

str_repeat

Repete uma string uma certa quantidade de vezes.

```
string str_repeat ( string entrada, int quantidade)
```

Parâmetros	Descrição
entrada	String inicial a ser repetida.
quantidade	Quantidade de repetições.

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 entrada )
```

Parâmetros	Descrição
entrada	String cujo comprimento será calculado.

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 por outra em um dado contexto.

```
mixed str_replace ( mixed procura, mixed substitui, mixed contexto)
```

Parâmetros	Descrição
procura	String a ser substituída.
substitui	String substituta.
contexto	String inicial a ser submetida à substituição.

Exemplo:

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

Resultado:

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

strpos

Encontra a primeira ocorrência de uma string dentro de outra.

```
int strpos (string principal, string procurada [, int offset])
```

Parâmetros	Descrição
principal	String qualquer.
procurada	String a ser procurada dentro da string principal.
offset	Quantidade de caracteres a ser ignorada.

No exemplo a seguir, a função `strpos()` vasculha a variável `$minha_string` para encontrar em qualquer posição dentro dela a variável `$encontrar`:

```
<?php
$minha_string = 'O rato roeu a roupa do rei de Roma';
$encontrar = 'roupa';
$posicao = strpos($minha_string, $encontrar);
if ($posicao)
{
    echo "String encontrada na posição $posicao";
}
else
{
    echo "String não encontrada";
}
?>
```

Resultado:

```
String encontrada na posição 14
```

1.11 Manipulação de arrays

A manipulação de arrays no PHP é, sem dúvida, um dos recursos mais poderosos da linguagem. O programador que assimilar bem esta parte terá muito mais produtividade no seu dia-a-dia. Isto porque os arrays no PHP servem como verdadeiros contêineres, servindo para armazenar números, strings, objetos, dentre outros, de forma dinâmica. Além disso, o PHP nos oferece uma gama enorme 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 , ... )`.

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

ou

```
$cores = array(0=>'vermelho', 1=>'azul', 2=>'verde', 3=>'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['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, percorrendo cada uma das posições do array. Exemplo:

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

Resultado:

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

1.11.4 Acesso

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

```
<?php
$minha_multa['carro'] = 'Pálio';
$minha_multa['valor'] = 178.00;

// alteração de valores
$minha_multa['carro'] .= ' ED 1.0';
$minha_multa['valor'] += 20;

// exibe o array
var_dump($minha_multa);

$comidas[] = 'Lazanha';
$comidas[] = 'Pizza';
$comidas[] = 'Macarrão';

// alteração de valores
$comidas[1] = 'Pizza Calabreza';

// exibe o array
var_dump($comidas);
?>
```

Resultado:

```
array(2) {
    ["carro"]=>
    string(12) "Pálio ED 1.0"
    ["valor"]=>
    float(198)
}
array(3) {
    [0]=>
    string(7) "Lazanha"
    [1]=>
    string(15) "Pizza Calabreza"
    [2]=>
    string(8) "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['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 possui. 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.

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

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

array_push

Adiciona elementos ao final de um array. Tem o mesmo efeito de utilizar a sintaxe `$array[] = $valor`.

```
int array_push (array nome_array, mixed valor [, mixed ...])
```

Parâmetros	Descrição
nome_array	Array a ser acrescido do valor.
valor	Valor a ser adicionado.
...	Pode-se adicionar *n* valores.

Exemplo:

```
<?php
$a = array("verde", "azul", "vermelho");
array_push($a, "amarelo");
var_dump($a);
?>
```

Resultado:

```
array(4) {
  [0]=> string(5) "verde"
  [1]=> string(4) "azul"
  [2]=> string(8) "vermelho"
  [3]=> string(7) "amarelo"
}
```

array_pop

Remove um valor do final de um array.

```
mixed array_pop (array nome_array)
```

Exemplo:

```
<?php
$a = array("verde", "azul", "vermelho", "amarelo");
array_pop($a);
var_dump($a);
?>
```

Resultado:

```
array(3) {
  [0]=>  string(5) "verde"
  [1]=>  string(4) "azul"
  [2]=>  string(8) "vermelho"
}
```

array_shift

Remove um elemento do início de um array.

```
mixed array_shift (array nome_array)
```

Exemplo:

```
<?php
$a = array("verde", "azul", "vermelho", "amarelo");
array_shift($a);
var_dump($a);
?>
```

Resultado:

```
array(3) {
  [0]=>  string(4) "azul"
  [1]=>  string(8) "vermelho"
  [2]=>  string(7) "amarelo"
}
```

array_unshift

Adiciona um elemento no início de um array.

```
int array_unshift (array nome_array, mixed valor [, mixed ...])
```

Parâmetros	Descrição
nome_array	Array a ser acrescido do valor.
valor	Valor a ser adicionado.
...	Pode-se adicionar *n* valores.

Exemplo:

```
<?php
$a = array("verde", "azul", "vermelho");
array_unshift($a, "amarelo");
var_dump($a);
?>
```

Resultado:

```
array(4) {
  [0]=>  string(7) "amarelo"
  [1]=>  string(5) "verde"
  [2]=>  string(4) "azul"
  [3]=>  string(8) "vermelho"
}
```

array_pad

Preenche um array com um dado valor, determinada quantidade de posições.

```
array array_pad (array nome_array, int tamanho, mixed valor)
```

Parâmetros	Descrição
nome_array	Array a ser preenchido.
tamanho	Quantidade de posições.
valor	Valor a ser preenchido.

Exemplo:

```
<?php
$a = array("verde", "azul", "vermelho");
$a = array_pad($a, 6, "branco");
var_dump($a);
?>
```

Resultado:

```
array(6) {
    [0]=>  string(5) "verde"
    [1]=>  string(4) "azul"
    [2]=>  string(8) "vermelho"
    [3]=>  string(6) "branco"
    [4]=>  string(6) "branco"
    [5]=>  string(6) "branco"
}
```

array_reverse

Recebe um array e retorna-o na ordem inversa.

```
array array_reverse (array nome_array, boolean preservar_chaves)
```

Parâmetros	Descrição
nome_array	Array a ser revertido.
preservar_chaves	Manter a associação de índices.

Exemplo:

```
<?php
$a[0] = 'green';
$a[1] = 'yellow';
$a[2] = 'red';
$a[3] = 'blue';
$b = array_reverse($a, true);
var_dump($b);
?>
```

Resultado:

```
array(4) {
    [3]=>  string(4) "blue"
    [2]=>  string(3) "red"
    [1]=>  string(6) "yellow"
    [0]=>  string(5) "green"
}
```

array_merge

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

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

Parâmetros	Descrição
nome_array1	Primeiro array a ser mesclado.
nome_array2	Segundo array a ser mesclado.
...	Pode-se mesclar *n* arrays.

Exemplo:

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

Resultado:

```
array(4) {
    [0]=>  string(5) "verde"
    [1]=>  string(4) "azul"
    [2]=>  string(8) "vermelho"
    [3]=>  string(7) "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 nome_array [, mixed valor_procurado])
```

Parâmetros	Descrição
nome_array	Array cujos índices desejamos descobrir.
valor_procurado	Parâmetro opcional. Se preenchido, retornará apenas índices contendo este valor.

Exemplo:

```
<?php
$exemplo = array('cor' => 'vermelho', 'volume' => 5, 'animal'=>'cachorro');
$indices = array_keys($exemplo);
print_r($indices);
?>
```

Resultado:

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

array_values

Retorna um array contendo os valores de outro array.

```
array array_values (array nome_array)
```

Parâmetros	Descrição
nome_array	Array cujos valores desejamos descobrir.

Exemplo:

```
<?php
$exemplo = array('cor' => 'vermelho', 'volume' => 5, 'animal'=>'cachorro');
$valores = array_values($exemplo);
print_r($valores);
?>
```

Resultado:

```
Array
(
    [0] => vermelho
    [1] => 5
    [2] => cachorro
)
```

array_slice

Extrai uma porção de um array.

```
array array_slice (array nome_array, int início, int tamanho)
```

Parâmetros	Descrição
nome_array	Array cuja porção desejamos extrair.
início	Primeira posição a ser extraída.
tamanho	Tamanho da porção extraída.

Exemplo:

```
<?php
$a[0] = 'green';
$a[1] = 'yellow';
$a[2] = 'red';
$a[3] = 'blue';
$a[4] = 'gray';
$a[5] = 'white';
$b = array_slice($a, 2, 3);
print_r($b);
?>
```

Resultado:

```
Array
(
    [0] => red
    [1] => blue
    [2] => gray
)
```

count

Retorna a quantidade de elementos de um array.

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

Exemplo:

```
<?php
$a = array('refrigerante', 'cerveja', 'vodka', 'suco natural');
echo 'o array $a contém ' . count($a) . ' posições';
?>
```

Resultado:

```
o array $a contém 4 posições
```

array_in

Verifica se um array contém um determinado valor.

```
boolean array_in (mixed valor, array nome_array)
```

Parâmetro	Descrição
valor	Valor a ser procurado.
nome_array	Array a ser vasculhado.

Exemplo:

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

Resultado:

```
suco natural encontrado
```

sort

Ordena um array pelo seu valor, não mantendo a associação de índices.

```
boolean sort (array nome_array)
```

Exemplo:

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

Resultado:

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

rsort

Ordena um array em ordem reversa pelo seu valor, não mantendo a associação de índices.

```
boolean rsort (array nome_array)
```

Exemplo:

```
<?php
$a = array('refrigerante', 'cerveja', 'vodka', 'suco natural');
rsort($a);
print_r($a);
?>
```

Resultado:

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

asort

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

```
void asort (array nome_array)
```

Exemplo:

```
<?php
$a[0] = 'green';
$a[1] = 'yellow';
$a[2] = 'red';
$a[3] = 'blue';
$a[4] = 'gray';
$a[5] = 'white';
asort($a);
print_r($a);
?>
```

Resultado:

```
Array
(
```

```
    [3] => blue
    [4] => gray
    [0] => green
    [2] => red
    [5] => white
    [1] => yellow
)
```

ksort

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

```
boolean ksort (array nome_array)
```

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, separando os elementos por meio de um separador.

```
array explode (string separador, string padrão)
```

Parâmetros	Descrição
separador	Caractere que será utilizado para desmembrar a string, convertendo-a em um array.
padrão	String que desejamos converter em um array.

Exemplo:

```
<?php
$string = "31/12/2004";
var_dump(explode("/", $string));
?>
```

Resultado:

```
array(3) {
    [0]=>
    string(2) "31"
    [1]=>
    string(2) "12"
    [2]=>
    string(4) "2004"
}
```

implode

Converte um array em uma string, separando os elementos do array por meio de um separador.

```
array implode (string separador, array padrão)
```

Parâmetros	Descrição
separador	Caractere que será utilizado para delimitar a nova string criada.
padrão	Array original que desejamos converter em string.

Exemplo:

```
<?php
$padrao = array('Maria', 'Paulo', 'José');
$resultado = implode(' + ', $padrao);

var_dump($resultado);
?>
```

Resultado:

```
string(20) "Maria + Paulo + José"
```

1.12 Manipulação de objetos

Nesta seção, veremos uma série de funções relacionadas à manipulação de objetos. Para maiores detalhes sobre orientação a objetos, veja o Capítulo 2.

get_class_methods

Retorna um vetor com os nomes dos métodos de uma determinada classe.

```
array get_class_methods (string nome_classe)
```

Exemplo:

```
<?php
class Funcionario
{
    function SetSalario()
    {
    }
    function GetSalario()
    {
    }
    function SetNome()
    {
    }
    function GetNome()
    {
    }
}

print_r(get_class_methods('Funcionario'));
?>
```

Resultado:

```
Array
(
    [0] => SetSalario
    [1] => GetSalario
    [2] => SetNome
    [3] => GetNome
)
```

get_class_vars

Retorna um vetor com os nomes das propriedades e conteúdos de uma determinada classe. Note que são valores estáticos definidos na criação da classe.

```
array get_class_vars (string nome_classe)
```

Exemplo:

```
<?php
class Funcionario
{
    var $Codigo;
    var $Nome;
    var $Salario = 586;
    var $Departamento = 'Contabilidade';

    function SetSalario()
    {
    }
    function GetSalario()
    {
    }
}

print_r(get_class_vars('Funcionario'));
?>
```

Resultado:

```
Array
(
    [Codigo] =>
    [Nome] =>
    [Salario] => 586
    [Departamento] => Contabilidade
)
```

get_object_vars

Retorna um vetor com os nomes e conteúdos das propriedades de um objeto. São valores dinâmicos que se alteram de acordo com o ciclo de vida do objeto.

```
array get_object_vars (object nome_objeto)
```

Exemplo:

```
<?php
class Funcionario
{
    var $Codigo;
    var $Nome;
    var $Salario = 586;
    var $Departamento = 'Contabilidade';
    function SetSalario()
    {
    }
```

```
    function GetSalario()
    {
    }
}
$jose = new Funcionario;
$jose->Codigo = 44;
$jose->Nome = 'José da Silva';
$jose->Salario += 100;
$jose->Departamento = 'Financeiro';

print_r(get_object_vars($jose));
?>
```

Resultado:

```
Array
(
    [Codigo] => 44
    [Nome] => José da Silva
    [Salario] => 686
    [Departamento] => Financeiro
)
```

get_class

Retorna o nome da classe a qual um objeto pertence.

```
string get_class (object nome_objeto)
```

Exemplo:

```
<?php
class Funcionario
{
    var $Codigo;
    var $Nome;

    function SetSalario()
    {
    }
    function GetSalario()
    {
    }
}

$jose = new Funcionario;
echo get_class($jose);
?>
```

Resultado:

```
Funcionario
```

get_parent_class

Retorna o nome da classe ancestral (classe-pai). Se o parâmetro for um objeto, retorna o nome da classe ancestral da classe à qual o objeto pertence. Se o parâmetro for uma string, retorna o nome da classe ancestral da classe passada como parâmetro.

```
string get_parent_class (mixed objeto)
```

Parâmetros	Descrição
objeto	Objeto ou nome de uma classe.

Exemplo:

```
<?php
class Funcionario
{
   var $Codigo;
   var $Nome;
}

class Estagiario extends Funcionario
{
   var $Salario;
   var $Bolsa;
}

$jose = new Estagiario;

echo get_parent_class($jose);
echo "\n";     // quebra de linha
echo get_parent_class('estagiario');
?>
```

Resultado:

```
Funcionario
Funcionario
```

is_subclass_of

Indica se um determinado objeto ou classe é derivado de uma determinada classe.

```
boolean is_subclass_of (mixed objeto, string classe)
```

Parâmetros	Descrição
objeto	Objeto ou nome de uma classe.
classe	Nome de uma classe ancestral.

Exemplo:

```
<?php
class Funcionario
{
   var $Codigo;
   var $Nome;
}

class Estagiario extends Funcionario
{
   var $Salario;
   var $Bolsa;
}

$jose = new Estagiario;

if (is_subclass_of($jose, 'Funcionario'))
{
   echo "Classe do objeto Jose é derivada de Funcionario";
}

echo "\n";     // quebra de linha

if (is_subclass_of('Estagiario', 'Funcionario'))
{
   echo "Classe Estagiario é derivada de Funcionario";
}
?>
```

Resultado:

```
Classe do objeto Jose é derivada de Funcionario
Classe Estagiario é derivada de Funcionario
```

method_exists

Verifica se um determinado objeto possui o método descrito. Podemos verificar a existência de um método antes de executar por engano um método inexistente.

```
boolean method_exists (object objeto, string método)
```

Parâmetros	Descrição
objeto	Objeto qualquer.
método	Nome de um método do objeto.

Exemplo:

```
<?php
class Funcionario
{
    var $Codigo;
    var $Nome;

    function GetSalario()
    {
    }

    function SetSalario()
    {
    }
}
$jose = new Funcionario;

if (method_exists($jose, 'SetNome'))
{
    echo 'Objeto Jose possui método SetNome()';
}
if (method_exists($jose, 'SetSalario'))
{
    echo 'Objeto Jose possui método SetSalario()';
}
?>
```

Resultado:

```
Objeto Jose possui método SetSalario()
```

call_user_func

Executa uma função ou um método de uma classe passado como parâmetro. Para executar uma função, basta passar seu nome como uma string, e, para executar um método de um objeto, basta passar o parâmetro como um array contendo na posição 0 o objeto e na posição 1 o método a ser executado. Para executar métodos estáticos, basta passar o nome da classe em vez do objeto na posição 0 do array.

```
mixed call_user_func (callback função [, mixed parâmetro [, mixed ...]])
```

Parâmetros	Descrição
função	Função a ser executada.
parâmetro	Parâmetro(s) da função.

Exemplo:

```
<?php
// exemplo chamada simples
function minhafuncao()
{
    echo "minha função! \n";
}

call_user_func('minhafuncao');

// declaração de classe
class MinhaClasse
{
    function MeuMetodo()
    {
        echo "Meu método! \n";
    }
}

// chamada de método estático
call_user_func(array('MinhaClasse', 'MeuMetodo'));

// chamada de método
$obj = new MinhaClasse();
call_user_func(array($obj, 'MeuMetodo'));
?>
```

Resultado:

```
minha função!
Meu método!
Meu método!
```