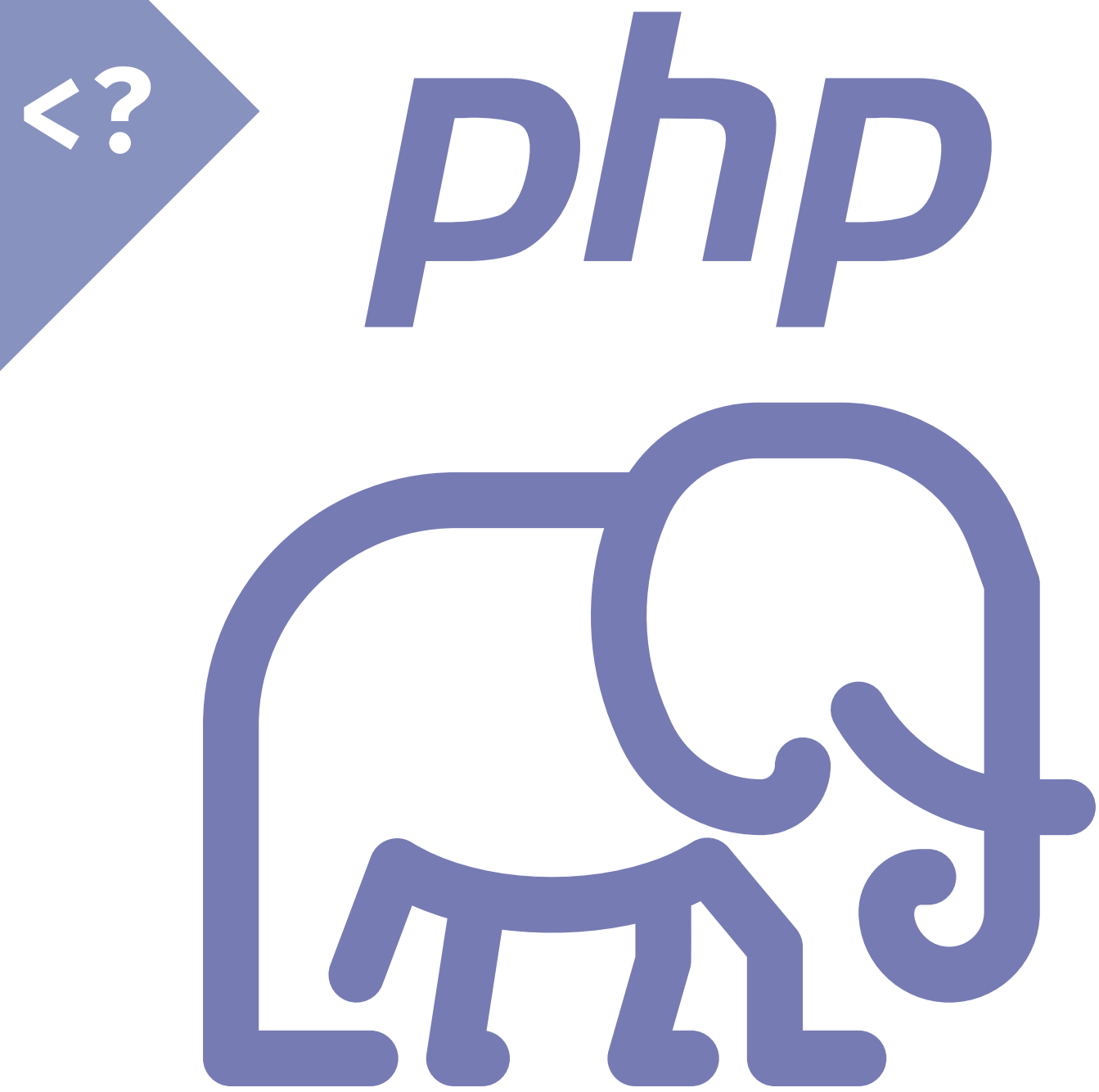
Inclui PSRs e boas práticas

Desenvolvimento Web com



Atualizado para PHP 8.1

{ PROFESSOR CHOREN }

Prof. Esp. Anderson Coimbra Choren

Sobre o Autor

Olá, me chamo Anderson Coimbra Choren e gostaria de compartilhar com você, caro leitor, um pouco da minha vida profissional, para que possamos nos conhecer melhor.

Sou um profissional com mais de 14 anos de atuação na área de Tecnologia da Informação. Sendo graduado em Análise e Desenvolvimento de Sistema e pós-graduado em Especialização em Formação de Professores na Docência do Ensino Profissional e Superior, ambos pelas Escolas e Faculdades QI do Brasil, atualmente atuando como professor do curso técnico em informática na mesma instituição, lecionando em disciplinas relacionadas a programação e análise de sistemas. Atuante também nas áreas de UX e UI, áreas essas onde possui formações livres.

FACEBOOK.COM/GROUPS/431311894533110
@PROFESSORCHOREN
@PROFESSORCHOREN
PROFESSORCHOREN@GMAIL.COM
ANDERSONCHOREN.COM.BR

Sumário

Capítulo 1 - Introdução a aplicações para ambiente web

Neste capítulo faremos uma breve introdução sobre o ambiente de desenvolvimento em nuvem, explicando conceitos como o modelo Cliente-Servidor, protocolos de comunicação para Internet e sistemas e softwares de gestão e protocolos, além de outros aspectos importantes para sua completa imersão no mundo do desenvolvimento web.

1.1 Protocolos de rede

Quando falamos em redes de computadores, nos deparamos com diversas características técnicas, como padrões (IEEEs, ISOs, entre outros), modelos de redes, camadas do modelo OSI(acrônimo do inglês *Open System Interconnection*), entre outros. Mas como o fundamento desta publicação não é tratar do treinamento de técnicos em redes de computador, vamos focar em um conceito chave, os protocolos da camada de aplicação do modelo OSI. Mais especificamente nos protocolos HTTP, HTTPS, FTP e SFTP.

1.1.2 Protocolo HTTP e HTTPS

🔓 Protocolo HTTP

O protocolo HTTP(*HyperText Transfer Protocol*) faz parte da lista de protocolos pertencentes à camada de aplicação do modelo OSI, junto com outros diversos. Sua finalidade é definir regras e modelos para a comunicação entre conteúdos de

hipertexto, sendo ele o protocolo base da *World Wide Web* (Nome técnico da Internet que todos conhecemos).

Dentre as inúmeras características técnicas deste protocolo, neste momento, nos é importante os métodos de solicitação. Sendo eles: *POST, GET, PUT e DELETE*. Existem outros métodos, mas esses são os principais e mais importantes. Caso deseje se aprofundar no assunto, você encontrará um artigo completo no site Wikipédia.com, onde poderá se aprofundar mais nas características técnicas deste protocolo, assim como links de acesso para as demais camadas do modelo OSI.

🔒 Protocolo HTTPS

O protocolo HTTPS(*HyperText Transfer Protocol Secure*), nada mais é que uma evolução do protocolo HTTP, sendo que essa versão do protocolo possui um sistema de conexão criptografada de ponta a ponta, permitindo assim uma maior segurança na transferência de recursos pela Internet.

Em resumo, se um dia você vier a criar uma aplicação web, irá querer implementar o uso da criptografia SSL ao seu domínio, pois isso trará muitos benefícios para sua aplicação, tanto em segurança, quanto em confiança por parte dos usuários e mecanismos de busca, como o Google. Eles não gostam de sites inseguros 😒.

1.1.2 Protocolo FTP e SFTP

🔓 Protocolo FTP

O protocolo FTP(*File Transfer Protocol*) é mais um dos protocolos da camada de aplicação do modelo OSI. Sendo baseado

no protocolo TCP, ao qual não iremos tratar neste, sua finalidade é prover uma série de regras e padrões para a transferência de arquivos por meio de redes de computadores.

Esse protocolo foi muito usado, e ainda é, para o envio de arquivos de arquivos de código-fonte de um sistema para o servidor no qual ele será mantido e irá funcionar para acesso via Internet.

Em resumo, sempre que você envia um arquivo através de uma rede de computadores, mesmo que não saiba, você está fazendo uso do protocolo FTP.

🔒 Protocolo SFTP

Assim como o HTTPS, o protocolo SFTP é uma evolução de seu protocolo original, adicionando uma criptografia a conexão de rede.

1.2 Entendendo o modelo Cliente-Servidor

O modelo cliente-servidor, é uma estrutura de aplicação distribuída que **distribui as tarefas e cargas de trabalho** entre os **fornecedores** 👔 de um recurso ou serviço, **designados como servidores**, e os **requerentes** dos serviços, **designados como clientes** 👨.

Na figura 1.1 é possível vermos uma ilustração do modelo Cliente-Servidor, onde encontramos três dispositivos cliente (Um celular, um notebook e um computador de mesa), todos se comunicando com o servidor através da Internet.

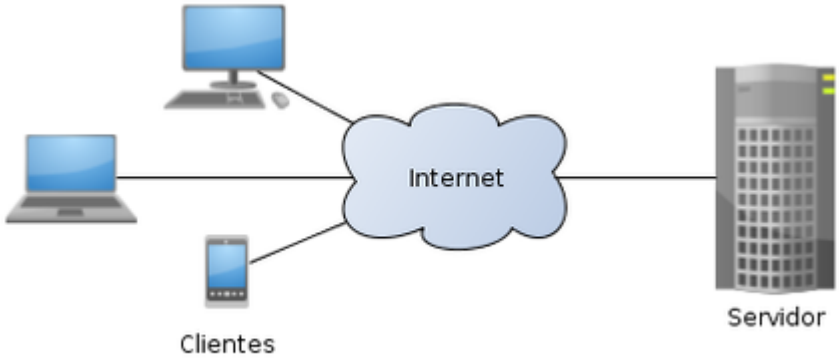


Figura 1.1 - Exemplo do modelo de arquitetura Cliente/Servidor

1.2.1 Lado servidor e suas responsabilidades

Neste modelo de arquitetura de distribuição, o lado servidor (Computador no qual a aplicação é mantida e na qual a mesma é executada), é responsável por gerenciar as informações da aplicação, como acesso a base de dados (banco de dados); Manipulação de arquivos; E demais tarefas que exijam permissão de acesso direto ao Hardware em que o sistema está sendo executado.

1.2.2 Lado cliente e suas responsabilidades

Neste modelo de arquitetura de distribuição, o lado cliente (navegador de internet), fica responsável pela comunicação com o servidor, ou seja, é ele que solicita e monitora o andamento de todas as solicitações feitas ao mesmo. Sendo responsável também por notificar toda e qualquer resposta vinda do mesmo, seja ela positiva, onde a solicitação foi recebida pelo servidor e o processamento foi bem sucedido, ou seja negativa, onde tanto a solicitação foi rejeitada pelo servidor ou houve algum erro na execução da aplicação, o que acabou por

acarretar uma Exceção (erro crítico durante a execução de um procedimento dentro da aplicação).

1.2.2 Requisições e Respostas

Quando falamos do modelo de arquitetura Cliente/Servidor, existem dois tópicos fundamentais que devemos compreender, sem tal compreensão não seria possível fazer uso desse modelo de arquitetura de informação, sendo elas, **requisições** e **respostas**.

1.2.2.1 Requisições

Uma requisição (em inglês *Request*) no dialeto do desenvolvimento de sistemas é a ação de requisitar (solicitar, pedir) para algo ou alguém realizar uma certa tarefa. No caso do modelo Cliente/Servidor, o lado cliente é responsável por executar essa solicitação, requisitando ao lado servidor que busque ou processe alguma informação mantida por ele, seja ela armazenada em um arquivo, na própria memória do mesmo ou em uma base de dados.

Na figura 1.2 podemos visualizar os dados técnicos de uma requisição para o meu site, resposta essa que apresenta que a conexão com o servidor onde ele se encontra foi bem sucedida.

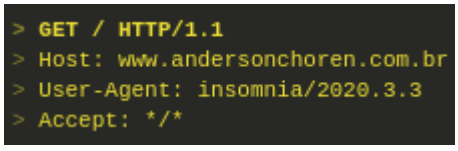


Figura 1.2 - Exemplo de requisição HTTP

1.2.2.1 Resposta

Uma resposta (em inglês Response), como o próprio nome já sugere, é a resposta à uma requisição feita ao servidor. Essa resposta é composta por diversas informações como o status da solicitação (status code), verbo *HTTP* (HyperText Transfer Protocol), Domínio requisitado (endereço da página) e etc.

Na figura 1.3 é possível observar que o servidor onde se encontra meu site tanto recebeu a requisição, quando a mesma foi um sucesso.

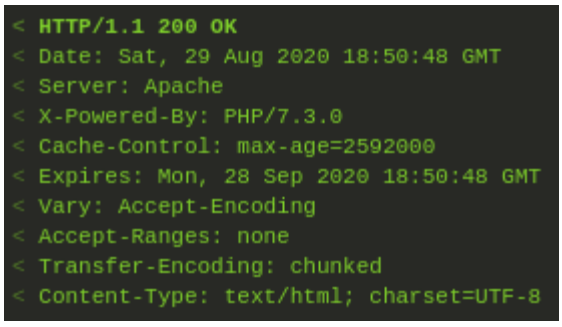


Figura 1.3 - Exemplo de resposta HTTP 1.1 200 OK

1.3 Servidor HTTP Apache

Como bem vimos, uma aplicação web é formada por dois lados, o lado servidor e o lado cliente. Mas como bem sabemos, tudo que opera em um computador nada mais é que um programa, e como o servidor não é diferente. Existem diversos softwares do tipo servidor, mas os dois mais famosos são o **Nginx** e o **Apache**, sendo que para esse material trataremos do Apache, já que dentre eles é o presente em servidores Web.

1.3.1 Servidor Apache

Figura 1.4 - Logo do Servidor HTTP Apache

O Servidor HTTP Apache (do inglês Apache HTTP Server) ou somente Apache, é o servidor web criado em 1995 por Rob McCool. É um servidor Web de código livre sob a licença GNU GPL (GNU General Public License), sendo um dos mais populares e utilizados servidores do tipo. Segundo uma pesquisa realizada em dezembro de 2007, o servidor Apache representa cerca de 47,20% dos servidores ativos no mundo. Esse número aumentou em maio de 2012, quando foi constatado que o Apache servia aproximadamente 54,68% de todos os sites.

1.4 Linguagem interpretada vs linguagem compilada

Antes de iniciarmos nossos estudos com a linguagem PHP é necessário entender um aspecto importante sobre essa linguagem. O PHP é uma linguagem de programação de alto nível e interpretada. Isso quer dizer que nosso código não é compilado em linguagem de máquina. Não está entendendo nada, não é verdade? 😅

Bom, deixe-me explicar melhor então. No universo do desenvolvimento de softwares existem dois grandes grupos de linguagens de programação, ao menos quando falamos em build de aplicação. Ah não, lá vem mais jargão técnico 🤨. Relaxa, vou explicar. Build (Do inglês: Construir ou construção) é a ação de criar um código-fonte pronto e preparado para rodar no computador, os tão famosos .exe do Windows. Então melhorou? 😅

Bom, continuando… Como havia comentado antes, existem dois tipos grupos de linguagens de programação, as que fazem o Build da aplicação para código nativo ou linguagem de máquina (Código binário) e as interpretadas, que fazem uso de um outro software para esse processo, e esse software será responsável por ler, interpretar e compilar a aplicação para nós.

Agora você deve estar se perguntando “OK, mas se no final do processo, seja uma linguagem interpretada ou não, teremos um código binário, porque existem dois tipos de linguagens?”. A resposta é bem simples. De forma resumida e bem superficial, antes que algum conservador queira minha cabeça, todos os sistemas operacionais operaram como uma taxa de bits, seja ela 32 bits ou 64 bits, isso sem falar em arquiteturas Arm e S.O (Sistema Operacional). Sendo assim, para rodarmos um programa nesses computadores (só lembrando, um servidor também é um computador, OK 😉) precisamos de programas que façam uso desta mesma arquitetura, ou poderemos ter problemas de alocação de memória e etc.

Aí entra o interpretador. Com ele um mesmo código pode ser escrito e executado em diversos computadores e arquiteturas

diferentes, com sistemas operacionais diferentes, seja Linux. MacOS ou Windows (🤢). OK, você não viu esse último emoji 😅.

Continuando… Dessa forma, não preciso me preocupar com a plataforma ou S.O em que o sistema irá rodar, já que será interpretado e devidamente compilado para funcionar corretamente no hardware em que está sendo utilizado.

Mas claro, nem tudo é “um mar de rosas”, existem algumas desvantagens neste processo, o principal deles é a velocidade de execução e resposta da aplicação. Uma aplicação compilada em bytecode sempre será mais rápida que uma interpretada. Porém é algo digno de análise, sua aplicação precisa ser extremamente rápida, se sim use uma linguagem compilada. Mas se a resposta é nem tanto, bom então relaxa e deixa o interpretador fazer o trabalho para você 😉.

Mas você deve estar se perguntando do porquê de tanto papo sobre linguagem interpretada vs compilada. A resposta é bem simples também, PHP é uma linguagem interpretada 😃.

1.4.1 Interpretador PHP

Bom, agora que já entendemos o que é uma linguagem compilada e uma linguagem interpretada, vamos tratar do PHP e seu interpretador de códigos.

O PHP, como bem já sabemos, é uma linguagem interpretada, assim como diversas outras como o Java (Sim, eu sei ele é compilado também, na verdade meio a meio, pois temos a JVM no processo), o Python, entre outras. Sendo assim, precisamos instalar um interpretador PHP em nosso servidor Web, seja ele o Apache ou não.

Mas por sorte isso é muito fácil, basta acessar o site oficial do PHP e instalar o interpretador do PHP. Mas é claro, não esqueça antes de instalar o seu servidor web através do site oficial do Apache ou do Nginx, como preferir. Ah, talvez você vá precisar de um banco de dados, como o MySQL ou o MariaDB.

Mas e se eu te disser que existe um jeito mais fácil 😌.

1.4.2 Os "Amps" da vida

Bom, como você já pode ter percebido, preparar um ambiente para desenvolvimento web não é lá muito simples, temos que baixar o servidor web, o interpretador do PHP e talvez um banco de dados, e mais, depois tem que configurar tudo isso 😒.

Bom, felizmente temos uma solução simples, rápida e prática; O uso de softwares do tipo "AMP".

"AMP" é o nome dado aos softwares que provêm uma infraestrutura de servidor de forma simples e prática, onde basta baixar um único programa para ter um servidor web (Apache ou Nginx), um banco de dados (MySQL ou MariaDB) e o interpretador PHP (Alguns tem mais de um, como interpretador para a linguagem Perl). E as alternativas são muitas, como é possível ver nessa pequena pesquisa no site alternativeto.net.

Bom, eu particularmente já usei grande parte deles, e não vi muita diferença de um para o outro. Mas para esse material usarei o XAMPP, que além de todos os software que comentei antes, ainda tem um interpretador para o Perl.

Só por curiosidade XAMPP significa **X**(Todas os S.O) **A**pache **M**ariaDB **P**HP **P**erl. Legal né 😁.

Capítulo 2 - Sintaxe básica do PHP

Figura 2.1 - Logo do PHP

Neste capítulo aprenderemos um pouco sobre os princípios básicos da sintaxe do PHP, como extensões de arquivos, comandos de saída de dados, uso de comentários, anotações de código (Annotations) e muito mais.

2.1 Extensões de um programa PHP

Assim como qualquer outro arquivo, seja ele um código-fonte ou não, um arquivo PHP tem uma extensão própria, sendo ela *.php*, mas existem variações para arquivos PHP. Sendo elas:

Tabela 2.1 - Extensões para arquivos PHP

Extensão	Significado
.class.php	Arquivo contendo classe(s) PHP.
.inc.php	Arquivo PHP a ser incluído, normalmente possui constantes e ou configurações.
.php7,.php5	Arquivo na versão “X” do PHP.
.phtml	Arquivo contendo código PHP junto ao HTML.

É importante ressaltar que algumas dessas extensões podem não ser aceitas em alguns tipos de servidores Web, devido a incompatibilidade ou mesmo a configurações feitas no mesmo. Sendo assim, é aconselhável que faço uso só da extensão **.php** em seus arquivos.

2.2 Delimitador de código

O código de um programa PHP deve sempre ser declarado dentro de delimitadores próprios da linguagem. Veja um exemplo a seguir:

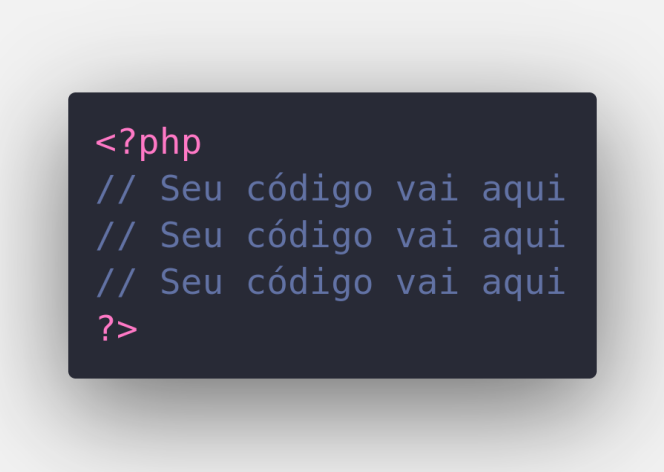


Figura 2.2 - Exemplo de código PHP

Segundo as orientações da PSR-1: Basic Coding Standard, quando **NÃO HOUVER** código HTML misturado ao código PHP, não há a necessidade de realizar o fechamento do delimitador PHP.

👨‍💻 Mão no código

```php
<?php

// Seu código vai aqui
// Seu código vai aqui
// Seu código vai aqui
```

Isso também é válido 😉

2.3 Comentários

Assim como todas as linguagens de programação, o PHP possui marcadores para comentários nos códigos, a fim de facilitar a leitura e compreensão de ações do mesmo.

2.3.1 Comentário de única linha

Para que possamos informar ao interpretador que não desejamos que ele “leia” uma linha, pois desejamos usá-la para guardar um comentário sobre o código, devemos usar os dois símbolos de /(Barra).

👨‍💻 Mão no código

```php
<?php
// Essa linha não é interpretada
```

2.3.1 Comentário de múltiplas linhas

```php
<?php
/*
    Tudo que eu escrever aqui
    não será interpretado pelo PHP
*/
```

2.4 Declaração de variável

A declaração de variáveis no PHP é muito simples, basta fazer uso do caractere `$` (cifrão), declarando logo após, sem espaços o nome da variável.

👨‍💻 Mão no código

```php
<?php
$nome = "Anderson";
```

2.5 Comandos de saída

2.5.1 Comando echo

O comando **echo** permite a impressão de uma ou mais variáveis no console, sendo o mais utilizado para tal finalidade.

👨‍💻 Mão no código

```php
<?php
$nome = "Anderson";
$sobrenome = " Choren";
echo $nome,$sobrenome;
```

🖨️ Resultado: Anderson Choren

2.5.1 Comando print

O comando **print** permite a impressão de uma variável no console.

👨‍💻 Mão no código

```php
<?php
$nome = "Anderson Choren";
print $nome;
```

🖨️ Resultado:

Anderson Choren

2.5.1 Comando var_dump

O comando **var_dump** permite a impressão de uma variável no console de forma explicativa, exibindo o tipo de dado e o conteúdo da variável. Muito útil para processos de debug 🐞.

👨‍💻 Mão no código

```php
<?php
var_dump(pi());
```

🖨️ Resultado:

```
float(3.1415926535898)
```

2.5.1 Comando print_r

O comando **print_r** permite a impressão de uma variável no console de forma explicativa, assim como o **var_dump**, mas em um formato mais legível.

👨‍💻 Mão no código

```php
<?php
print_r(pi());
```

🖨️ Resultado:

```
3.1415926535898
```

2.5.1 Comando var_export

O comando **var_export** permite a impressão de uma variável no console de forma explicativa, assim como o **var_dump** e o **print_r**, mas em um formato mais legível.

👨‍💻 Mão no código

```php
<?php
var_export(pi());
```

🖨️ Resultado:

```
3.14159265358979311599796346854418516159057617187 5
```

2.6 Escapando o HTML

Muitas são as vezes em que necessitamos imprimir conteúdos PHP juntamente com conteúdos HTML. Porém essa pode não ser algo tão simples, principalmente quando temos muito conteúdo no arquivo ou há a necessidade de criar condições para a impressão de conteúdos. Para isso podemos utilizar “o sistema de escapamento HTML do PHP”, onde podemos definir qual parte do código deverá ser processada no lado servidor, ou seja, pelo PHP. E qual deverá ser processada no lado cliente, ou seja, pelo navegador.

Vejamos agora alguns exemplos de códigos PHP com escapamento para conteúdos HTML. Não se preocupe se algo não ficar 100% claro, mas para frente aprenderemos sobre o uso destas funções, por enquanto vamos focar apenas no conceito e não na função do código 😉.

Escapando comandos de saída

```php
<?php
    $nome = "Anderson Choren";
?>
<body>
    <h1>Seja muito bem-vindo <?= $nome ?></h1>
</body>
```

🖨️ **Resultado:** `Seja muito bem-vindo Anderson Choren`

O uso do demarcador **<?= $variável ?>** é uma variação da expressão **<?php echo $variavel ?>**. Permitindo uma rápida impressão de valores sem a necessidade de inferir o comando **echo**.

Escapando estruturas de controle

```php
<?php
    $numero_de_voltas = 5;
?>
<body>
    <?php
        // Declaramos o loop for
        for($volta=0;$volta<$numero_de_voltas;$volta++):
    ?>
    <p>Já estamos na volta: <?= $volta ?></p>
    <?php
        // Comando de encerramento do loop for
        endfor;
    ?>
</body>
```

🖨️ Resultado:

```
Já estamos na volta: 0

Já estamos na volta: 1

Já estamos na volta: 2

Já estamos na volta: 3

Já estamos na volta: 4
```

2.7 Annotations

PHPDoc ou simplesmente Annotations, são marcações especiais feitas em blocos de comentários para definição de características do código-fonte do projeto, como tipo de dado que uma variável ou parâmetro de método, autor de um método ou classe, tipo de retorno de um método, pacote de uma classe e etc. Por existirem diversas anotações distintas no PHP, trataremos aqui das principais e mais importantes. Mas se

desejar mais informações sobre elas, você encontra uma documentação completa no site phpdocs.org.

2.7.1 @var

A anotação **@var** é utilizada para declarar uma anotação referente a uma variável, onde podemos declarar seu tipo de dado e sua descrição (Uma forma de explicar o motivo de seu nome ou o que deve armazenar).

2.7.1 @param

A anotação **@param** é utilizada para declarar uma anotação referente a um parâmetro de um método ou função, onde podemos declarar seu tipo de dado esperado, seu nome e sua descrição (Uma forma de explicar o motivo de seu nome ou o que deve receber).

2.7.1 @return

A anotação **@return** é utilizada para declarar uma anotação referente ao retorno de um método ou função, onde podemos declarar seu tipo de dado que será retornado, e sua descrição (Uma forma de explicar o que será retornado e/ou sob que circunstâncias isso ocorrerá).

2.7.1 @autor

A anotação **@autor** é utilizada para declarar uma anotação referente ao autor de um método, classe ou biblioteca, muito útil para momentos que disponibilizamos nossos códigos no GitHub, no Packagist, ou mesmo dentro de um projeto da empresa onde trabalhamos. É uma forma de “assinatura do artista”.

2.7.1 @copyright

A anotação **@copyright** é utilizada para declarar uma anotação referente aos direitos de um módulo ou projeto. Definindo assim a quem ele pertence. Uma forma definir a propriedade intelectual do código-fonte do projeto.

2.7.1 @licence

A anotação **@licence** é utilizada para declarar uma anotação referente a licença de uso de um módulo ou projeto. Definindo assim sob quais leis ou restrições aquele módulo ou projeto pode ser distribuído. Devemos apenas declarar o nome da licença, pois as cláusulas da mas fazem parte de um arquivo externo o **.licence**.

É importante ressaltar que o uso desse tipo de **anotação não é obrigatória**, porém é **uma das recomendações da PSR**, sendo assim seu uso é **altamente recomendado**, principalmente quando uma equipe é responsável pelo projeto ou o mesmo é de grande escala. Para fins didáticos na figura abaixo encontramos um exemplo básico do uso dessas anotações em um função fictícia.


```php
<?php
/**
 * Essa função tem como finalidade calcular o preço de venda de um produto
de acordo com uma taxa de lucro definida
 *
 * @copyright 2020 Corujador
 * @author Anderson Coimbra Choren <andersonchoren@gmail.com>
 * @param float $Markup Margem de lucro
 * @param float $preco_de_custo Preço de curso do produto
 * @return float Valor de venda do produto
 */
function calcularValorDeVenda(float $Markup,float $preco_de_custo):float{
    return ($preco_de_custo * $Markup)+$preco_de_custo;
}
```

2.8 PSR (PHP Standards Recommendations)

O PHP Standard Recommendation (PSR) é uma especificação do PHP publicada pelo PHP Framework Interop Group.

Ela serve à padronização de conceitos de programação em PHP. O objetivo é permitir a interoperabilidade de componentes e fornecer uma base técnica comum para a implementação de conceitos comprovados para programação ideal e práticas de teste. O PHP-FIG é formado por vários fundadores de famosos frameworks PHP, como o Laravel, CakePHP, entre outros. Você também pode ver a lista completa dos membros no site oficial do PHP-FIG no menu “Personnel”.

Cada PSR é sugerida por um dos membros e votada de acordo com um protocolo estabelecido para agir de forma consistente e de acordo com seus processos acordados.

No momento da construção desse material existem 18 PSRs aceitas pelo grupo, e mais 2 em desenvolvimento. Mas para esse material falaremos das duas principais, a **PSR-1**, que trata de

padrões básicos de codificação PHP e a **PSR-4**, que trata do padrão de carregamento de arquivos .php em nossos projetos.

2.8.1 PSR-1

A PSR-1 consiste em padrões básicos para a codificação em PHP, tratando dos mais básicos aspectos, como as variações dos delimitadores PHP aceitos pela comunidade, até conceitos mais avançados como padronização dos namespaces do projeto.

A seção a seguir foi uma livre interpretação do conteúdo apresentado no site oficial do PHP-FIG. Alguns trechos foram alterados e outros mantidos na integralidade apresentada na documentação.

2.8.1.1 Tags PHP

O código PHP DEVE usar as <?php ?>tags longas ou short-echo <?= ?>; **NÃO DEVE** usar as outras variações de tag.

👨‍💻 Mão no código

```php
<?php echo "Isso pode 😉"; ?>
<?= "Isso pode 😉"; ?>
<? echo "Isso NÃO pode!!!! 😐"; ?>
```

2.8.1.2 Codificação de caracteres

O código PHP DEVE usar apenas UTF-8 sem BOM.

Quanto a isso não precisamos nos preocupar, já que o próprio editor de texto por padrão apresenta essa configuração. Mas caso não o faça, você deverá fazê-lo manualmente.

2.8.1.3 Efeitos colaterais

Um arquivo **DEVE** declarar novos símbolos (classes, funções, constantes, etc.) e não causa nenhum outro efeito colateral, ou DEVE executar lógica com efeitos colaterais, mas **NÃO DEVE** fazer ambos.

👨‍💻 Mão no código

📝 formatacoes.php

```php
<?php

include 'Uteis.php';

function formatarDatas($tipo="BR",$data):void
{
    if(!empty($data)){
        echo Uteis::formatarData($tipo,$data);
    }
}

echo formatarData("EN","29/08/2012");
```

Isso não é nada bom 🤨

Acredito que esteja se perguntando “Afinal, o que está fazendo esse código? 🤔”. Mas tenha paciência pequeno *Padawan*, tudo será respondido a seu tempo. Agora precisamos focar nas normativas previstas nas PSRs e não necessariamente no código

PHP. Teremos espaço nesta publicação para todas essas funções, onde aprenderemos, de forma mais clara e bem mais ilustrativa, a utilização de todas essas funções.

Refatorando o código

Bom, mas você deve estar se perguntando, “OK, mas como devo proceder neste caso então”. E a resposta é bem simples, devemos refatorar o código para permitir uma melhor organização do código da nossa aplicação, permitindo que cada arquivo tenha apenas uma responsabilidade. As declarações de funções em um arquivo próprio, assim como as impressões no console. Assim cada arquivo terá uma responsabilidade única e poderá ser reutilizado.

Esse tipo de organização tem um nome técnico, na verdade é mais um conceito. Esse conceito se chama “Alta coesão/Baixo acoplamento”. Esse conceito visa uma melhor organização dos arquivos do projeto, a fim de tornar o código mais modular e reutilizável, assim poderemos fazer uso de uma mesma funcionalidade em partes distintas do projeto, sem a necessidade de reescrever essas funcionalidades.

Vejamos então um exemplo mais coeso e menos acoplado para esse código, assim seguiremos as recomendações da PSR-1 e tornamos nosso código mais reutilizável 😉.

👨‍💻 Mão no código

📝 formatacoes-v2.php

```php
<?php

include 'Uteis.php';

function formatarDatas($tipo="BR",$data):?string
{
    if(!empty($data)){
        return Uteis::formatarData($tipo,$data);
    }

    return null;
}
```

📝 index.php

```php
<?php

include 'formatacoes-v2.php';

echo formatarData("EN","29/08/2012");
```

Bem melhor agora 😁

2.8.1.4 Namespace

Os namespaces e classes DEVEM seguir a PSR de “carregamento automático”: **PSR-4**. Isso significa que cada classe está em um

arquivo por si só e em um namespace de pelo menos um nível: um nome de fornecedor de nível superior.

Os nomes das classes **DEVEM** ser declarados em **StudlyCaps**.

Código escrito para PHP 5.3 e posterior DEVE usar namespaces formais.

👨‍💻 Mão no código

```php
<?php

namespace Model\Util;

function MostrarDataEHora():string
{
  return date("d/m/Y H:i");
}
```

O código escrito para 5.2.x e anteriores DEVE usar a convenção de pseudo-namespacing de **Vendor_prefixos** em nomes de classe.

👨‍💻 Mão no código

```php
<?php

function Model_Util_MostrarDataEHora()
{
  return date("d/m/Y H:i");
}
```

Neste momento não se preocupe com a função **Date**, o foco principal agora é compreender as normativas regidas pela PSR-1. Teremos um capítulo dedicado somente a compreensão das funções do PHP.

2.8.1.5 Constantes

As constantes de classe **DEVEM** ser declaradas em letras maiúsculas, caso o nome seja composto, devemos separar as palavras com o sublinhado.

👨‍💻 Mão no código

```php
<?php
// Isso é válido 😉
const PI = 3.14159265358979;
// E assim também 😉
define("NUMERO_PI",3.14159265358979
);

// Mas isso aqui não 😕
const pi = 3.14159265358979;
// Nem isso aqui 😕
define("NUMEROPI",3.14159265358979
);
```

2.8.1.6 Propriedades

Segundo PHP-FIG “Este guia evita intencionalmente qualquer recomendação sobre o uso de **$StudlyCaps**, **$camelCase** ou **$under_score** nos nomes de propriedades”.

De toda forma, eu particularmente, prefiro definir as propriedades fazendo uso do padrão **camelCase**. Mas não é uma regra, só mantenha um padrão, não variando entre eles em uma

mesma aplicação. Isso poderá acarretar em um código não muito legível, muito desorganizado e despadronizado.

👨‍💻 Mão no código

```php
<?php
// Isso não é legal
$nome = "Anderson";
$segundoNome = "Coimbra";
$sobrenome = "Choren";

// Isso é legal
$nome = "Anderson";
$segundo_nome = "Coimbra";
$sobrenome = "Choren";
```

2.8.1.7 Métodos ou funções

Os nomes dos métodos **DEVEM** ser declarados em **camelCase()**.

👨‍💻 Mão no código

```php
<?php

// Isso NÃO é legal 😕
function somar_dois_numeros($n1,$n2):float
{
    return $n1+$n2;
}

// Isso é legal 😉
function somarDoisNumeros($n1,$n2):float
{
    return $n1+$n2;
}
```

2.8.2 PSR-4

Esta PSR descreve uma especificação para classes de carregamento automático de arquivos.

Este PSR também descreve onde colocar os arquivos que serão carregados automaticamente de acordo com a especificação.

2.8.2.1 Especificação

O termo “classe” refere-se a classes, interfaces, características e outras estruturas semelhantes.

Um nome de classe totalmente qualificado tem o seguinte formato:

\<NamespaceName>(\<SubNamespaceNames>)*\<ClassName>

- O nome de classe totalmente qualificado **DEVE** ter um nome de namespace de nível superior, também conhecido como **“namespace de fornecedor”**.
- O nome de classe totalmente qualificado **PODE** ter um ou mais nomes de sub-namespace.
- O nome de classe totalmente qualificado **DEVE** ter um nome de classe final.
- Os sublinhados **não têm nenhum significado especial** em nenhuma parte do nome totalmente qualificado da classe.
- Os caracteres alfabéticos no nome de classe totalmente qualificado **PODEM** ser qualquer combinação de minúsculas e maiúsculas.
- Todos os nomes de classes **DEVEM** ser referenciados diferenciando maiúsculas de minúsculas.

2.8.2.2 Exemplos

A tabela a seguir mostra o caminho do arquivo correspondente para um determinado nome de classe totalmente qualificado, prefixo de namespace e diretório base.

Tabela 2.2 - Exemplos de namespaces qualificados

NOME DE CLASSE TOTALMENTE QUALIFICADO	PREFIXO DE NAMESPACE	DIRETÓRIO BASE	CAMINHO DO ARQUIVO RESULTANTE
\Acme\Log\Writer\File_Writer	Acme\Log\Writer	./acme-log-writer/lib/	./acme-log-writer/lib/File_Writer.php
\Aura\Web\Response\Status	Aura\Web	/caminho/para/aura-web/src /	/path/to/aura-web/src/Response/Status.php
\ Symfony \ Core \ Request	Symfony\Core	./vendor/Symfony/Core/	./vendor/Symfony/Core/Request.php
\ Zend \ Acl	Zend	/usr/inclui/Zend /	/usr/includes/Zend/Acl.php