Introdução ao

PHP Orientado a Objetos

De forma prática

Ribamar FS

Sumário

Agradecimentos

Sou grato a toda a cadeia em torno do PHP.
- Ao Rasmus, por ter criado.
- À equipe que atualmente toca o PHP e tem corrigido erros e melhorado, adicionando novos e importantes recursos a cada versão.
- Ao Google, por me levar aos recursos existentes da enorme comunidade ao redor do PHP
- Ao Github por criar e manter este maravilhosa rede social, que abriga projetos e páginas também de forma gratuita e bastante simples e ágil.
- Às comunidades e fóruns criados ao redor dos issues dos repositórios do Github.
- Ao pessoal do Stackoverflow, que é quase onipresente quando faço uma pergunda no Google.
- A toda a rede de grupos espalhadas pela internet, especialmente aos colegas dos grupos do Facebook, tanto aos mais experientes que usam seu tempo e generosidade para ajudar, quanto aos iniciantes que eprguntam.
- A todos os demais, não deixando ninguém de fora, autores de livros, criadores de conteúdo em geral: tutoriais, srtigos, dicas, etc. A todos os demais não citados diretamente.

Público Alvo

Programadores iniciantes em PHP que já deram seus passos iniciais no PHP estruturado.
Programadores com experiência em outras linguagens.

E a qualquer pessoa interessada em aprender programar em PHP, desde que esteja com motivação suficiente para acompanhar o conteúdo, pois idealmente deveria conhecer antes o PHP estruturado.

Minhas motivações para escrever este livro

Só escrevo um livro quando acredito que ele poderá ser útil para quem o ler. Se acreditasse que um outro livro já faz o que este fará, então bastaria uma referência ao outro livro.

Uma observação importante é que durante vários anos estudando encontrei diversos tipos de professores, de péssimos e excelentes. Ensinar é algo que tenho feito por por muito tempo na minha vida. Portando me interesso muito pelo assunto, estudo e reflito sobre o mesmo. Se fala muito em didática como uma característica de bons professores. Eu acabei por elaborar uma definição para didática, como sendo a forte motivação que um professor tem para ensinar seus alunos. Por conta desta motivação ele procura formas de mais simples de ensinar para que seus alunos entendam melhor. Ele seleciona as formas mais simples, começa pelo simples para somente depois ir passando gradativamente para o mais complexo. Esta forte motivação que ele tem de ensinar e seu grande interesse por cada aluno faz com que ele ensine bem e a isso chamo de didática.

Eu tenho me empenhado para ser um destes professores e também o faço quando escrevo. Afinal de contas meu interesse em transmitir algo é o de me fazer entender e não o de mostrar o quanto sei.

Objetivo principal deste livro

O objetivo deste livro é o de preparar o programador para começar a usar um bom framework PHP que use MVC e padrões de projeto sem grandes dificuldades e ao contrário aprendendo de forma suave. Tanto que ao final existe um capítulo com uma introdução ao framework PHP mais popular atualmente, que é o Laravel.

A recomendação geral não é que se crie seu próprio framework PHP do zero, mas que conheçamos o PHP orientado a objetos, o MVC, boas práticas e alguns padrões de projeto para poder usar com mais facilidade um dos bons frameworks PHP existentes. Mas se tiver interesse em criar seu próprio framework do zero veja este artigo:

https://ribafs.github.io/backend/phpoo/mvc/fw-zero/

Então para ser realmente prático estarei criando um aplicativo em PHP usando orientação a objetos e MVC, que é baseado no meu repositório do /Github:

https://github.com/ribafs/simplest-mvc

Sobre o autor

Ribamar FS é um programador PHP com forte foco no back-end. Com experiência de mais de 20 anos e a maior parte deles estudando e trabalhando somente com o paradigma estruturado.

É fã do software open source e livre e do Linux, o qual usa para trabalhar e em casa. Gosta dos SGBDs PostgreSQL, MySQL e SQLite.

Também gosta de mexer com servidores e ultimamente tem usado servidores VPS para hospedar seu site.

Ficou fascinado pela distribuição linux Alpine. Pela sua leveza quando comparada com as demais.

É formado em engenharia civil mas apaixonou-se pela informática tendo abandonado a engenharia e desde então trabalhado somente com informática.

Sente grande satisfação em colaborar com a comunidade e tem publicado uma grande quantidade de repositórios no GitHub: https://github.com/ribafs.

Atualmente está migrando seu site do servidor VPS no linode para o Github:

https://ribafs.github.io após ter descoberto que pode fazer algo parecido com o que fazia no servidor usando o Joomla, agora fazendo com o MkDocs. Compartilhou um tutorial sobre o MkDocs, de como criou seu site aqui:

https://github.com/ribafs/mkdocs-ribafs

Inclusive com os fontes.

Participa de uma grande quantidade de grupos de programação e servidores no Facebook. Onde compartilha seus conhecimentos e tira suas dúvidas.

Uma das coisas de que gosta é de produzir conteúdo, por conta disso geralmente está fazendo isso em um site ou blog. Já criou e publicou alguns livros. A maior parte deles está disponível gratuitamente em seu novo site:

https://ribafs.github.io/sobre/livros/

Livro sobre o Laravel 9

O livro que criou antes deste é o livro sobre o laravel 9, está sendo vendido por R$ 30,00 com material restrito: aplicativos criados durante a elaboração do livro.
Interessados mandem e-mail para ribafs@gmail.com.

Um pouco da história do PHP

Versões e lançamentos do PHP

PHP 1 - 1995 - Rasmus Lerdorf libera o código fonte para o público
PHP 2 - 1997
PHP 3 – 1998 – PHP oeirentado a objetos começou nesta versão
PHP 4 - 2000
PHP 5 – 2004 – Nesta versão a OO se consolidou
PHP 6 - Não existiu
PHP 7 - 2015
PHP 7.1 - 2016
PHP 7.2 - 2017
PHP 7.3 – 2018
PHP 7.4 – 2019
PHP 8.0 – 2020
PHP 8.1 – 2021
PHP 8.2 – 2022

https://php.watch/versions

O que o PHP pode fazer?

Qualquer coisa. O PHP é focado principalmente nos scripts do lado do servidor, portanto, você pode fazer qualquer coisa que outro programa CGI pode fazer, como coletar dados de formulários, gerar páginas com conteúdo dinâmico ou enviar e receber cookies. Mas o PHP pode fazer muito mais.

Existem três áreas principais onde os scripts PHP são usados:

Scripts no lado do servidor (server-side). Este é o mais tradicional e principal campo de atuação do PHP. Você precisa de três coisas para isto funcionar: o interpretador do PHP (CGI ou módulo do servidor), um servidor web e um navegador web. Você precisa rodar o servidor web conectado a uma instalação do PHP. Você pode acessar os resultados de seu programa PHP com um navegador web, visualizando a página PHP através do servidor web. Tudo isso pode rodar na sua máquina pessoal se você estiver apenas experimentando programar com o PHP. Veja a seção das instruções de instalação para mais informações.

Scripts de linha de comando. Você pode fazer um script PHP para executá-lo sem um servidor ou navegador. A única coisa necessária é o interpretador PHP. Esse tipo de uso é ideal para script executados usando o cron (Unix, Linux) ou o Agendador de Tarefas (no Windows). Esses scripts podem ser usados também para rotinas de processamento de texto simples. Veja a seção Utilizando o PHP em linha de comando para mais informações.

Escrever aplicações desktop. O PHP provavelmente não é a melhor linguagem para criação de aplicações desktop com interfaces gráficas, mas se você conhece bem o PHP, e gostaria de usar alguns dos seus recursos avançados nas suas aplicações do lado do cliente, você pode usar o PHP-GTK para escrever programas assim. Você também tem a possibilidade de escrever aplicações multi-plataformas desse jeito. O PHP-GTK é uma extensão do PHP, não disponibilizada na distribuição oficial. Caso esteja interessado no PHP-GTK, visite » o site do projeto. Temos também o uso do PHP com a biblioteca Electron para uso em desktop https://github.com/cztomczak/phpdesktop

O PHP pode ser utilizado na maioria dos sistemas operacionais, incluindo Linux, várias variantes do Unix (como HP-UX, Solaris e OpenBSD), Microsoft Windows, macOS, RISC OS e provavelmente outros. O PHP também tem suporte à maioria dos servidores web atualmente. Isso inclui o Apache, o IIS e muitos outros. E isso inclui qualquer servidor web que possa utilizar o binário FastCGI do PHP, como o lighttpd e o nginx. O PHP trabalha tanto como módulo quanto como um processador CGI.

Com o PHP, portanto, você tem liberdade de escolha de sistema operacional e de servidor web. Além disso, você pode escolher entre utilizar programação estruturada ou programação orientada a objeto (OOP), ou ainda uma mistura das duas.

Com PHP você não está limitado a gerar somente HTML. As habilidades do PHP incluem geração de imagens, arquivos PDF e até animações Flash (utilizando libswf e Ming) criados dinamicamente, on the fly. Você pode facilmente criar qualquer texto, como XHTML e outros arquivos XML. O PHP pode gerar esses arquivos e salvá-los no sistema de arquivos, em vez de mostrá-los em tela, formando um cache no lado do servidor para seu conteúdo dinâmico.

Uma das características mais fortes e mais significativas do PHP é seu suporte a uma ampla variedade de banco de dados. Escrever uma página web consultando um banco de dados é incrivelmente simples usando uma das extensões específicas de banco de dados (por exemplo, mysql), ou usando uma camada de abstração como o PDO ou conectar a qualquer banco de dados que suporte o padrão "Open Database Connection" usando a extensão ODBC. Outros bancos de dados podem utilizar cURL ou sockets, como o CouchDB.

O PHP também tem suporte para comunicação com outros serviços utilizando protocolos como LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (no Windows) e incontáveis outros. Você também pode abrir sockets de rede e interagir diretamente usando qualquer outro protocolo. O PHP também suporta o intercâmbio de dados complexos WDDX, utilizado em virtualmente todas as linguagens de programação para web. Falando de comunicação, o PHP implementa a instanciação de objetos Java e os utiliza transparentemente como objetos PHP.

O PHP tem recursos úteis para processamento de texto, incluindo expressões regulares compatíveis com Perl (PCRE), e muitas outras extensões e ferramentas para analisar e acessar documentos XML. O PHP padroniza todas as extensões XML a partir da base sólida da libxml2, além de estender o conjunto de recursos adicionando suporte a SimpleXML, XMLReader e XMLWriter.

E existem muitas outras extensões interessantes, que são categorizadas tanto alfabeticamente quanto por categoria. E existem também as extensões PECL adicionais que podem, ou não, estar documentadas dentro do próprio manual do PHP, como a » XDebug.

Como você pode ver, esta página não é suficiente para descrever todos os recursos e benefícios que o PHP pode oferecer. Leia as seções sobre a Instalação do PHP, e veja a parte da referência das funções para detalhes sobre as extensões mencionadas aqui.

https://www.php.net/manual/pt_BR/intro-whatcando.php

Quando usar o PHP puro e quando usar um framework

Consideramos que usar o core do PHP é como resolver um problema matemático usando apenas papel e caneta. Trabalhar com um bom framework seria como resolver problemas matemáticos usando uma calculadora.

Mas para fazer a soma de 2 + 2 eu faço bem mais rápido que usando uma calculadora. Além disso, se eu não souber somar, não poderei usar a calculadora. Antes aprender PHP depois usar o framework.

Problema matemático principal para solução com PHP

Apenas alguns alunos podem obter resultados usando papel e caneta da mesma forma que com PHP puro. Apenas alguns desenvolvedores podem escrever o código de maneira fácil e em formato confiável.

Framework - Resolvendo problema matemático

Todos podem obter o resultado usando a calculadora da mesma forma que com PHP. Mesmo iniciantes podem escrever o código de maneira fácil e em formato confiável num framework.

O principal problema com o PHP puro é quando os desenvolvedores escrevem sua própria lógica, é difícil entender o resultado, então a maioria dos desenvolvedores está escolhendo frameworks inovadores.

Framework

A maioria dos frameworks não oferece confiabilidade, consistência e economia de tempo. Alguns dos frameworks inovadores estão tendo um rico conjunto de funcionalidades, então o desenvolvedor não precisa escrever código inteiro, os desenvolvedores precisam acessar o código usando o framework e desenvolver um aplicativo web PHP. Os frameworks não fornecem as soluções para programadores ruins, mas fornecem confiabilidade ao escrever código.

Melhorar Projetos

Todo mundo quer migrar para tecnologias sofisticadas. Se algum site ou aplicativo da Web foi desenvolvido com o core do PHP, é difícil aprimorar os componentes do site, mas se o site ou aplicativos da Web foram desenvolvidos com um Framework PHP popular, é mais fácil aprimorar os recursos.

O Core do PHP é ruim?

Não é nada ruim. O core do PHP ajuda você a escrever o código e entender o código. quando o desenvolvedor estiver no estágio inicial. Recomendamos fortemente que você aprenda o core do PHP, porque não queremos vê-lo como um desenvolvedor ruim. De acordo com a teoria do mundo, simples sempre dá melhor resultado com base forte. De acordo com a teoria do mundo, se você conhece o PHP puro, atingirá seu objetivo usando o framework PHP.

https://www.tutorialspoint.com/php/php_core_vs_frameworks.htm

Convertendo código estruturado para código orientado a objetos

Tive tal procedimento em um site que eu mesmo desenvolvi quando `PHP` não tinha `POO` ainda. O grande problema é entender a regra de negócio atual e traduzir para a Programação Orientada a Objeto. São coisas diferentes, e a principal meta sua é entender o Procedural e Analisar antes de implementar em POO.

Dica:
1) O que o seu sistema faz ?
Exemplo: Faz um cadastro de cliente, como você pode reaproveitar alguma coisa, talvez a View e alguns código, mas, em termos reais, a parte codigo terá que ser reescrita, é um trabalho grande.

2) Posso reaproveitar código ?
Exemplo: As `SQL` você pode aproveitar, acredito que são iguais ou praticamente. Agora com certeza vai ter que usar `PDO` ou `MySqli`, então, ai começa as alterações.

3) Vale a pena mudar ou fazer outro?
Muitas vezes é melhor projetar em POO um novo sistema que possa ser feito em paralelo ao antigo, evitando assim a parada do sistema, e dando enfase ao novo sistema.

4) Trabalhar com Padrões ?
Eu indico trabalhar com padrões e nomenclaturas atuais, interfaces, abstract, class. Um fator importante é saber POO bem, e talvez até implementar com Frameworks MVC, exemplo, Laravel ou Zend (indico Laravel é mais rápido o aprendizado, o Zend a curva de aprendizado é maior)

5) Código Procedural para Código POO, cuidado?
Cuidado, como já foi relatado POO é conceito, siga os conceitos POO e faça um novo código com tais conceito. Experiência: uma vez trabalhei numa equipe que tinha a programação em Cobol e eles começaram a fazer um sistema em VB6, o que aconteceu, tudo que eles faziam no Cobol eles replicavam no VB6, isso foi o pior absurdo que eu já presenciei.
https://pt.stackoverflow.com/questions/21181/dicas-para-se-transformar-c%C3%B3digo-procedural-em-orientado-a-objeto

- *Boa resposta, muitos pontos relevantes. Destaco: "é melhor projetar em POO um novo sistema que possa ser feito em paralelo ao antigo".*

Participei de um projeto recentemente onde tive que portar um sistema desktop programado em VB6 para ASP.NET C# OO (3 camadas).

Minha análise foi a seguinte:
- *Entender como o sistema(e cada página dele) funciona e como usá-lo (Identificar o que é o que e pra que serve).*
- *Entender a regra de negócio (isso é muito importante pois você pode encontrar restrições que acabam quebrando outras páginas que dependem dessa regra).*
- *Identificar os objetos e métodos do sistema e listá-los.*
- *Procurar por funções e métodos que podem se tornar genéricos.*

Como OO é um conceito creio que minha experiência pode te ajudar

1 – Introdução ao PHP

O que é o PHP?

O PHP (um acrônimo recursivo para PHP: Hypertext Preprocessor) é uma linguagem de script open source de uso geral, muito utilizada, e especialmente adequada para o desenvolvimento web e que pode ser embutida dentro do HTML.

Em vez de muitos comandos para mostrar HTML (como acontece com C ou Perl), as páginas PHP contém HTML em código mesclado que faz "alguma coisa". O código PHP é delimitado pelas instruções de processamento (tags) de início e fim <?php e ?> que permitem que você entre e saia do "modo PHP".

O que distingue o PHP de algo como o JavaScript no lado do cliente é que o código é executado no servidor, gerando o HTML que é então enviado para o cliente/navegador. O navegador recebe os resultados da execução desse script, mas não sabe qual era o código fonte. Você pode inclusive configurar seu servidor web para processar todos os seus arquivos HTML com o PHP, e então não há como os usuários dizerem o que você tem na sua manga.

A melhor coisa em usar o PHP é que ele é extremamente simples para um iniciante, mas oferece muitos recursos avançados para um programador profissional. Não tenha medo de ler a longa lista de recursos do PHP. Pode entrar com tudo, o mais rápido que puder, e começar a escrever scripts simples em poucas horas.

Apesar do desenvolvimento do PHP ser focado nos scripts do lado do servidor, você pode fazer muito mais com ele. Veja sobre isso na seção O que o PHP pode fazer?, ou vá diretamente para o tutorial introdutório se você estiver interessado apenas em programação web.

https://www.php.net/manual/pt_BR/intro-whatis.php

Ponto forte do livro

Ao meu ver o ponto forte deste livro é a construção do aplicativo em PHPOO com MVC, passo a passo e com explicações quando importante. Levando em conta que a prática leva a consolidação do conhecimento e da segurança.

Introdução ao PHP Orientado a Objetos

O PHP inclui uma modelagem a objetos completa. Entre os recursos disponíveis há: visibilidade, classes e métodos abstratos e final, additional métodos mágicos, interfaces, clonagem.

O PHP trata objetos da mesma maneira que referências ou manipuladores, significando que cada variável contém uma referência a um objeto ao invés de uma cópia de todo o objeto. Veja Objetos e Referências

https://www.php.net/manual/pt_BR/oop5.intro.php

Programação orietntada a objetos não é linguagem, mas um paradigma, uma forma de trabalhar com programação que existe em várias linguagens de programação. Algumas linguagens somente trabalham orientado a objetos, comoé o caso do Java. O PHP nos oferece a liberdade de trabalhar orientado a objetos ou estruturado. É importante entender os dois paradigmas para usar adequadamente de acordo com a necessidade.

PHP não é uma linguagem que foi criada para ser orientada a objetos (só começou a suportar orientação a objetos na versão 3, sendo aprimorada na versão 4, na versão 5.3e o suporte a orientação a objetos está excelente), os programadores PHP utilizavam ou a programação estruturada ou orientação a funções (nomenclatura usada por estudantes para definir um método de desenvolvimento). Este método basicamente organiza as funções mais utilizadas em arquivos específicos, como por exemplo, um arquivo chamado funções de banco e neste arquivo são colocadas as funções de insert, update e delete, depois bastava incluir o arquivo no local onde deseja utilizar as funções. Para isso utiliza-se os métodos include, include_once, require ou require_once do PHP e chamar as funções.

https://www.devmedia.com.br/introducao-a-orientacao-a-objetos-em-php/26762

A programação estruturada no PHP divide o código em blocos e funções. Já a orientada a objetos usa somente classes e objetos. Veremos detalhes sobre estes dois conceitos e sobre vários outros mais a frente.

2 – Ambiente de Desenvolvimento

Para trabalhar com PHP atualmente precisamos basicamente:

- O PHP instalado, de preferência a versão 8
- Um SGBD, sugiro o MySQL por ser mais popular, somente se for trabalhar com banco de dados
- Um servidor web, apache, nginx, etc. Opcional pois o PHP tem um servidor web intereno.

Não irei detalhar aqui a instalação do ambiente. Caso não tenha instale:

- No Windows o Laragon ou o Xampp
- No Linux instale os pacotes da própria distribuição

3 – Ferramentas

O PHP não é exigente em termos de ferramentas, podemos trabalhar até com o Xed ou com o Bloco de notas.

Editor de Código

A minha sugestão vai para o VSCode, que ajuda muito na programação com PHP e com diversas outras linguagens.

https://code.visualstudio.com

Existe uma grande quantidade de conteúdo sobre ele, pois é muito popular atualmente, mas passarei algumas dicas que me são mais úteis:

Reindentar Código

F1
Reindent lines

Usando snippets

- Crie um arquivo HTML
- Digite o ponto de exclamação (!) e em seguida tecle Enter. Aparecerá um snippet para HTML 5.

Ainda num arquivo HTML digite

Supondo que queira criar a seguinte estrutura:

```
<div class="container">
   <div class="row">
      <div class="col-md-6"></div>
   </div>
</div>
```

Basta que digite:

div.container>div.row>div.col-md-6 e tecle Enter que a mágica está feita.

Agora pesquisa sobre o uso da extensão Emmet no VSCode, que tem muito mais.

Ícones
Um bom conjunto de ícones torna o uso do VSCode ainda melhor. Sugestão

Vá no ícone Extensions e digite Material Icon Theme

Após instalar clicar acima em
Set File Icon Theme
E clicar em Material Icon Theme

Material Theme
Após instalar selecione a alternativa acima na caixa de comando
Também pode clicar em Set Color Theme e escolher

Agora abra uma pasta e veja como ficam os ícones para cada pasta e tipo de arquivo.

Existem muitas extensões úteis mas as instale com moderação, somente quando precisar, caso contrário desinstale, pois o VSCode sozinho já um pouco pesado e as extensões ainda o tornam mais pesado.

Gerenciador de Bancos de dados

Meu preferido é o Adminer

https://www.adminer.org

Em um único arquivo php temos um gerenciador simples e eficientes, além de que suporta os principais SGBDs livres: PostgreSQL, MySQL/MariaDb, SQLite, etc.

Claro que pode usar qualquer outros, phpMyAdmin ou mesmo a console do MySQL.

Criador de Sites para Documentação

Se precisar documentar de forma decente um projeto seu, minha recomendação vai para o MkDocs - https://www.mkdocs.org

Se precisar de ajuda com ele veja este tutorial com exemplo:
https://github.com/ribafs/mkdocs-ribafs

Composer

Atualmente o composer é muito importante, pois é o gerenciador de dependências do PHP e com ele se instala pacotes. Os bons frameworks para serem instalados precisam do composer e não somente eles.

A instalação no Windows é muito simples. Apenas baixe de

https://getcomposer.org/

E instale

Instalar composer2 no linux mint 20.3

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') ===
'906a84df04cea2aa72f40b5f787e49f22d4c2f19492ac310e8cba5b96ac8b64115ac402c8cd292b8a034
82574915d1a8') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-
setup.php'); } echo PHP_EOL;"
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

sudo mv composer.phar /usr/bin/composer

PHP Online

Idealmente devemos instalar o PHP em nosso desktop para estudar com mais flexibilidade e recursos. Mas caso não seja possível, podemos usar uma das ferramentas online

https://www.w3schools.com/php/php_compiler.asp

https://app.codingrooms.com/w/YdixDb887sNJ

https://glitch.com/
node
react
eleventy
sqlite

https://www.gitpod.io/

https://ide.goorm.io

https://replit.com

Diversas linguagens: PHP e outras
https://ideone.com/

Extensões para Navegadores

Existem boas extensões para navegadores que ajudam os programadores, em especial a ferramenta Inspecionar nativa dos navegadores, ajudam muito principalmente quando debugando Javascript, mas não somente, pois também vem com recurso para testar se o site está responsivo e mais.

4 – PHP Estruturado – Resumo

Estruturas de controle

if - é uma construção que testa se uma expressão é verdadeira
Caso seja verdadeira executa um código, no caso aqui mostra a mensagem "a é maior que b"

```
<?php

$a = 5;
$b = 3;

if ($a > $b){
   echo "a é maior que b";
}
```

// Caso o arquivo contenha somente código PHP e nada de HTML, então não use a tag de fechamento, ? >, como agora

Experimente em seu computador para ver o comportamento. Faça o mesmo com os demais exemplos. Lembre que a segurança se consolida com a prática.

else - Como o if testa se uma expressão é verdadeira, então se ela não for verdadeira executa o que vem abaixo do else.

```
<?php

$a = 3;
$b = 5;

if ($a > $b){
   echo "a é maior que b";
} else {
   echo "a NÃO é maior que b";
}
```

elseif - esta estrutura testa uma segunda, uma terceira, etc expressão.

```
<?php

$a = 3;
$b = 3;

if ($a > $b){
   echo "a é maior que b";
} elseif ($a==$b){
   echo "a é igual a b";
} else {
   echo "a NÃO é maior que b nem igual a b";
}
```

Laços de repetição

for – é uma estrutura que repete a execução de um código enquanto uma condição seja satisfeita.

```php
<?php

/* exemplo 1 - Controla o fluso no for*/
echo "<br><br>1- ";
for ($i = 1; $i <= 10; $i++) {
   echo $i;
}

echo "<br><br>2- ";
/* exemplo 2 - Controle o fluxo no if interno*/
for ($i = 1; ; $i++) {
   if ($i > 10) {
      break; // O break encerra o processamento do for
   }
   echo $i;
}

echo "<br><br>3- ";
/* exemplo 3 - Controle o fluxo também no if interno*/
$i = 1;
for (; ; ) {
   if ($i > 10) {
      break;
   }
   echo $i;
   $i++;
}

echo "<br><br>4- ";
/* exemplo 4 */
for ($i = 1; $i <= 10;$i++);
         echo $i;
```

continue – podemos usar este comando nos laços. Ele irá abandonar a iteração atual e continuar na próxima. Lembrando cada execução do laço chama-se iteração, sem o n.

```php
<?php
$arr = array(1,2,3,4,5,6,7,8);
while (list ($key, $value) = each ($arr)) {
   if (!($key % 2)) { // pula itens pares, ou seja, processa somente ímpares
      continue;
   }
   echo ($value);
}
```

while – É um laço condicional, que executa os comandos em seu corpo de acordo com a condição.

```
<?php
/* exemplo 1 */

$i = 1;
while ($i <= 10) {
  echo $i++;  /* o valor impresso será
              $i depois do acréscimo
              (post-increment) */
}

/* exemplo 2

$i = 1;
while ($i <= 10):
  echo $i;
  $i++;
endwhile;
*/
```

do while - Executa pelo menos uma vez incondicionalmente

```
<?php

$i = 0;
do {
  echo $i;
} while ($i > 0);
```

foreach – laço para varrer arrays e objetos

```
<?php

$a = array(1, 2, 3, 17);
$i = 0; /* para exemplo somente */
foreach ($a as $v) {
  echo "\$a[$i] => $v.<br>";
  $i++;
}
/* exemplo foreach 3: chaves e valores */
$a = array (
  "um" => 1,
  "dois" => 2,
  "três" => 3,
  "dezessete" => 17
);
foreach ($a as $k => $v) {
  echo "\$a[$k] => $v.<br>";
}
```

switch – Ums estrutura semelhante a uma cadeia de elseifs.

```
<?php
```

```
$i = 1;

// Estrutura com if
if ($i == 0) {
   echo "\$i igual a 0";
} elseif ($i == 1) {
   echo "\$i igual a 1";
} elseif ($i == 2) {
   echo "\$i igual a 2";
}
echo "<br>";
// Estruturas com switch
switch ($i) {
   case 0:
      echo "\$i igual a 0";
      break;
   case 1:
      echo "\$i igual a 1";
      break;
   case 2:
      echo "\$i igual a 2";
      break;
}
echo "<br>";
$i = 2;
// Executará todos, falta o break
switch ($i) {
   case 0:
      echo "\$i igual a 0";
   case 1:
      echo "\$i igual a 1";
   case 2:
      echo "\$i igual a 2";
}
echo "<br>";
// Simulando intervalos
switch ($i) {
   case 0:
   case 1:
   case 2:
      echo "\$i é menor que 3 mas não negativo";
      break;
   case 3:
      echo "\$i é 3";
}
echo "<br>";
// Valor default
switch ($i) {
   case 0:
      echo "\$i igual a 0";
      break;
   case 1:
      echo "\$i igual a 1";
      break;
```

```
  case 2:
     echo "\$i igual a 2";
     break;
  default:
     echo "\$i não é igual a 0, 1 ou 2";
}
/* Sintaxe alternativa (em obsolescência)
switch ($i):
  case 0:
     echo "\$i igual a 0";
     break;
  case 1:
     echo "\$i igual a 1";
     break;
  case 2:
     echo "\$i igual a 2";
     break;
  default:
     echo "\$i não é igual a 0, 1 ou 2";
endswitch;
*/
```

Array - é um mapa ou conjunto ordenado de pares de chaves e valores.

Uma chave é como um índice
Um valor é o valor da chave

Exemplo de array vazio

```
$ar = array();
```

No PHP atual podemos fazer assim:

```
$ar = [];
```

Funções nativas do PHP para arrays:

is_array() - checar se uma variável contém um array e retorna true ou false.

https://phpro.org/tutorials/Introduction-To-Arrays.html
Exemplos

```
<?php
$a = array("a" => "maçã", "b" => "banana");

$b = ["a" =>"pêra", "b" => "framboesa", "c" => "morango"];
```

Função para encotnrar o dia da semana de uma data qualquer

```
<?php

// https://pt.wikipedia.org/wiki/PHP
date_default_timezone_set("America/Fortaleza");
```

```
function diasemana($data) {
    $d = explode('/', $data);
    $anohoje = $d[2];
    $meshoje = $d[1];
    $diahoje = $d[0];
    $diasemana = date("w", mktime(0,0,0,$meshoje,$diahoje,$anohoje) );
    switch($diasemana)
    {
        case"0": $diasemana = "Domingo";       break;
        case"1": $diasemana = "Segunda Feira"; break;
        case"2": $diasemana = "Terça Feira";   break;
        case"3": $diasemana = "Quarta Feira";  break;
        case"4": $diasemana = "Quinta Feira";  break;
        case"5": $diasemana = "Sexta Feira";   break;
        case"6": $diasemana = "Sabado";        break;
    }
    return "$diasemana";
}

echo '<h1>' , diasemana('03/08/1956') ,  ' - Dia da Semana ' , '</h1>' ;
```

Temos **arrays unidimensionais** e multidimensionais (aqueles com mais de uma dimensão).

Array unidimensional

```
<?php

$client = [1, 'José', 11];

print '<table border="1">
<tr><td><b>Código</td><td><b>Nome</td><td><b>Idade</td></tr>
<tr>';
foreach ($client as $element){
   print '<td>'.$element.'</td>';
}
print '</tr></table>';
```

Array multidimensional

```
<?php

// Versão com array()

$frutas = array(
   // Brasil will act as key
   "Brasil" => array(
      // Subject and marks are
      // the key value pair
      'Banana' => 170,
      'Manga' => 145,
      'Laranja' => array(
         'prata' => 34,
         'rosa' => array(
            'grande' => 50,
            'pequena' => array(
               'amarela' => 10,
               'verde' => 20,
            ),
         ),
      )
   )
);

print $frutas['Brasil']['Laranja']['rosa']['pequena']['amarela'];
```

Funções anônimas no PHP

```
<?php
$a = function()
{
   print 'Olá';
};

// Observe o ponto e vírgula ao final, indicando uma variável.

// Chamando a função:

$a();



$falar = function ($msg)
{
   return "A mensagem é $msg";
}

print $falar('Estou aqui');
```

Operador Ternário

condição ? Expressão 1 se true : expressão 2 se false,

```
$action = (empty($_POST['action'])) ? 'default' : $_POST['action'];
$idade = 25;
echo ($idade<18) ? "Não pode dirigir" : "Já pode dirigir";
```

5 – PHP Orientado a Objetos Conceitos

Os conceitos na orientação a objetos são muito fortes e é uma grande diferença entre os paradigmas estruturado e orientado a objetos. Visto que praticamente não temos conceitos no estruturado mas no OO são muito fortes e rpecisamos entender os conceitos para melhor aproveitar os recursos da orientação a objetos.

Class - modelo de onde se derivam os objetos. Como uma planta de casa através da qual se constroi casas.

Objetos - derivados das classes. Não se usa classes na programação, mas somente objetos.

Propriedades - são correspondentes às variáveis no paradigma estruturada

Métodos - são similares as funções do estruturada

Visibilidade de propriedades e métodos:
- private - visível somente dentro da classe que a criou e define
- public - visível em qualquer objeto que o use (esta é a visibilidade default. Caso não usemos uma explicitamente será usada)
- protected - visível somente na classe que o criou ou nas classes que a estendem

Importante em termos de segurança das informações:
- Use somente private
- Caso não consiga atentder, então use protected
- Somente se não atender use public

Toda definição de classe
- começa com a palavra-chave class,
- seguido por um nome da classe, que pode ser qualquer nome que não seja uma palavra reservada no PHP,
- seguido por um par de chaves, que contém a definição dos membros e métodos da classe.

$this - Uma pseudo variável, $this está disponível quando um método é chamado dentro de um contexto de objeto. $this é uma referência para o objeto chamador do método (normalmente o objeto ao qual o método pertence, mas pode ser outro objeto, se o método é chamado estaticamente no contexto de um objeto secundário).

Exemplo simples de classe:

```
class Pessoa
{
   // Propriedade
   private $nome = 'João Brito';

   // Método
   public function getNome(){
      return $this->nome;
   }
}
```

A reutilização de código é um dos aspectos mais importantes da programação orientada a objetos.

Outro exemplo

```
class FiguraGeometrica
{
   // Propriedades
   public $altura;
   public $largura;

   // Método
   public area($altura, $largura){
      return $largura * $altura;
   }
}
```

$this

É usado para acessar as propriedades da classe dentro dos métodos (propriedades que estão fora dos métodos), como também os métodos da classe,seguidos do operador ->
O $this é uma pseudo propriedade

Exemplos:

$this->largura
$this->area()

Observe que diferente da propriedade, o método é chamado com seu nome().

Instância

Instância é o resultado de se criar um objeto partindo de uma classe

Quando criamos um novo objeto a partir de uma classe dizemos que criamos uma instância da classe.

Um objeto é uma variável criado quando instanciamos uma classe usando o operador new. Exemplo:

```
$objeto = new NomeClasse();

$homem = new Humano();
```

Através do objeto podemos acessar as propriedades e métodos da classe, mas somente aqueles que permitirem acesso.

```
class Humano
{
   public $nome = 'Ribamar';
   public $sobreNome = 'Sousa';

   public function nomeCompleto(){
      $nome = 'Variável local ao método';
      // $this->nome - acessa nome fora do método
      return $this->nome . ' ' . $this->sobreNome;
   }
}

echo $homem->nomeCompleto()
```

Geralmente a programação é processada na sequência de cima para baixo, mas objetos, como funções, podem chamar uma classe que foi definida acima de onde ele se encointra.

Constantes de classe

Convenciona-se usar o nome das constantes com todas as letras em maiúsculas, palavras compostas separadas por sublinhado
Para chamar uma constante de fora de uma classe não há necessidade de instanciar a classe, apenas NomeClasse::NOME_CONSTANTE;
Para chamar de dentro da classe basta usar:
self::nomeDaConstante;

```
<?php

class MinhaClasse
{
   const CONSTANTE = 'valor constante';

   function mostrarConstante() {
      echo  self::CONSTANTE . "\n";
   }
}

echo MinhaClasse::CONSTANTE . "\n";

$classname = "MinhaClasse";
echo $classname::CONSTANTE; // A partir do PHP 5.3.0

$classe = new MinhaClasse();
$classe->mostrarConstante();

echo $classe::CONSTANTE; // A partir do PHP 5.3.0
```

Construtor

É um método especial que é executado automaticamente sempre que a classe é instanciada. Seu nome é iniciado com dois sublinhados '__": __construct(). É um dos chamados métodos mágicos, pois tem uma execução específica e é executado automaticamnente. Importante: o __construct() é obrigado a ter a visibilidade public.

É um método que utiliza o nome reservado __construct() e que não precisa ser chamado da forma convencional, pois é executado automaticamente quando instanciamos um objeto a partir de uma classe. Sua utilização é indicada para rotinas de inicialização. O método construtor se encarrega de executar as ações de inicialização dos objetos, como por exemplo, atribuir valores a suas propriedades.

```
class Humano
{
   private $nome;
   private $sobreNome;

   public function __construct($n, $s){
      $this->nome = $n;
      $this->sobreNome = $s;
   }

   public function nomeCompleto(){
      return $this->nome . ' ' . $this->sobreNome;
   }
}

$homem = new Humano('Ribamar', 'Sousa');
$mulher = new Humano('Fátima', 'Evangelista');

echo $homem->nomeCompleto();
echo $mulher->nomeCompleto();
```

Construtor da classe Noticia

```
class Noticia
{
   public $titulo;
   public $texto;

   public function __construct($valorTit, $valorTxt)
   {
      $this->titulo = $valorTit;
      $this->texto = $valorTxt;
   }
```

Método construtor na subclasse NoticiaPrincipal

```
class NoticiaPrincipal extends Noticia
{
   public $imagem;

   public function __construct($valorTit, $valorTxt, $valorImg)
   {
      parent::__construct($valor_tit, $valorTxt);
      $this->imagem = $valorImg;
   }
```

O método construtor da classe Noticia é herdado e executado automaticamente na subclasse NoticiaPrincipal. Porém, as "características específicas de NoticiaPrincipal não serão inicializadas pelo método construtor da classe pai". Outro detalhe importante: caso a subclasse NoticiaPrincipal tenha declarado um método construtor em sua estrutura, este mesmo método da classe Noticia não será herdado. Nesse caso podemos chamar o método construtor da classe Noticia, através de uma chamada específica: parent::__construct()

Destrutor

O método destrutor será chamado assim que todas as referências a um objeto forem removidas ou quando o objeto for explicitamente destruído ou qualquer ordem na sequência de encerramento.

Como os construtores, destrutores pais não serão chamados implicitamente pela engine. Para executar o destrutor pai, deve-se fazer uma chamada explicitamente a parent::__destruct() no corpo do destrutor.

No PHP chamar o destrutor é desnecessário, tendo em vista que ao fechar o script/programa ele encerra todas as referências aos objetos.

```
<?php

class MinhaClasseDestruivel {
  function __construct() {
     print "No construtor<br>";
     $this->name = "MinhaClasseDestruivel";
  }

  function __destruct() {
     print "Destruindo " . $this->name . "<br>";
  }
}

$obj = new MinhaClasseDestruivel();
```

Segurança das informações

getters e setters

Idealmente todas as propriedades devem ter visibilidade private ou protected. Caso precisemos que seja vista fora da classe então criamos um método para ler o valor de uma propriedade e outro para alterar o mesmo. Estes métodos são chamados de getter (ler) e setter(alterar).

https://www.youtube.com/watch?v=QaM22Qgo3gM&list=PLwXQLZ3FdTVEau55kNj_zLgpXL4JZUg8I&index=3

```php
<?php

class pessoa {
	// propriedade
	private $nome;

	// método setter
	function setNome($novoNome) {
		$this->nome = $novoNome;
	}

	// método getter
	function getNome() {
		return $this->nome;
	}
}

$p = new pessoa;
$p->setNome('Ribamar');
print $p->getNome();
```

Herança

Herança é quando uma classe é criada tendo outra como base.

Fazemos isso usando a palavra reservada **extends** e a classe que serve de base é chamada de classe pai/parent. A que herda é chamada de filha ou derivada e herda propriedades e métodos da classe pai.

```php
class Animal // Classe pai/base
{
   public $especie;
   public $peso;

   public function tipoEspecie(){
      return "Este animal é da espécie {$this->especie}"
   }
}

$animal = new Animal();
$animal->especie = "Mamíferos";
echo $animal->tipoEspecie();

class Mamifero extends Animal // Classe filha/derivada
{
   // Herda propriedades e métodos da classe Animal
   public $quantidadePernas;
   public temPelo;

   public function temQuantasPernas(){
      return "O animal da espécie {$this->especie} tem {$this->quantidadePernas}";
```

```
    }
}

$mamifero = new Mamifero();
$mamifero->especie = 'Cavalo'; // Observar que especie não está na classe filha, mas ela herda esta propriedade
$mamifero->quantidadePernas = 4;
echo $mamifero->temQuantasPernas();
```

Method Overriding

Definição de Method Overriding:
É quando a função da classe base é redefinida com o mesmo nome, assinatura e especificador de acesso (public ou protected) da classe derivada.

Overriding/substituição de métodos

As definições de função nas classes filhas substituem as definições com o mesmo nome nas classes pais. Em uma classe filha, podemos modificar a definição de uma função herdada da classe pai.

A razão de fazer isso é para prover funcionalidades adicionais sobre as definidas na classe base.

Exemplo prático: você tem uma classe com nome Bird da qual derivam duas outras classes: Eagle e Swift. A classe Bird tem definidos os métodos defined para eat, fly, etc, mas cada uma desstas classes será especializada para Eagle e Swift e deve ter seu próprio estilo de voar que você precisa override as funcionalidades de voar.

Exemplo de herança com pai – filho – neto

```
<?php

class Pai
{
	public function indexPai(){
		return 'Index pai';
	}
}

class Filho extends Pai
{
	public function indexFilho(){
		return 'Index filho';
	}
}

class Neto extends Filho
{
	public function indexNeto(){
		return 'Index neto';
```

```
	}
}
// Vou instanciar a classe Neto e com a instância vou trazer um método da classe pai sem instanciar ele
// E também trazer um método da classe filha sem ter instanciado ela

$neto = new Neto();
print $neto->indexPai();
print '<br>';
print $neto->indexFilho();
print '<br>';
print $neto->indexNeto();
```

Todo quadrado é um retângulo, mas tendo todos os lados iguais. Então vamos criar uma classe genérica retangulo e outra quadrado extendendo retãngulo

```
<?php
// Exemplo simples de herança e bem didático
class Pai
{
	public function indexPai(){
			return 'Index pai';
	}
}

class Filho extends Pai
{
	public function indexFilho(){
			return 'Index filho';
	}
}

class Neto extends Filho
{
	public function indexNeto(){
			return 'Index neto';
	}
}
// Vou instanciar a classe Neto e com a instância vou trazer um método da classe pai sem instanciar ele
// E também trazer um método da classe filha sem ter instanciado ela

$neto = new Neto();
print $neto->indexPai();
print '<br>';
print $neto->indexFilho();
print '<br>';
print $neto->indexNeto();


<?php
require 'Retangulo.php';
```

```
class Quadrado extends Retangulo
{

}

$q = new Quadrado();
print 'A área do quadrado é '.$q->getArea(4, 4);

print '<br><br>Perímetro é '.$q->getPerimetro(4,4);

print '<br><br>O lado do quadrado mede '.$q->l;
```

Convenções para PHP Orientado a Objetos

Ao seguirmos as convenções definidas por um framework estamos garantindo boa performance, facilidade de manutenção, baixa curva de aprendizagem ao contratar programadores, segurança e escalabilidade.

Um banco de dados que não atende a convenção pode gerar confusão e dificuldade de manutenção do código da aplicação.

Ao não utilizar a convenção de nomeclatura de banco de dados do framework é possível que o programador tenha trabalho para definir em cada model qual é a sua respectiva tabela. Imagine um cenário de 100 models ou mais…

Uma aplicação que segue as convenções de seu framework garante o principal objetivo da utilização da ferramenta: produtividade. Com isso também garante uma pequena curva de aprendizagem ao trocar ou expandir a equipe além, claro, dos fatores de segurança, performance e organização que são essenciais para uma aplicação robusta e escalável.
https://medium.com/@carloscarneiropmw/overview-das-conven%C3%A7%C3%B5es-do-laravel-d2e76b3db38a

Nomes de arquivos em php

Propriedades e métodos devem ter seu nome usando camelCase.

Cada classe com um arquivo exclusivo e nunca duas ou mais classes em um único arquivo

Nomes de arquivos de classes de forma similar aos nomes das classes em CamelCase

ClientesController.php

class ClientesController

Nomes de arquivos que não são de classes

nome.php
nome_arquivo.php

Para o php os nomes de classes são case insensitives, portanto ele considera Cliente igual a clientE, mas é importante usar as convenções

Nome de métodos, propriedades, funções e variáveis - camelCase

Propriedades - sempre declaradas no início da classe
Métodos - logo abaixo das propriedades

Objeto == instância, que é criado a partir de uma classe

Uso de chaves:

```
class Nome
{

}
```

Métodos

```
public function nome()
{

}
```

Laços

```
foreach()
{

}
```

if

```
if($x == 0) {

}
```

Instância/Objeto

Usar parêntesis ao final do nome da classe, ao instanciar
$obj = new NomeClasse();

Namespace

Como funciona o namespace no PHP

O objetivo principal do namespace é o de permitir que tenhamos duas ou mais classes com o mesmo nome sem conflito.

Napespaces surgiram no PHP 5.3

São mais indicados para grandes projetos, com uma grande quantidade de classes, mas também podem ser usado em pequenos projetos.

Namespaces possibilitam o agrupamento de classes, interfaces, funções e constantes, visando evitar o conflito entre seus nomes, atuando como um encapsulador para estes itens. Seu funcionamento é equivalente ao de diretórios em sistemas operacionais, onde dois arquivos de mesmo nome não podem existir em um único diretório, mas nada impede a existência de dois arquivos de mesmo nome localizados em diretórios distintos. Este mesmo princípio é aplicado no PHP através de namespaces, ao utilizar este recurso temos mais liberdade na hora de criar classes, funções e etc, não sendo mais necessário utilizar prefixo para diferenciar seus nomes.

Duas classes Produto convivendo sem conflito, mas cada uma em seu próprio namespace:

```
require 'classes/produto.php';
require 'models/produto.php';

// Com namespaces

$produtom = new \models\Produto();
$produtoc = new \classes\Produto();
$produto->mostrarDetalhes();
```

Imagine que temos duas classes Book, a primeira no namespace Bookstore\Domain e a segunda no Library\Domain .
Para resolver o conflito, podemos fazer assim:

```
use Bookstore\Domain\Book;
use Library\Domain\Book as LibraryBook;
```

Duas classes Book mas uma com um alias, ambas convivendo sem conflitos graças ao namespace

É definido usando a palavra-chave namespace no topo do arquivo. Deve ser o primeiro comando do arquivo e nada antes dele, com uma única exceção que é a função declare().

Mesmo que seja possível usar mais de um namespace por arquivo evite, pois é confuso e não é uma boa prática.

Geralmente o nome do namespace é o nome do próprio projeto.

Usar apenas um namespace por arquivo

```
<?php
namespace NomeProjeto\PrimeiraParte;

class Clientes{

}

// Em outro arquivo
use \NomeProjeto\PrimeiraParte;

$clientes = new Clientes();
```

ou
$clientes = new \NomeProjeto\PrimeiraParte\Clientes();

Exemplo:

Criar duas pastas:

classes
models

Criar uma classe Produto em classes e models, em arquivos produto.php

produto.php

```
<?php

class Produto{
    public function mostrarDetalhes(){
        echo 'Detalhes do produto da pasta classes';
    }
}
```

e

```
<?php

class Produto{
    public function mostrarDetalhes(){
        echo 'Detalhes do produto da pasta models';
    }
}
```

Criar no raiz um arquivo index.php

No index.php incluir as duas classes:

```
require 'classes/produto.php';
require 'models/produto.php';

$produto = new Produto();
```

Acusará erro.

Então adicionar o namespace:

```
<?php

namespace classes;

class Produto{
    public function mostrarDetalhes(){
        echo 'Detalhes do produto da pasta classes';
    }
}
```

e

```
<?php

namespace models;

class Produto{
    public function mostrarDetalhes(){
        echo 'Detalhes do produto da pasta models';
    }
}
```

No index.php

```
$produto = new \classes\Produto();
```

e

```
$produto = new \models\Produto();
```

Referência
Curso de PHPOO do Node Studio
https://www.youtube.com/watch?v=o2CXLk74ggE&list=PLwXQLZ3FdTVEau55kNj_zLgpXL4JZUg8I&index=13

Veremos na prática o uso dos namespaces, quando criamos o aplicativo simplest_mvc.

6 – PHP Moderno

Ferramentas que caracterizam o PHP moderno

https://www.youtube.com/watch?v=ZQVztAwoHUA

PSRs
- https://www.php-fig.org/
https://www.php-fig.org/psr/

Composer
https://getcomposer.org/

Packgist
https://packagist.org/

Git
https://git-scm.com/

GitHub
https://github.com/

PHPUnit
https://phpunit.de/

Monolog - ferramenta de logs
https://github.com/Seldaek/monolog

PHP so Jeito Certo - em vários idiomas
http://br.phptherightway.com/

VSCode
https://code.visualstudio.com

Funções anônimas

Traits - permite herança composta

Templates - twigs e

Servidor web nativo para ambientes de desenvolvimento

php -S localhost:8000

Vagrant

Docker

Microfrfameworks:
Siles
Lumen
Slim

PHP 7 e 8

Types, null coalesce e spaceship operator

DRY

SOLLID

Próximos recursos do PHP
https://wiki.php.net/rfc

Padrões de Projeto com PHP em português
https://designpatternsphp.readthedocs.io/pt_BR/latest/README.html

O perigo do extremismo
Um problema com regras e diretrizes na programação é que elas geralmente só servem a um propósito em um contexto específico. Saindo desse contexto, uma boa regra pode se tornar uma regra horrível. De fato, toda boa regra se torna ruim quando levada ao extremo.

O princípio KISS, que é um acrônimo para “Keep It Simple, Stupid”, é um bom e extremamente sábio princípio que geralmente é visto por pessoas experientes como um conselho muito bom a seguir, mas mesmo este grande princípio torna-se um perigo para um projeto, se levado ao extremo. Existe tal coisa como “muito simples” resultando em falta de funcionalidade necessária.

Use os frameworks atuais com moderação. Antes de usar verifique se é adequado.

Não somos obrigados a sempre usar padrões de projetos em nosso código. É importante usar quando eles no trará vantagens e evitar quando eles nos trouxer grande complexidade.

Assim também vale para a orientação a objetos. Alguns pequenos projetos não justifica o uso da POO.

Siga o PHP-FIG mas com critérios, de forma a collher algo de útil para você e sua empresa. Não para procurar fazer tudo que o grupo criou.

Fique sempre atento aos bons comportamentos para deixar seu código e aplicativo mais seguros.

http://br.phptherightway.com/
https://phpthewrongway.com/pt_br/
https://www.freecodecamp.org/news/this-is-what-modern-php-looks-like-769192a1320/
https://www.airpair.com/php/posts/best-practices-for-modern-php-development
https://github.com/odan/learn-php
https://medium.com/@FernandoDebrand/guia-pr%C3%A1tico-do-modern-php-desenvolvimento-e-ecossistema-c9715184e463
https://phpthewrongway.com/pt_br/

Por que usar um CMS para a criação de um site

Formas principais de fazer um site:

1. Criá-lo com arquivos HTML estáticos;
2. Programar todo o site, com uma linguagem de programação e um banco de dados;
3. Utilizar um Sistema de Gerenciamento de Conteúdo (CMS), como o WordPress, ou Joomla;

Criar páginas estáticas tem vários problemas:

- Adicionar uma página nova pode exigir alterações manuais em todas as outras páginas;
- Mudar elementos como menus e headers, também vão exigir mudanças em todas as outras páginas (a não ser que você use iframes, mas se você faz isso, meus pêsames);
- Quando o cliente decidir mudar o visual e estrutura do site, vai haver muito retrabalho. É melhor jogar tudo fora e começar outro novo;
- Fica muito mais fácil de acontecerem erros, devido a links quebrados, ou tags escritas incorretamente (afinal, qualquer um pode errar uma tag, às 3h da manhã, tendo que entregar tudo às 8h);
- Segurança e técnicas novas de otimização serão difíceis de serem adicionadas, devido aos primeiros itens desta lista;

Os bons CMS's já trazem funções como:

- Gerenciamento de usuários, logins, permissões
- Categorização de conteúdos
- Construção de menus
- Temas visuais para todas as páginas
- Mecanismos de imagens e vídeos para as páginas
- Sistemas de blogs
- Formulários de contato
- Tradução e internacionalização

https://webdevacademy.com.br/artigos/por-que-usar-cms/

Veja este artigo criando sites estáticos com bons recursos, que talvez justifique usar um site estático ao invés de dinâmico em alguns casos:
https://github.com/ribafs/mkdocs-ribafs

Porque usar um framework para criar aplicativos

Um framework é um conjunto de códigos comuns abstraídos de vários projetos com o objetivo de prover funcionalidades genéricas em um projeto que utilize esse framework. Através de programação, ele permite a customização dessas funcionalidades para torná-las mais específicas de acordo com a exigência de cada aplicação.

De forma resumida o framework é uma estrutura, uma fundação para você criar a sua aplicação. Em outras palavras o framework te permite o desenvolvimento rápido de aplicações (RAD), o que faz economizar tempo, ajuda a criar aplicações mais sólidas e seguras além de reduzir a quantidade de código repetido.

Os desenvolvedores utilizam frameworks por vários motivos, e o maior deles é para agilizar o processo de desenvolvimento. A re-utilização de código em vários projetos vai economizar muito tempo e trabalho… Isso é garantido pois o framework já traz uma série de módulos pré-configurados (e funcionando) para fazer as mais variadas e comuns

tarefas como envio de e-mails, conexão com o banco de dados, sanitização (limpeza) de dados e proteção contra ataques.

Não recomendado para:
- Programadores iniciantes
- Projetos pequenos

Pontos fortes:
- Qualidade do código do aplicativo
- Segurança
- Produtividade
- Fácil manutenção do código
- Documentação e material online fartos
- Pequeno tempo de desenvolvimento

7 – MVC no PHP

O MVC é o padrão de arquitetura mais usado na web e é requisito praticamente essencial para qualquer programador. Os principais frameworks o utilizam e alguns XMS também.

O MVC (Model View Controller) é um dos padrões de arquitetura mais utilizados atualmente. A maioria dos grandes frameworks e CMS o utilizam para separar o código em camadas lógicas. Cada camada tem uma resposabilidade. Veja abaixo.

Aqui é até redundante dizer que para os que estão querendo aprender sobre MVC, a experimentação prática dos exemplos é imprescindível, portanto experimente, altere, personalize e teste bastante até entender e ficar satisfeito.

Model
Representa a letra M do MVC. Nesta camada são realizadas as operações de validação, leitura e escrita de dados no banco de dados. É responsável por salvar e receber dados do banco de dados, como também efetua diversos processamentos com os dados.

Basicamente qualquer coisa para ler, alterar, salvar ou excluir dados é nesta camada. A camada Model é a camada que sofreu a maior transformação na versão 3.

Uma boa prática é trazer para esta camada tudo que diz respeito às regras de negócio, como cálculos ou validações de integridade de dados.

Controller
É o responsável pela integração entre as camadas Model e View. Basicamente a View irá realizar uma solicitação para o Controller como por exemplo uma coleção de dados ou a solicitação de remover algum item do banco e o Controller, por sua vez, irá enviar a instrução para a camada Model executar.

Controllers

Os controllers correspondem ao 'C' no padrão MVC. Após o roteamento ter sido aplicado e o controller correto encontrado, a ação do controller é chamada. Seu controller deve lidar com a interpretação dos dados de uma requisição, certificando-se que os models corretos são chamados e a resposta ou view esperada seja exibida. Os controllers podem ser vistos como intermediários entre a camada Model e View. Você vai querer manter seus controllers magros e seus Models gordos. Isso lhe ajudará a reutilizar seu código e testá-los mais facilmente.

Mais comumente, controllers são usados para gerenciar a lógica de um único model. Por exemplo, se você está construindo um site para uma padaria online, você pode ter um RecipesController e um IngredientsController gerenciando suas receitas e seus ingredientes. No CakePHP, controllers são nomeados de acordo com o model que manipulam. É também absolutamente possível ter controllers que usam mais de um model.

Os controllers fornecem uma série de métodos que são chamados de ações. Ações são métodos em um controller que manipulam requisições. Por padrão, todos os métodos públicos em um controller são ações e acessíveis por urls.

Nesta camada (Controller) também podemos realizar verificações que não se referem às regras de negócio, visto que a boa prática é manter as regras de
negócio no Model.

View
Representa a letra V do MVC. É a camada responsável por tudo que é visual, páginas, formulários, listagens, menus, o HTML em geral. Tudo aquilo que interage com o usuário deve estar presente nesta camada. Representadas por HTML.

A View não realiza operações diretamente com o banco de dados nem trata diretamente com o Model. Ela as solicita e e exibe através do Controller, que intermedia suas solicitações com o Model.

Fluxo das Informações no MVC
- Geralmente Nascem na View quando um usuário faz uma solicitação, clicando num botão submit ou num link ou entrando um link diretamente no navegador
- Então são recebidos num Router que ativa o devido action/controller
- Daí são enviadas para o Controller, que a filtra (se for o caso) e a envia para o Model
- O Model analisa de acordo com a solicitação (uma consulta ao banco) e a devolve ao Controller
- O Controler por sua vez devolve o resultado para a View
- E a View renderiza o resultado e o mostra para o usuário

Abordagem sobre as 3 camadas: [C]ontroller, [V]iew e [M]odel

Um exemplo bem organizado de uso do MVC é o Framework CakePKP, que traz as 3 camadas bem definidas e organizadas dentro da pasta "src".

De forma mais completa o fluxo das informações entre as 3 camadas acontece assim no CakePHP:
- O usuário clica num link para editar um registro
- O dispatcher (expedidor) verifica a URL requisitada (/cakes/comprar) e redireciona ao controller correto;
- O controller executa a lógica específica da aplicação. Por exemplo, verifica se o Ricardo está logado e tem acesso ao site;
- O controller também usa os models para acessar os dados da sua aplicação. Muitas vezes, os models representam as tabelas do banco de dados, mas podem representar registros LDAP, feeds de RSS ou até mesmo arquivos do sistema.
- Neste exemplo, o controller usa o model para trazer ao usuário as últimas compras do banco de dados;
- Depois que o controller fez sua mágica sobre os dados, ele repassa para a view. A view faz com que os dados fiquem prontos para a representação do usuário;
- Uma vez que a view tenha usado os dados provenientes do controller para construir a página, o conteúdo é retornado ao browser do usuário.

Benefícios

Por que usar MVC? Porque é um verdadeiro padrão de projeto (design pattern) e torna fácil a manutenção da sua aplicação, com pacotes modulares de rápido desenvolvimento. Elaborar tarefas divididas entre models, views e controllers faz com que sua aplicação fique leve e independente. Novas funcionalidades são facilmente adicionadas e pode-se dar nova cara nas características antigas num piscar de olhos. O design modular e separado também permite aos desenvolvedores e designers trabalharem simultaneamente, incluindo a habilidade de se construir um rápido protótipo. A separação também permite que os desenvolvedores alterem uma parte da aplicação sem afetar outras.

Se você nunca desenvolveu uma aplicação neste sentido, isso vai lhe agradar muito, mas estamos confiantes que depois de construir sua primeira aplicação em CakePHP, você não vai querer voltar atrás.

Referências
https://www.youtube.com/watch?v=VInLNcHm8tA&list=PLtxCFY2ITssBl_nihh4HC5-ZlnIPEpVQD - Curso de PHP + MVC grátis em 21 aulas
https://www.youtube.com/watch?v=vvS7JgEcmic - PHP MVC Fácil
https://www.youtube.com/watch?v=GlMZDMyy-jE&list=PLxNM4ef1Bpxiah1JPIqK1mkwi0h20EoQ1 - PHP7 com MVC
https://www.youtube.com/watch?v=2dqI8o6bvjM&list=PLLfNZbkxufIUsLRzQCCGaek4PxQB4RLGe - Curso de MVC com PHP OO
https://phpro.org/tutorials/Model-View-Controller-MVC.html

Rotas

Meu router preferido

Trabalha sozinho, sem precisar ficar adicionando ou removendo novas rotas.

Este exemplo de router captura a URL.

Quando estamos na view do aplicativo e ativamos a URL de alguma forma:
- Digitando algo no location do navegador
- Clicamos num dos links: menu, botão novo, editar, excluir

Então o Router recebe a solicitação e verifica:
- Nome do controller
- Nome do action
- parâmetros

Caso a solicitação confira com algum dos controllers, actions e parâmetros:
- Instanciará o controller e o chamará com o action e passará o parâmetro

Caso não exista o controller e/ou o action então.

Chamará o controller de erro e passará a devida mensagem de erro contendo controller e action para orientação do usuário

Tudo isso acontece sem a intervenção do programador, tudo de forma automática.

Tudo que foi dito acima pode ser constatado na prática e/ou no código Router.php.

Gostei muito deste exemplo de router e criei diversas customizações na tentativa tanto de enender quanto de simplificar.

Passei a enender melhor mas simplificar não consegui muito, o exemplo original é mais eficiente. Este Router praticamene é o original.

Tudo partiu deste aplicativo

https://github.com/panique/mini3

Algum código que derivou do mini3:

https://github.com/ribafs/livro-php-fontes/tree/master/MVC/ComBD/mini-framework
https://github.com/ribafs/tutoriais/tree/master/6PHPOO/MVC/mini-mvc7 (última customização)

Finalmente criei este em 12 etapas
https://github.com/ribafs/simplest-mvc

8 – Criando um aplicativo em PHPOO com MVC

Algo como um pequeno framework, com a finalidade de consolidar os conceitos vistos até aqui e preparando para usar um bom framework.

Comecei meu aprendizado do PHP com o paradigma estruturado e passei vários anos estudando e programando assim. Aprender orientação a objetos deu trabalho, mas mais trabalho deu criar meu primeiro aplicativo OO usando o padrão de arquitetura MVC.

Foi quando encontrei o aplicativo mini3:

https://github.com/panique/mini3

Entendi, consegui rodar com sucesso e era simples de forma que logo eu estava criando algumas customizações.

Finalmente criei um com 12 fases, partindo da primeira fase que nem acessava banco de dados mas apenas um array. Fui avançando nas fases de forma didática para tornar o entendimento mais fácil.

Um dos aplicativos que criei partindo do mini3, foi o mini-mvc

https://github.com/ribafs/mini-mvc

Usei este para criar o nosso aplicativo, que o chamarei app-php-mvc, e que estarei abrigando no repositório

https://github.com/ribafs/phpoo-livro

Criação do aplicativo app-php-mvc

Requisito

Precisa ter o Apache com mod_rewrite ativado para fundionar. Ou outro servidor web.

Não funciona com apenas o servidor web nativo do PHP.

Caso não tenha o apache rodando e com mod_rewuite ativo conseguirá abrir o index.php normalmente mas nenhum link funcionará.

Criando o aplicativo do "zero", passo a passo:

Criar a pasta

c:\xampp\htdocs\app-php-mvc

ou

/var/www/html/app-php-mvc

Criar, dentro da pasta o arquivo

app-php-mvc/.htaccess

Que deve conter

```
# Habilita o mod_rewrite e Redireciona todos os requests para o diretório public/
RewriteEngine on
RewriteRule ^(.*) public/$1 [L]
```

Testar pelo navegador

http://localhost/app-php-mvc

Receberemos a mensagem

Internal Server Error

Esta mensagem é provocada pelo .htaccess, por conta de não existir a pasta public

Criemos a pasta

public

Testar novamente pelo navegador

http://localhost/app-php-mvc

Receberemos novamente a mensagem acima

Agora vamos criar um outro .htaccess, este dentro da pasta public

public/.htaccess

```
Options -MultiViews
RewriteEngine On
Options -Indexes
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule ^(.+)$ index.php?url=$1 [QSA,L]
```

Testar novamente pelo navegador

http://localhost/app-php-mvc

Agora receberemos outra mensagem:

Forbidden

You don't have permission to access this resource.

Veja que o .htaccess do public tenta acessar o index.php e não encotnra.

Obs.: Não estranhe de tanto fazer estes testes que só levam a erros. Acho importante se acostumar com estas mensagens, pois geralmente nos deparamos com elas.

Agora vou criar o arquivo

public/index.php

Contendo

```
<?php
// FrontController, única entrada para o aplicativo

if(!file_exists('../vendor')) {
    die ('<h3>Precisa executar antes o comando<br>composer du</h3>');
}

define('DS', DIRECTORY_SEPARATOR);

// Captura o path completo do aplicativo. DIRECTORY_SEPARATOR adiciona uma barra ao final do path
define('ROOT', dirname(__DIR__) . DS);

// Captura a pasta do projeto: path full mais src, como '/var/www/html/mini-framework2/src'.
define('APP', ROOT . 'App' . DS);
define('CORE', ROOT . 'Core' . DS);

// Este é o auto-loader para as dependências do Composer (para atualizar o namespace em seu projeto execute: composer dumpautoload).
require_once ROOT . 'vendor/autoload.php';

// Carregar as configurações da aplicação (error reporting etc.)
require_once CORE . 'config.php';

// Carregar a classe Router
use Core\Router;

// Iniciar a aplicação através do Router
$router = new Router();

Quero chamar a atenção para as linhas:

if(!file_exists('../vendor')) {
    die ('<h3>Precisa executar antes o comando<br>composer du</h3>');
}
```

Quando baixamod este projeto e vários outros do Github, geralmente o .gitignore impede que baixemos o vendor, com todas as dependências, então ao tentar executar pela primeira vez ele acusará um erro chato, dizendo qua não encontrou a pasta vendor com o autoload, mas isso tudo em inglês e ao meu ver devemos deixar sempre mais amigável as mensagens de erro dos nossos projetos. As três linhas acima verificam se a pasta vendor existe no raiz, caso não exista diz para que se execute o comando

composer du

Este arquivo cria algumas cosntantes que serão usadas no aplicativo.

Depois importa o autoload

Também importa o arquivo Core/config.php, que cria algumas constantes para o aplicativo, outras cosntantes para a conexão com o banco de dados.

E finalmente verifica se a constante DEBUG está true. Se sim ele ativa as mensagens de erro e se não as desativa e ativa gravação dos logs.

Depois ele inclue a classe Core/Router.php

E finalmente a instancia.

A classe Router contém dois métodos, o splitUrl() e o construtor. O método construtor chama o método splitUrl logo no início.

O método splitUrl() pega a requisição recebida pela URL e a divide em três partes:
controller/action/parameter

É bom observar que todo o código do Router.php está dentro do cosntrutor. Isso foi feito intencionalmente, para que no momento em que a classe seja instanciada todo o seu código já funcione.

Vejamos o que faz a classe Router

http://localhost/app-php-mvc/controller/action/parameter

Os únicos actions que valem são:
index, add, edit e delete

index e add não usam parâmetros
edit e delete usam e são os respectivos números dos registros. Qualquer número diferente de um registro não será reconhecido.

http://localhost/app-php-mvc/

- Primeiro ela testa se o comprimento da parte controller da URL é zero. Se for ele atribui para o controller o valor default (que existe no config.php) e também o action default, que é o index

- Se o comprimento do controller não for zero, ou seja, se o usuário digitar algo, ele verifica se o controller digitado pelo user existe na pasta App/Controllers. Se existir o arquivo ele instancia sua classe.

- Agora ele vai verificar se tem um action na URL. Se existir então verifica se existe um parâmetro. Caso o parâmetro seja vazio instanciará o controller somente com o action

- Caso o controller exista e não exiswta o action ele verificará se foi digitado algum action

- Caso não tenha sido digitado ele atribuirá o action default, que é o index

- Se tiver digitado o controller que existe e um action que não existe ele emitirá uma mensagem de erro avisando ao usuário

- Finaliza quando ele verifica que o controller digitado não existe, então envia o controller ErrorController para avisar ao user que o controller não existe.

Veja que esta classe é responsável por receber as rotas digitadas pelo usuário e encaminhar devidamente para o respectivo recurso.

Antes de continuar.

Testar novamente pelo navegador

http://localhost/app-php-mvc

Agora veremos a mensagem disparada pelas 3 linhas.

Então vamos criar o composer.json, que nos ajudará a criar o autoload com o PSR-4 para os namespaces.

Criar no raiz o arquivo

composer.json

Contendo

```
{
    "name": "ribafs/app-php-mvc",
    "description": "Aplicactivo com PHPOO usando MVC com rotas",
    "type": "project",
    "license": "MIT",
    "authors": [
        {
            "name": "Ribamar FS",
            "email": "ribafs@gmail.com"
        }
    ],
          "minimum-stability": "stable",
  "require":{
  },
    "autoload":
    {
        "psr-4":
        {
            "App\\" : "App/",
            "Core\\" : "Core/"
        }
    }
}
```

Este arquivo diz ao composer para definir os dois namespaces App e Core que estão ligados as pastas App e Core

Faça as devidas adaptações para você.

Agora vamos criar as pastas no raiz:

App
Core

Então teremos na pasta app-php-mvc a estrutura abaixo com 3 pastas e o .htaccess

App
Core
public

Agora executar o comando

composer du

Receberemos

Generating autoload files
Generated autoload files

Testar novamente pelo navegador

http://localhost/app-php-mvc

Agora ele reclama que o arquivo config.php incluído não existe.

Core/config.php

Então vamos criar o Core/config.php

Observe a convenção, os arquivos que contém classes tem inicial maiúscula, os que não contém classe iniciam com minúsculas.

Testar novamente pelo navegador

http://localhost/app-php-mvc

Agora ele nos pede o Core/Router.php

Então vamos criar o Core/Router.php

Vale a pena dar uma olhada com calma na classe Core/Router.

Testar novamente pelo navegador

http://localhost/app-php-mvc

Agora ele pede App\Controllers\ClientsController

Criarei então

Testar novamente pelo navegador

http://localhost/app-php-mvc

Agora ele pede o Core/Controller.php

Criemos

Testar novamente pelo navegador

http://localhost/app-php-mvc

É interessante ir criando cada arquivo, passo a passo, pois assim percebemos o caminho percorrido pelas informações.

Como o Core/Controller.php usa o Model.php, agora ele pede Core/Model.php

Criemos

Testar novamente pelo navegador

http://localhost/app-php-mvc

Como o Model.php usa o Connection.php ele nos pede Core/Connection.php

Criemos

Como o Core/Controller.php usa as views, então ele nos pede
App/views/templates/header.php, aproveitarei e criarei os 3 templates.

App/views/templates

Agora ele já usa o header.php e o menu.php e os mostra na tela mas pede o App/views/clients/index.php, criarei os 3 arquivos: index, add e edit.

Agora ele já me mostra a listagem do index.

Mas se clicarmos em edit ele reclamará que o model App\Models\ClientsModel. Criarei.

Acredito que agora já posso criar o projeto completo. E também acredito que o código não seja tão trabalhoso de entender.

A classe que mais me deu trabalho de entender foi a Router, mas como entendi agora expliquei com detalhes o que fica mais simples para você.

Mas é bom lembrar que temos um forum para discutir sobre este livro e sobre este aplicativo no repositório do Github.

https://github.com/ribafs/phpoo-livro/discussions

Referências

Curso de PHPOO da Node Studio - 27 aulas
https://www.youtube.com/playlist?list=PLwXQLZ3FdTVEau55kNj_zLgpXL4JZUg8I

Curso gratuito de Laravel 9 do tiago Matos - 34 aulas
https://www.youtube.com/playlist?list=PLcoYAcR89n-reidRFA3XCIvQPeKFt4dQU

[Live] PHP e Laravel na Prática: criando um projeto open source do zero, com Laravel 8 e Github
https://www.youtube.com/watch?v=UngvQxEsK_M
https://github.com/erikaheidi/streamaru

Clean Code
https://github.com/jupeter/clean-code-php

Descobrindo o PHP Moderno
https://www.youtube.com/watch?v=voiBMLEpHsM

Crea tu framework MVC con PHP - 17 aulas
https://www.youtube.com/playlist?list=PLty0cFLf07jXQA5_P9rDMWjpEet2wTXN1

9 - Exceções

Tratamento de exceções

O PHP possui um modelo de exceções similar ao de outras linguagens de programação. Uma exceção pode ser lançada (throw) e capturada (catch). Código pode ser envolvido por um bloco try para facilitar a captura de exceções potenciais. Cada bloco try precisa ter ao menos um bloco catch ou finally correspondente.

catch

Múltiplos blocos catch podem ser utilizados para capturar exceções diferentes. A execução normal (quando nenhuma exceção é lançada dentro de um try) irão continuar a execução após o último catch definido em sequência. Exceções podem ser lançadas (ou relançadas) dentro um bloco catch.

Quando uma exceção é lançada o código seguinte não é executado, e o PHP tentará encontrar o primeiro bloco catch coincidente. Se uma exceção não for capturada, um erro PHP fatal será lançado com a mensagem "Uncaught Exception ..." na ausência de uma função definida com set_exception_handler().

A partir do PHP 7.1 um bloco catch pode especificar múltiplas exceções usando o caractere pipe (|). Isto é útil quando diferentes exceções de diferentes hierarquias de classes são tratadas da mesma forma.
finally

A partir do PHP 5.5, um bloco finally pode ser especificado após ou no lugar de blocos catch. Códigos dentro de finally sempre serão executados depois do try ou catch, independente se houve o lançamento de uma exceção, e antes que a execução normal continue.

O objeto lançado precisa ser uma instância da classe Exception ou uma subclasse de Exception. Tentar lançar um objeto sem essa ascendência resultará em um erro fatal.

Exception é a classe base para todas as exceções.

```
class Newsletter
{
   public function cadastrarEmail($email){
      if(!filter_var($email, FILTER_VALIDATE_EMAIL)){
         throw new Exception ('E-mail inválido', 1);
      }else{
         echo 'E-mailc adastrado com sucesso';
      }
   }
}

$nl = new Newsletter();
try {
   $nl->cadastrarEmail('riba@');
```

```
}catch(Exception $e){
    echo 'Mensagem; '.$e->getMessage();// code, file, message, line
    echo '<br>Código: '.$e->getCode();
    echo '<br>Arquivo: '.$e->getFile();
    echo '<br>Linha: '.$e->getLine();
}


<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('Divisão por zero.');
    }
    return 1/$x;
}

try {
    echo inverse(5) . "\n";
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Exceção capturada: ',  $e->getMessage(), "\n";
}

// Execução continua
echo "Olá mundo\n";
?>


<?php
function inverse($x) {
    if (!$x) {
        throw new Exception('Divisão por zero.');
    }
    return 1/$x;
}

try {
    echo inverse(5) . "\n";
} catch (Exception $e) {
    echo 'Exceção capturada: ',  $e->getMessage(), "\n";
} finally {
    echo "Primeiro finaly.\n";
}

try {
    echo inverse(0) . "\n";
} catch (Exception $e) {
    echo 'Exceção capturada: ',  $e->getMessage(), "\n";
} finally {
    echo "Segundo finally.\n";
}

// Execução continua
```

```
echo "Olá mundo\n";
?>


try {
    throw new Exception('Hello world');
} catch (Exception $e) {
    echo 'Uh oh! ' . $e->getMessage();
} finally {
    echo " - I'm finished now - home time!";
}
```

Exemplo de try catch em classe de conexão

```
<?php
$host = '127.0.0.1';
$db   = 'cadastro';
$user = 'root';
$pass = 'root';

$opt = [
    PDO::ATTR_ERRMODE            => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES   => false,
];

try {
    $pdo = new PDO("mysql:host=$host;dbname=$db", $user, $pass, $opt);
    /*** echo a message saying we have connected ***/
//    echo 'Conectado para o banco de dados<br />';

    /*** close the database connection ***/
//    $pdo = null;
}catch(PDOException $e){
    echo $e->getMessage();
}
```

Os pilares da orientação a objetos

O paradigma orientado a objetos é um dos mais utilizados nas linguagens de programação. Neste artigo, veremos os pilares da orientação a objetos.

O paradigma orientado a objetos é um dos paradigmas mais utilizados no mercado de trabalho. Além de ser um dos primeiros paradigmas com o qual nós temos contato quando começamos a estudar desenvolvimento de software, a maioria das linguagens utilzadas pela indústria em geral possui uma forte base orientada a objetos, o que faz com que seja essencial o domínio deste paradigma. Neste artigo, vamos verificar quais são os pontos principais do paradigma orientado a objetos.

Classes e objetos
Em linguagens orientadas a objeto, nós organizamos a maior parte do nosso código em estruturas chamadas classes.

Você pode entender uma classe como sendo inicialmente um molde, molde este que geralmente representa alguma estrutura do mundo real com a qual nosso código terá que lidar. Vamos imaginar que estejamos trabalhando com uma aplicação que lida com carros...

Provavelmente, nossa aplicação irá lidar com carros e, por causa disso, precisaremos de um molde para definirmos o que é um carro dentro do nosso código. Esse molde seria responsável por estabelecer *o que é um carro* e *o que pode fazer um carro*.

Detalhes:
https://www.treinaweb.com.br/blog/os-pilares-da-orientacao-a-objetos/

10 – Novidades do PHP 7 e 8

PHP 7

1. Desempenho Fantástico

O PHP 7 teve seu motor remodelado. Com isso, houve um grande ganho de desempenho.

Em alguns casos, é possível alcançar até 9 vezes mais velocidade. Mas esse número pode variar conforme a plataforma e a aplicação utilizada nos testes.

Eu utilizei o script para benchmark criado pela própria equipe do PHP, disponibilizado junto com o código-fonte da linguagem. Em comparação com a versão 5.6, consegui aproximadamente 9 vezes mais velocidade usando o PHP 7. Expliquei esse teste com mais detalhes neste meu artigo.

2. MySQL Removido

Desde o PHP 5.5, as funções mysql_* eram consideradas obsoletas. Ou seja, tudo indicava que elas seriam removidas em um futuro bem próximo.
Pois bem. A hora chegou.

No PHP 7, as funções mysql_* (como mysql_connect(), mysql_query() e outras) deixaram de existir. Agora é preciso utilizar MySQLi ou PDO.
Recomendo utilizar PDO, por ser mais robusta e ser independente de SGBD.

3. Funções ereg_* Removidas

Além das funções mysql_*, as funções ereg_* e eregi_* (como ereg(), ereg_replace, eregi() e outras) eram consideradas obsoletas desde o PHP 5.3.
Elas também foram removidas no PHP 7.

Agora é preciso usar as funções preg_*, da biblioteca PCRE, como preg_match e preg_replace.

As funções preg_* exigem delimitadores. Consequentemente, é possível utilizar modificadores, como "i" e "u". O "i", por exemplo, significa case-insensitive. Ou seja, se você usava eregi_*, passará a usar preg_*, sempre com o modificador "i".

3. Erros Fatais e Exceções

No PHP 7, erros fatais passaram a ser Exceções. Isso quer dizer que eles podem ser tratados em bloco try/catch, sem interromper a execução do script.
Para exemplificar, vamos executar este código (no PHP 7, não no PHP 5):

Veremos este erro:
E o texto "FIM" não será exibido. Isso ocorre pois a exceção interrompe o script.
Agora execute este script:

Ou seja, nossa aplicação tratou a exceção e a execução continuou normalmente.

4. Construtores do PHP 4 Obsoletos
Antes do PHP 5, os construtores recebiam o mesmo nome da classe. Por exemplo:

Isso continuou funcionando no PHP 5, mas era recomendado usar o método __construct, ficando desta forma:
O PHP 7 recomenda que seja usado método __construct em vez do método com o mesmo nome da classe. Ou seja, o uso de construtores no padrão do PHP 4 continuará sendo possível, mas é um recurso obsoleto (Deprecated).

https://tableless.com.br/10-novidades-do-php-7/

Depreciações

As seguintes funções/funcionalidades serão obsoletas com o PHP 7.4. Para uma lista mais abrangente de depreciações, confira as Notas de Atualização do PHP 7.4.

Alterar a prioridade do operador da concatenação

Atualmente, em PHP os operadores aritméticos “+” e “-“, e o operador de string “.” são associativos à esquerda e têm a mesma precedência (leia mais sobre Precedência do Operador).

Como exemplo, considere a seguinte linha:
echo "sum: " . $a + $b;

No PHP 7.3 este código produz o seguinte aviso:

Warning: A non-numeric value encountered in /app/types.php on line 4

Isto porque a concatenação é avaliada da esquerda para a direita. É o mesmo que escrever o seguinte código:
echo ("sum: " . $a) + $b;

Esta RFC propõe alterar a precedência dos operadores, dando a “.” uma precedência inferior à dos operadores “+” e “-“, para que as adições e subtracções sejam sempre efectuadas antes da concatenação da string. Essa linha de código deve ser equivalente ao seguinte:
echo "sum: " . ($a + $b);

Esta é uma proposta em duas etapas:
- A partir da versão 7.4, o PHP deve emitir um aviso de obsolescência quando encontrar uma expressão não incluída entre parênteses com “+”, “-” e “.
- A mudança real de precedência destes operadores deve ser adicionada com o PHP 8.

Ambas as propostas foram aprovadas por larga maioria de votos.

Depreciar operador ternário associativo-esquerdo

No PHP o operador ternário, ao contrário de muitas outras linguagens, é deixado-associativo. De acordo com Nikita Popof, isso pode ser confuso para programadores que alternam entre diferentes linguagens.

Atualmente, em PHP o seguinte código está correto:

$b = $a == 1 ? 'one' : $a == 2 ? 'two' : $a == 3 ? 'three' : 'other';

É interpretado como:

$b = (($a == 1 ? 'one' : $a == 2) ? 'two' : $a == 3) ? 'three' : 'other';

E isso pode levar a erros, porque pode não ser o que pretendemos fazer. Assim, esta RFC propõe depreciar e remover o uso da associatividade esquerda para operadores ternários e forçar os desenvolvedores a usar parênteses.

Esta é outra proposta de duas etapas:

- A partir do PHP 7.4, os operadores ternários aninhados sem o uso explícito de parênteses geram um aviso de depreciação..
- A partir do PHP 8.0, haverá um erro de execução de compilação.

Esta proposta foi aprovada com 35 a 10 votos a favor.
https://kinsta.com/pt/blog/php-7-4/

Arrays de constantes utilizando a função define()

Constantes do tipo array agora podem ser definidas com a função define(). No PHP 5.6 elas só poderiam ser definidas com const.

```
<?php
define('ANIMAIS',[
    'gato',
    'cachorro',
    'pombo'
]);

echo ANIMAIS[0];
```

Arrow functions

São funções anônimas escritas de forma mais suscinta

Introduzidas no PHP 7.4

Funcionam de forma semelhante as closures mas capturam automaticamente as variáveis globais

```
$x = 20;
$y = 30;

$arrFunction = fn($z) => "$x - $y - $z";

echo $arrFunction(10);
```

Usam a palavra reservada fn mas não precisam de return nem de chaves

Classes Anônimas foram

Introduzidas pelo PHP 7

Uma classe que não tem nome e somente faz sentido quando queremos instanciar apenas um único objeto de uma classe e uma única vez.

Classe normal

```
class Classe1
{
    public function teste(){
        print 'normal';
    }
}

$classe = new classe();
```

Classe anônima, já inicia com o objeto recebendo a classe

```
$obj = new class
{
    public function teste(){
        print 'anônima';
    }
};

// Observe que precisamos terminar com o ;

$obj->teste();
```

Divisão de inteiros com intdiv()

A nova função intdiv() realiza a divisão de inteiros de seus operandos e a retorna.

```
<?php
var_dump(intdiv(10, 3));
```

O exemplo acima irá imprimir:
int(3)

Sintaxe de escape de códigos Unicode

Toma um código Unicode em sua forma hexadecimal e imprime esse código em UTF-8 em uma string delimitada por aspas ou um heredoc. Qualquer código válido é aceito, com os zeros à esquerda sendo opcionais.

```
<?php
echo "\u{aa}<br>";
echo "\u{0000aa}<br>";
echo "\u{9999}<br>";
```

Saídas:

ª
ª (o mesmo que antes mas com os zeros à esquerda opcionais)
香

OPERADOR NULL COALESCE

O operador Null Coalesce é representado pela linguagem PHP com a utilização de dois sinais de interrogação, por exemplo:

< variável > = < valor > ?? < há valor definido? Senão retorna isso >;

$x = $a ?? 10;

Se a variável $a existir, o valor da mesma será definido para $x, do contrário, o número 10 é que será utilizado.

É comum a necessidade de verificarmos se um determinado valor está condito num Array e então, fazer uso do mesmo. Atualmente, é necessário o uso de uma condição para averiguar que o item existe, ou então, verificar que a variável é diferente de Null, para que somente então, possamos fazer uso da mesma. Até porque, se utilizarmos uma variável que contém Null é será levantado uma exceção e o Script terá sua execução interrompida.

O operador Null Coalesce veio para agilizar essa constante necessidade de verificação e utilização do valor contido no item ou variável checado.

A sintaxe do operador Null Coalesce é bastante semelhante a sintaxe do Operador Ternário, porém, com este operador, é verificado se a referência existe ou então, se um elemento esta contido num Array. Caso o valor exista, o mesmo será utilizado, do contrário, iremos definir algum outro valor ou então, alguma outra verificação.

É interessante observar que o operador Null Coalesce fornece uma estrutura para checagem e utilização, porém, é possível continuarmos trabalhando sem fazer uso do mesmo.

A seguir utilizamos algumas estruturas e também, a estrutura do operador Null Coalesce.

operador ternário

```
echo $x ? $x : "";

echo $x ?: "";
```

operador Null Coalesce

```
echo $x ?? "";
echo $x ?? $b ?? "";
```

No exemplo seguir, utilizamos o operador Null Coalesce para definir o que será impresso na saída padrão.

```
// $a não está definido
$b = 10;

echo $a ?? 2; // saída 2
echo $a ?? $b ?? 7; // saída 10
```

http://excript.com/php/null-coalescing-operator-php.html

Operador "nave espacial" (spaceship)

O operador nave espacial é utilizado para comparação entre duas expressões. Retornará respectivamente -1, 0 ou 1 quando $a for menor que, igual a, ou maior que $b. As comparações são feitas de acordo com a já conhecida regras de comparação de tipos do PHP.

```
<?php
// Integers
echo 1 <=> 1; // 0
echo '<br>';
echo 1 <=> 2; // -1
echo '<br>';
echo 2 <=> 1; // 1
echo '<br>';
// Floats
echo 1.5 <=> 1.5; // 0
echo '<br>';
echo 1.5 <=> 2.5; // -1
echo '<br>';
echo 2.5 <=> 1.5; // 1
echo '<br>';
// Strings
echo "a" <=> "a"; // 0
echo '<br>';
echo "a" <=> "b"; // -1
echo '<br>';
echo "b" <=> "a"; // 1
```

https://rafaelti.wordpress.com/2017/06/01/php-7-operador-nave-espacial-spaceship/

Propriedades tipadas

A versão 7 fez várias melhorias no sistema de tipos do PHP. A versão 7.4 adiciona suporte para propriedades tipadas, onde podemos usar quando quisermos declarar tipos para propriedades das classes. As propriedades de classe não exigem que declaremos explicitamente os métodos getter/setter para elas. Algumas das características mais importantes são:

- Os tipos também podem ser declarados em propriedades static;
- Referências também possuem suporte a propriedades tipadas;
- As propriedades tipadas são afetadas pela diretiva strict_types, assim como os tipos de parâmetro e de retorno;
- Os tipos podem ser usados com notação var;
- Os valores padrão para propriedades tipadas podem ser declarados;
- O tipo de propriedades múltiplas pode ser declarados em uma única declaração;
- int e um float são fundidos implicitamente;
- O tipo de uma propriedade anulável pode ser declarado;
- Uma propriedade de classe do tipo callable ou void não pode ser declarada.

Como exemplo, considere a seguinte classe chamada Catalog, que ilustra várias das características anteriores:

```
<?php
declare(strict_types=1);

class Catalog {
    public int $catalogid,$journalid;
    public ?string $journal=null;
```

```
    public static string $edition="January-February 2020";
    var bool $flag;
    public float $f=1;

    public function __construct(int $catalogid,int $journalid,string $journal,bool $flag)
    {
        $this->catalogid = $catalogid;
        $this->journalid = $journalid;
        $this->journal = $journal;
        $this->flag = $flag;
    }
}

$c = new Catalog(123,345,"PHP Magazine",true);
echo "Catalogid: ".$c->catalogid."<br>";
echo "Journalid: ".$c->journalid."<br>";
echo "Journal: ".$c->journal."<br>";
echo "Flag: ".$c->flag."<br>";
echo "Edition: ".Catalog::$edition."<br>";
?>
```

https://www.infoq.com/br/articles/php7-new-type-features/

Typed Properties 2.0

As declarações de tipo de argumento, ou dicas de tipo, permitem especificar o tipo de uma variável que se espera que seja passada para uma função ou um método de classe. Dicas de tipo são um recurso disponível desde o PHP 5, e desde o PHP 7.2 podemos usá-las com o tipo de dados do object. Agora o PHP 7.4 traz o type hinting um passo à frente adicionando suporte para declarações de tipo de propriedade de primeira classe. Aqui está um exemplo muito básico:

```
class User {
        public int $id;
        public string $name;
}
```

Todos os tipos são suportados, com exceção de void e callable:

```
public int $scalarType;
protected ClassName $classType;
private ?ClassName $nullableClassType;
```

A RFC explica o motivo pelo qual não há suporte para void e callable:

O tipo de vazio não é suportado, porque não é útil e tem semântica pouco clara.
O tipo de chamada não é suportado, porque seu comportamento é dependente do contexto.

Assim podemos usar com segurança bool, int, float, string, array, object, iterable, self, parent, qualquer classe ou nome de interface, e tiposanuláveis (? type).

Os tipos podem ser usados em propriedades estáticas:

```
public static iterable $staticProp;
```

Também são permitidos com a notação var:

```
var bool $flag;
```

É possível definir valores de propriedade padrão, que obviamente devem corresponder ao tipo de propriedade declarado, mas somente propriedades anuláveis podem ter um valor null padrão:

```
public string $str = "foo";
public ?string $nullableStr = null;
```

O mesmo tipo aplica-se a todas as propriedades numa única declaração:

```
public float $x, $y;
```

O que acontece se fizermos um erro no tipo de propriedade? Considere o seguinte código:

```
class User {
	public int $id;
	public string $name;
}

$user = new User;
$user->id = 10;
$user->name = [];
```

No código acima, declaramos um tipo de propriedade string, mas definimos um array como valor de propriedade. Neste cenário, obtemos o seguinte erro Fatal:
Fatal error: Uncaught TypeError: Typed property User::$name must be string, array used in /app/types.php:9
Esta RFC foi aprovada com 70 votos a 1 voto.

Usando void como tipo de retorno para funções

Como mencionado, o suporte para declarações do tipo de retorno foi adicionado no PHP 7.0, enquanto o PHP 7.1 introduziu o tipo de retorno void. Em uma função que retorna void, a instrução do retorno deve estar vazia ou ser totalmente omitida. O NULL não deve ser confundido com o tipo void, já que NULL é um valor do tipo nulo, enquanto void implica a ausência de um valor.

Para demonstrar o uso de um tipo de retorno void, vamos criar um script chamado hello-void.php e copiar o seguinte código abaixo:

```
<?php
function hello(): void
{
        echo 'Hello';
        return;
}
echo hello();
echo "<br/>";
```

O script declara uma função chamada hello() com o tipo de retorno void. A função produz uma string 'Hello' e faz um retorno vazio. Se o tipo de retorno for void, a função não deve retornar um valor. Execute o script, e receba "Hello" como resultado.

https://www.infoq.com/br/articles/php7-new-type-features/

Declarações de tipo de retorno

O PHP 7 adiciona suporte a declarações de tipo de retorno. Similar às declarações de tipo de argumento as declarações de tipo de retorno especificam o tipo do valor que será retornado por uma função. Os mesmos tipos estão disponíveis para declarações de tipo de retorno assim como estão disponíveis para declarações de tipo de argumentos.

```
<?php
function Reverse(string $a) : string
{
        return strrev($a);
}

try{
  echo Reverse('hello');
}catch (TypeError $e) {
        echo 'Error: '.$e->getMessage();
}
// olleh
```

A documentação completa e exemplos de declarações de tipo de retorno podem ser encontradas na referência de declarações de tipo de retorno .
https://www.infoq.com/br/articles/php7-new-type-features/

Try ... catch com vários blocos catch. Novidade da versão 7.1.0 do PHP

```
<?php

try {
	//code...
} catch (Exception1 | Exception2 $ex12) {
	// handle exception 1 & 2
} catch (Exception3 $ex3) {
	// handle exception 3
}
```

Ignorando a variável de exceção. Novidade da versão 8.0

```
<?php

try {
	//code...

} catch (Exception) {
	// handle exception
}
```

PHP 8

Novas Funções no PHP 8

Para finalizarmos nossa lista, não podemos deixar de comentar sobre três novas funções que serão acrescentadas ao PHP. São elas:

- str_contains
- str_starts_with() e str_ends_with()
- get_debug_type

A função str_contains

A nova função str_contains permite realizar uma busca dentro de uma string.

Sua síntaxe será como no exemplo abaixo:
str_contains (string $haystack , string $needle) : bool

Esta sintaxe significa que será executado uma verificação para indicar se $needle está presente dentro da string $haystack. Caso sim, ela retornará o valor booleano true. Caso não esteja, então, retornará false.

Portanto, agora podemos utilizar a função str_contains para escrever o código como no exemplo abaixo:

```
$string = 'Frase de exemplo';

$verificar= 'exemplo';
```

```
if (str_contains($string, $verificar)) {
	echo "A String foi encontrada";
} else {
	echo "A String não foi encontrada";
}
```

Como você pode perceber, isso tornará a busca dentro de uma string mais legível e menos propenso a erros.

Você pode estar lendo a RFC sobre essa função para poder verificar todas suas características.

As funções str_starts_with() e str_ends_with()

As funções str_starts_with() e str_ends_with() funcionam parecidos com a função anterior, a str_contains. Porém, a diferença é que elas verificam se um string começa ou termina com determinada string.

Sua similaridade com str_contains também se dá pela sintaxe. Veja nos códigos de exemplo abaixo a sintaxe da str_starts_with() e str_ends_with():

```
str_starts_with (string $haystack , string $needle) : bool
str_ends_with (string $haystack , string $needle) : bool
```

Portanto, quando você utilizar essa funções, será possível economizar no uso da CPU. Isso acontece pois não será necessário percorrer por toda uma string, a função irá verificar apenas o inicio ou o final.

A função get_debug_type

A nova função que chegará junto ao PHP 8 é a get_debug_type. Com ela, você poderá retornar o tipo de dado de uma variável.

Portanto, você pode perceber que ela é bem parecida com a função já existente gettype(). Porém, a get_debug_type() representa uma melhoria para o PHP, pois ela consegue retornar a verificação de tipo.

Você pode ver na própria RFC as principais diferenças de retornos entre as funções get_debug_type() e gettype()

https://www.homehost.com.br/blog/tutoriais/php/php-8/
https://www.yogh.com.br/blog/php-8-confira-as-novidades-desta-nova-versao/

Performance

Compilador JIT (Just in Time)

De longe, essa é uma das novidades mais esperadas para o PHP 8! Isso pois o JIT Compiler (Just in Time) irá proporcionar um aumento de performance para diversas funções! Principalmente quando se tratar de processamento de imagens e operações de Machine Learning.

Resumidamente, o JIT é um compilador que faz parte da extensão Opcache. Com o JIT, alguns Opcodes não precisarão ser interpretados pela ZendVM, pois, essas instruções serão executadas diretamente a nível de CPU. Por isso que você poderá observar um grande ganho de desempenho para algumas instruções.

No vídeo abaixo, você pode ver como o JIT entrega um resultado de desempenho bem mais rápido que o encontrado no PHP 7. Você pode encontrar o vídeo no Youtube, publicado por Zeev Surasky, co-autor da proposta PHP JIT. Na esquerda, ele apresenta o método atualmente utilizado, já na direita, seria a mesma aplicação rodando com o JIT.

Através do benchmark Mandelbrot, você pode ver a demonstração de qualidade do JIT. Nele, você pode ver um desempenho do PHP 8 de mais de 4 vezes superior ao PHP 7.4 (0,011 seg vs 0,046 seg no PHP 7.4).
De uma forma geral, você poderá observar grandes impactos de desempenho em operações a nível de CPU.

Porém, é claro que o JIT também traz algumas desvantagens, e um dos principais é que ele aumenta a probabilidade de novos BUGs surgirem. Ainda assim, caso isso aconteça, já podemos esperar futuras correções nas atualizações posteriores para esses bugs.

No RFC sobre o JIT você pode estar lendo mais sobre esse compilador e os resultados obtidos no PHP 8.

Benchmarks de performance do PHP 8

Se você está se perguntando o quão rápido é o PHP 8, nós temos a resposta. Nós comparamos 20 plataformas/configurações PHP em 7 versões diferentes do PHP (5.6, 7.0, 7.1, 7.2, 7.3, e 8.0).

O PHP 8.0 emergiu como o vencedor na maioria das plataformas que o suportam, incluindo WordPress e Laravel.

https://kinsta.com/pt/blog/php-8/

Avisos de Erros mais elaborados e precisos

Um grande problema para todos desenvolvedor é quando encontramos um erro.

Atualmente, com o PHP, você já pode ver os avisos de erros. Mas, esses avisos ainda são gerados de forma básica. Porém, no PHP 8.0, você já poderá encontrar avisos com explicações mais precisas.

Com certeza esse será um grande diferencial para os programadores, pois economizará muito tempo para solucionar problemas em meio as diversas linhas de código. Além de facilitar muito a vida de quem está aprendendo e iniciando na linguagem PHP.

Mudanças no método Construtor

Se tratando de sintaxe, uma grande mudança que será proveitosa tanto para desenvolvedores iniciantes quanto para quem já utiliza o PHP a mais tempo é as mudanças no método construtor.

Na programação Orientada a Objetos, o método construtor é um dos mais importantes para uma classe. Portanto, o PHP 8 se propõe a simplifica-lo.

Vamos utilizar como exemplo a criação de uma classe “Pessoa”. Dentro dessa classe, armazenaremos informações de nome, idade e altura. Atualmente, no php 7.4, você precisa fazer o código como no exemplo abaixo:

```
class pessoa{
    public string $nome;
    public int $idade;
    public float $altura;

    public function __construct(
        string $nome,
        int $idade,
        float $altura
    ){
        $this->nome= $nome;
        $this->idade= $idade;
        $this->altura= $altura;
    }
}
```

Se você observar bem, perceberá que você precisou repetir o nome dos atributos três vezes. Portanto, isso acaba tornando o código “redundante”. Pensando nisso, no PHP 8, com método contrutor você conseguirá reescrever a mesma classe conforme o exemplo abaixo:

```
class Pessoa{
    public function __construct(
        public string $nome,
        public int $idade,
        public float $altura
    ) {

    }
}
```

Observe que, agora, você poderá escrever a mesma classe porém com bem menos linhas de código. Além disso, torna o código mais simples de se ler e entender. Portanto, as mudanças do método construtor do PHP 8 serão muito interessante para os desenvolvedores.

Porém, é importante que você saiba que essa funcionalidade possuirá algumas exceções. Você apenas conseguirá utilizar esse recurso em métodos construtores para classes não abstratas. Portanto, para entender melhor esse assunto, recomendamos que você leia a RFC sobre a modificação do método construtor.

Compatibilidade do PHP 8 com os padrões atuais de DOM

O PHP 8 receberá uma atualização na sua DOM API. Portanto, serão adcionado algumas interfaces e classes para tornar a API ext/dom compatível com o padrão atual DOM, que está em contante atualização.

Ou seja, agora o PHP 8 estará compatível com as mudanças atuais do padrão DOM. Dessa forma, você conseguirá manipular elementos DOM mais facilmente através do próprio PHP.

Você pode ler mais sobre isso na RFC dessa proposta https://wiki.php.net/rfc/dom_living_standard_api

Instrução condicional match que surgiu no PHP 8

Parecida com o switch mas mais conciso

```
<?php
$idade = 23;
$resultado = match (true) {
    $idade >= 60 => 'idoso',
    $idade >= 25 => 'adulto',
    $idade >= 18 => 'asdolescente',
    default => 'criança',
};

print $resultado;
```

A extensão OPcache

PHP é uma linguagem interpretada. Isto significa que, quando um script PHP é executado, o intérprete analisa, compila e executa o código uma e outra vez a cada solicitação. Isto pode resultar em desperdício de recursos da CPU e tempo adicional.

É aqui que entra a extensão OPcache para jogar:

“OPcache melhora a performance do PHP ao armazenar bytecode de script pré-compilado na memória compartilhada, removendo assim a necessidade do PHP carregar e analisar scripts em cada solicitação”

Com OPcache habilitado, o intérprete PHP passa pelo processo de 4 etapas mencionado acima apenas quando o script é executado pela primeira vez. Como os bytecodes PHP são armazenados em memória compartilhada, eles estão imediatamente disponíveis como representação intermediária de baixo nível e podem ser executados na Zend VM imediatamente.

A partir do PHP 5.5, a extensão Zend OPcache está disponível por padrão, e você pode verificar se você a tem configurada corretamente, simplesmente ligando para phpinfo() a partir de um script em seu servidor ou verificando seu arquivo php.ini (veja as configurações de configuração do OPcache).

Pré-carga

OPcache foi recentemente melhorado com a implementação do pré-carregamento, um novo recurso OPcache adicionado com PHP 7.4. O pré-carregamento fornece uma maneira de armazenar um conjunto específico de scripts na memória OPcache“antes que qualquer código do aplicativo seja executado”. “Ainda assim, ele não traz melhorias tangíveis de performance para aplicativos típicas baseadas na web.

Com o JIT, o PHP dá um passo à frente.

Operador Nullsafe

Este RFC apresenta o operador nullsafe $-> com avaliação completa de curto-circuito.

Na avaliação de curto-circuito, o segundo operador só é avaliado se o primeiro operador não avaliar para null. Se um operador em uma cadeia avalia para null, a execução de toda a cadeia pára e avalia para null.

Considere os seguintes exemplos da RFC:

$foo = $a?->b();

Se $a é nulo, o método b() não é chamado e $foo está configurado para null.

Veja o RFC do operador nullsafe para exemplos adicionais, exceções e escopo futuro.

Parâmetros com nomes

Ao chamar métodos que possuem muitos parâmetros é comum que nos esqueçamos a ordem dos mesmos.

Também é comum termos que especificar um monte de parâmetros com valores padrão para modificar apenas o que queremos.

function setCookie(name: 'cookieExterno', value: 'teste');

Bem mais fácil, não é mesmo? O mesmo pode ser usado em construtores de classe.

Além de facilitar a vida, seu código ficará muito mais fácil de ser lido e entendido.

Os argumentos nomeados fornecem uma nova maneira de passar argumentos para uma função em PHP:

Os argumentos nomeados permitem passar argumentos para uma função com base no nome do parâmetro, ao invés da posição do parâmetro.

Nós podemos passar argumentos nomeados para uma função simplesmente adicionando o nome do parâmetro antes de seu valor:

callFunction(name: $value);

Nós também podemos usar palavras-chave reservadas, como mostrado no exemplo abaixo:

```
callFunction(array: $value);
```

Mas nós não temos permissão para passar um nome de parâmetro dinamicamente. O parâmetro deve ser um identificador, e a seguinte sintaxe não é permitida:

```
callFunction($name: $value);
```

De acordo com Nikita Popov, a autora deste RFC, os argumentos nomeados oferecem várias vantagens.

Em primeiro lugar, argumentos nomeados nos ajudarão a escrever um código mais compreensível porque seu significado é auto documentar. O exemplo abaixo da RFC é autoexplicativo:

```
array_fill(start_index: 0, num: 100, value: 50);
```

Os argumentos indicados são independentes da ordem. Isto significa que não somos forçados a passar argumentos para uma função na mesma ordem que a assinatura da função:

Precisa de uma hospedagem rápida, segura e amigável ao desenvolvedor para seus sites? Kinsta é construído com desenvolvedores WordPress em mente e fornece muitas ferramentas e um poderoso painel de controle.

Confira os nossos planos

```
array_fill(value: 50, num: 100, start_index: 0);
```

Também é possível combinar argumentos nomeados com argumentos posicionais:

```
htmlspecialchars($string, double_encode: false);
```

Outra grande vantagem dos argumentos nomeados é que eles permitem especificar apenas aqueles argumentos que queremos mudar. Nós não temos que especificar os argumentos padrão se não quisermos sobrescrever os valores padrão. O seguinte exemplo da RFC deixa isso claro:

```
htmlspecialchars($string, default, default, false);
// vs
htmlspecialchars($string, double_encode: false);
```

Property promotion

No PHP 7 e anterior era assim:

```
class Humano1
{
   private $nome;
   private $sobreNome;

   public function __construct($n, $s){
      $this->nome = $n;
      $this->sobreNome = $s;
   }

}
```

Agora no PHP 8

```
class Humano2
{

    public function __construct(public $nome, public $sobreNome){ // $nome e $sobreNome
são promovidas a propriedade, por conta do public
      $this->nome = $nome;
      $this->sobreNome = $sobreNome;
   }

}

$h1 = new Humano1('Ribamar', 'Sousa');
$h2 = new Humano2('Elias', 'EF');

echo $h1->nome . ' ' . $sobreNome;
echo $h2->nome . ' ' . $sobreNome;
```

str_contains

str_contains — Determine if a string contains a given substring

Descrição
str_contains(string $haystack, string $needle): bool

Exemplo #1 Using the empty string ''

```
<?php
if (str_contains('abc', '')) {
    echo "Checking the existence of the empty string will always return true";
}
```

O exemplo acima irá imprimir:

Checking the existence of the empty string will always return true

Exemplo #2 Showing case-sensitivity

```
<?php
$string = 'The lazy fox jumped over the fence';

if (str_contains($string, 'lazy')) {
    echo "The string 'lazy' was found in the string\n";
}

if (str_contains($string, 'Lazy')) {
    echo 'The string "Lazy" was found in the string';
} else {
    echo '"Lazy" was not found because the case does not match';
}
```

11 – Algo sobre o framework PHP Laravel

O Laravel é o framework PHP mais popular da atualidade e isso já acontece há alguns anos.

O framework Laravel nos oferece uma estrutura com muitos e bons recursos para a criação de aplicativos, APIs, sites, etc.

O Laravel é, basicamente, sobre equipar e capacitar os desenvolvedores. Seu objetivo é fornecer informações claras, código e recursos simples e bonitos que ajudam os desenvolvedores a aprender, iniciar e desenvolver rapidamente, e escreva um código simples, claro e que dure.

“Desenvolvedores felizes fazem o melhor código” está escrito na documentação. “A felicidade do desenvolvedor, do download ao deploy” foi o slogan não oficial por um tempo.

Onde outros frameworks podem ter como objetivo principal a pureza arquitetônica, ou compatibilidade com os objetivos e valores das equipes de desenvolvimento corporativo, o principal recurso do Laravel, seu foco está em servir o desenvolvedor individual.

O Laravel se concentra em "convenção sobre configuração" - o que significa que, se você estiver disposto para usar os padrões do Laravel, você precisará fazer muito menos trabalho do que com outros frameworks que exigem que você declare todas as suas configurações, mesmo se você estiver usando a recomendada configuração. Os projetos criados no Laravel levam menos tempo do que os criados na maioria dos outros PHP frameworks.

Nos bastidores, o Laravel usa muitos padrões de design e à medida que avança. Desde o início, o Laravel o ajuda a codificar ambiente acoplado. A estrutura se concentra no design de classes para ter uma única responsabilidade, evitando a codificação embutida nos módulos de nível superior. Módulos de nível superior não dependem dos módulos de nível inferior, tornando a codificação uma experiência agradável.

Devido a essa flexibilidade, o Laravel se tornou um dos frameworks mais populares de hoje.

O Laravel pode lidar com seu ciclo de vida de request/response com bastante facilidade. A complexidade dos requests não existe aqui.

Você pode pensar na primeira camada como a autenticação, enquanto o middleware testa se o usuário está autenticado. Se o usuário não estiver autenticado, o usuário é enviado de volta à página de login. Se o usuário efetuar login com sucesso, o usuário deve enfrentar a segunda camada, que pode consistir na validação do token CSRF. o processo continua assim e os casos de uso mais comuns do middleware Laravel que protege seu aplicativo: autenticação, validação de token CSRF, modo de manutenção, e assim por diante. Quando seu aplicativo está no modo de manutenção, uma visualização personalizada é exibida para todos os pedidos.

O autor do Laravel, Taylor Otwell, resume os recursos flexíveis do Laravel da seguinte forma (em uma conferência do PHPWorld em 2014):

• Procure simplicidade
• Configuração mínima

• Sintaxe concisa, memorável e expressiva
• Infraestrutura poderosa para PHP moderno
• Ótimo para iniciantes e desenvolvedores avançados
• Abraça a comunidade PHP

A versão atual do laravel é a 9

Início dos frameworks, seguindo o Rails

CakePHP foi o primeiro em 2005, foi seguido pelo Symfony, CodeIgniter, Zend Framework e Kohana (um fork do CodeIgniter). Yii em 2008 e Aura e Slim em
2010 . 2011 vem o FuelPHP e Laravel

Alguns dos principais recursos atuais:

- Instalação facilitada
- Configurações simples no .env
- namespace com psr-4
- Convenções sobre configurações
- Router
- MVC
- ORM Eloquente
- Validação de dados
- Tamplates com blade para as views
- Biblioteca para a criação de formulários
- Artisan
- Tinker
- Migrations
- Seeds
- Service Providers
- Middlewares
- Traits
- Tratamento de erros
- Autenticação
- Autorização
- Entre outros

Exemplo usando as convenções: produtos

- Router - routes/web.php
- Model - app/Product.php
- Controller - app/Http/Controllers/ProductController.php
- View - resources/views/products

Métodos/actions padrões de CRUD em controller:

- index - listagem dos registros
- create - form para adicionar novo registro. Trabalha em conjunto com store para adicionar ao banco
- store - armazena no banco novos registros
- show - mostra um registro na tela
- edit - form para edição de registro. Trabalha em conjunto com update, que armazena no banco
- update - armazena no banco alteração de registro
- destroy - excluir registro do banco

MVC

O Laravel usa o padrão de arquitetura MVC

Pastas do MVC do Laravel

- Rotas - /routes
- Model - /app/Models
- Controller - /app/Http/Controllers
- Views - /resources/views

Um controller é responsável por mapear as ações do usuário final para a resposta do aplicativo, enquanto as ações de um modelo incluem ativar processos de negócios ou alterar o estado do modelo. As interações do usuário e a resposta do modelo decidem como um controlador responderá selecionando uma view apropriada.

Fluxo resumido de código de uma aplicação com Laravel

- Um usuário através do Navegador acessa uma rota. Exemplo: clients
- A rota (routes/web.php) geralmente chama um action de um controller. No caso controller clients e action index()
- O controller/action carrega o model respectivo
- Então o model efetua a consulta ao banco de dados
- O banco de dados devolve o resultado para o model
- O model devolve para o controller/action
- O controller chama a view respectiva, passando para ela as informações recebidas do model
- A view mostra para o usuário as informações solicitadas na tela do aplicativo

Dicas
http://artesaos.github.io/laravel-br-awesome/

Alguns colegas dizem trabalhar com Laravel mesmo sem ter estudado PHP, mas me parece que não é a forma ideal. Se você começar com Laravel sem estudar PHP será

apenas um usuário do aplicativo que não entenderá quase nada e apenas seguirá algumas receitas de bolo.

Para ser um programador Laravel, daqueles que cria pacotes e customiza o framework, obrigatoriamente precisa entender PHP, e também conhecer PHPOO e MVC, sem contar que aprender PHP é uma vantagem extra na carreita.

Mãos a obra

Para usar o Laravel com grande facilidade, não se acostume, use este pacote:

https://github.com/ribafs/crud-generator-laravel-br

12 – Apêndices

A – Repositório

Criei um repositório no github para este licro, onde hospedei algum material de interesse, inclusive o aplicativo simplest-mvc e alguns tutoriais.

https://github.com/ribafs/phpoo-livro

B – Recomendações para maior eficiência

Programar não é tarefa fácil

Aprender uma linguagem de programação ou um framework não é tarefa simples. Tanto que existe muito conhecimento a ser assimilado para que se torne um programador profissional e é algo complexo. Veja o que se tem para aprender num framework como o laravel: terminal, composer, PHP, PHPOO, MVC (Controllers, Models, Views, Rotas), Middleware, Service Provider, Helpers, Commands, Migrations, Seeders, Factories, Packages e bem mais. Além da quantidade de conhecimento a absorver, temos a complexidade dos mesmos. Precisa saber disso para não começar enganado e se frustrar. Precisa ser daqueles que se sentem desafiados com problemas, com dificuldades e que consegue lidar com coisas complexas sem se intimidar. Digo isso não para assustar, não para desestimular você de aprender a programar. Não, digo para alertar, pois vejo alguns professores querendo dizer que tudo é tão fácil, quando não é. Se fosse fácil assim, todos saberiam ou aprenderiam com facilidade. Basta ver que o emrcado no mundo inteiro é carente de bons programadores. Porque isso acontece se os grupos de programação estão cheios? Muitos são atraídos, mas poucos em as habilidades suficientes para vencer os obstáculos de forma adequada. Então reflita antes de começar. Precisa entrar consciente do que vai enfrentar e então entrar pra valer. Quer realmente aprender programação? Ou melhor, está realmente motivado e disposto a pagar o preço? Não em dinheiro, mas em empenho, dedicação, estudo, trabalho, …

Raciocínio Lógico

Algo importante para ser um bom programador é ter um raciocínio claro e com lógica, pois tudo num computador é lógico. Esqueça os devaneios, as imaginações fantasiosas e por aí a fora. Precisa por os "pés no chão", ser realista com as coisas e com o mundo, ser racional, pensar com lógica e ter um raciocínio dedutivo tipo o de um detetive, que sai rastreando pistas até encontrar o culpado. Na programação, quando nos deparamos com um bug que não conhecemos, precisamos ser racionais para começar a procurar no lugar mais provável de ele estar. Se não estiver então no segundo mais provável e assim até encontrar. Sugiro que sempre brigue com os bugs por um bom tempo para somente então procurar ajuda no Google, grupos, etc. Assim vamos desenvolvendo nossos músculos do cérebro que precisam crescer. Se na primeira dificuldade for procurar ajuda não se desenvolverá e ficará dependente de ajuda. O ideal é que consiga resolver tudo "sozinho", mas sem extremismos.

Estudar muito é a receita para um bom profissional

- Tutoriais

- Livros
- Vídeos
- Treinamentos
- Cursos
- Grupos do Facebook
- Pesquisas no Google
- etc

Mas o que tornará o programador alguém que de fato tem segurança no que faz é a prática. Praticar muito, corrigir bugs, customizar, criar novos projetos e fazer isso muitas, muitas, muitas vezes.

Experimentar muito

- Estude exemplos de terceiros. Teste e guarde os que estiverem funcionando
- Pegue exemplos e os customize
- Seja organizado e guarde dicas e tutoriais testados de forma organizada, por assunto, para que quando precisar encontrar com facilidade

Praticar

É impossível aprender qualquer linguagem de programação sem escrever código. Não gaste todo o seu tempo lendo livros ou assistindo vídeos; Comece a codificar o mais rápido possível. Na verdade, depois de terminar cada lição ou seção, experimente imediatamente na prática.

Em geral, eu aconselho você a tentar criar um projeto de teste durante o aprendizado. Isso torna o processo não apenas mais realista, mas mais motivador. Sem um objetivo e um resultado adequados em mente, é emocionalmente difícil continuar aprendendo a teoria.

Ao escrever os códigos e implementar suas ideias, você vai se deparar com obstáculos e você terá perguntas a fazer, dúvidas a serem esclarecidas, e ao ter problemas e superá-los, você evolui como desenvolvedor e aumenta seu conhecimento.

Dicas de pesquisa no Google

O Google é uma ferramenta de busca espetacular e ajuda conhecer suas dicas

- Procure por "Laravel", excluindo o termo "iniciantes", mas inclua apenas os resultados do site "Laracasts":
Laravel -beginners site: laracasts.com
- Para pesquisar pela frase exata use aspas

"Meu site preferido de laravel". Somente retornará links que contenham exatamente a frase "Meu site preferido de laravel". Se você pesquisar

Meu site preferido de laravel (sem as aspas). Retornará qualquer site que contenha qualquer uma das palavras e claro que a quantidade de retorno será imensa, dificultando sua pesquisa.

Pesquisar no Google com eficiência trará muita informação relevante e com resultados muito mais rápidos do que pesquisar em fóruns e grupos.

- Tenha cuidado nas pequisas. A versão do laravel é muito importante. Exemplificando: O que funciona no 5.1 geralmente não funcionará no 8. Então nas pesquisas use a versão que procure e veja a data do tutorial/mensagem para comparar com a data de lançamento da versão do laravel. Veja aqui https://pt.wikipedia.org/wiki/Laravel. Também fique atento para as datas de pacotes do GitHub.

Cuidado com versões antigas

Como o PHP muda com frequência em novas versões, se você está iniciando ainda, quando pesquisar por projetos para estudar e testar, é prudente evitar projetos bem antigos. Verifique a data do tutorial do projeto, etc. O Github mostra a data dos arquivos e pastas. Caso baixe um projeto antido e esteja usando uma das últimas versões do PHP encontrará alguns erros por conta da desatualização e podem ser trabalhosos de corrigir. Por isso prefica tutoriais e projetos bem atualizados.

Como se comportar nas comunidades: listas, forums e grupos

Se você é educado, paciente e respeitoso, você tem melhores chances de receber ajuda. E nunca esqueça de colaborar com a comunidade. Aachou uma solução, poste. Sabe como ajudar a outra pessoa responda as dúvidas dela. Quando perguntar algo e encotnrar a reposta, edite sua pergunta e acima adicione [RESOLVIDO] e anote a solução encontrada, ajudando outros. Assim você exercita os seus conhecimentos e fortalece a comunidade. Quando uma mensagem em que parcicipa de um grupo no Facebook receber muitas respostas e não interessar mais, vá até a mensagem, clica nos 3 pontos e Desative notificações desta publicação. Pois é muito importante focar no objetivo e isolar distrações.

Aprender inglês

Para tornar o aprendizado menos difícil você precisa aprender inglês, pelo menos a ler e depois a escrever (opcional). Enquanto não domina acostume-se a usar o Translator do Google para ajudá-lo: https://translate.google.com.br/?hl=pt-BR&tab=wT. O inglês tanto o ajudará nos seus estudos quanto o valorizará como profissional.

Caso pretenda trabalhar no exterior então precisará do inglês fluente.

Lidando com problemas da forma correta

Como mencionado na dica anterior, ao se deparar com um problema que você não pode resolver, é fácil desistir da aprendizagem. Talvez essa seja a razão pela qual muitos começam a aprender, mas apenas alguns se tornam desenvolvedores profissionais. Nem tudo são unicórnios, arco-íris, borboletas e potes de ouro; O caminho estará cheio de problemas, perguntas difíceis, descrença. Eu já estive aí, confie em mim. O principal é continuar, não importa o que. Seja criativo; Sempre há uma saída, mas claro que isso é relativo e somente é verdade se você realmente quer ser programador, se realmente sente afinidade, tesão por programação.

Espero que você encontre sua melhor maneira de aprender e potencialmente criar projetos Laravel incríveis!

Não tenha medo de errar. Em geral a nossa cultura nos ensina a fugir de problemas. Mas problemas precisam ser enfrentados para então serem resolvidos. Com programação nos deparamos com erros praticamente todo dia e se quisermos ser um programador profissional e bom precisamos aprender a lidar com erros de uma forma proativa e se

sentindo desafiado ao invés de intimidado. Lembre que será mais um conhecimento na sua lista. Vá em frente e com o tempo começará a identificar com certa facilidade os erros. Se você já viu este e o resolveu, quando ele aparecer novamente você já sabe o que fazer.

Referências
Boa parte das recomendações acima colhi daqui:

https://www.linkedin.com/pulse/dicas-para-aprender-laravel-mais-rapidamente-rafael-gitahy-machado/?originalSubdomain=pt

Listas impressionantes

Vale a penas gastar algum tempo lendo as listas de links deste repositório para se inspirar, estudar e absorver conhecimento.

https://github.com/ribafs/awesomes

13 - Referências sobre PHPOO

https://www.php.net/manual/pt_BR/language.oop5.php

https://www.ecodezone.com/php-pdo-oop-crud-class-bootstrap/

https://github.com/ribafs/auto-crud

https://www.kodingmadesimple.com/2017/01/simple-ajax-pagination-in-jquery-php-pdo-mysql.html

https://www.youtube.com/watch?v=hzy_P_H-1CQ&list=PLwXQLZ3FdTVEau55kNj_zLgpXL4JZUg8I

Como programar sem orientação a objetos
https://medium.com/@brianwill/how-to-program-without-oop-74a46e0e47a3#.squpnjz4n

https://www.startutorial.com/homes/oo_beginner

https://www.phptutorial.net/php-oop/

https://phpenthusiast.com/object-oriented-php-tutorials/practice

https://code.tutsplus.com/pt/tutorials/object-oriented-php-for-beginners--net-12762

https://www.php.net/manual/pt_BR/language.control-structures.php

https://www.php.net/manual/pt_BR/language.oop5.php

https://www.w3schools.com/php/php_oop_what_is.asp

https://www.tutorialspoint.com/php/php_object_oriented.htm

https://www.killerphp.com/tutorials/php-objects-page-1/

https://tutorials.supunkavinda.blog/php/oop-intro

https://www.w3resource.com/php/classes-objects/php-object-oriented-programming.php

https://www.phptutorial.net/php-oop/

https://medium.com/@leandroembu/testando-relacionamentos-com-o-laravel-tinker-cf1fe028608f

https://www.javatpoint.com/laravel-tinker

https://www.educba.com/laravel-tinker/

https://therichpost.com/php-artisan-tinker-database-commands-which-makes-life-easy/

https://www.tech-prastish.com/blog/how-to-use-laravel-tinker/

https://www.phptutorial.net/php-oop/php-try-catch/

Conclusão

Existe uma certa resistência de alguns colegas que começaram a programar em PHP de forma estruturada. O PHP oferece muita liberdade e podemos trabalhar da forma que desejamos. Já o paradigma de orientação a objetos tem muitos conceitos que precisamos entender e usar para conseguir programar com o paradigma. Mas o paradigma OO é a evolução da programação, tanto que a maioria das linguagens modernas o adotam ou o usam como único paradigma. Assim como praticamente todos os grandes frameworks o adotam, como também o MVC.

Além de tudo, quando aprendemos a programar de forma orientada a objetos nos valorizamos para o mercado de trabalho que está a cada dia mais exigente.

Procure aprender e veja se vale a pena ou não. Não pode dizer que não é bom sem conhecer.

Lembre que existe muito mais sobre o paradigma de POO do que foi mostrado aqui. Faço votos de que tenha sido claro nas explicações e exemplos e também que o tenha estimulado ao ponto de perceber a importância da programação orientada a objetos, que especialmente permite o reuso do código com mais eficiência e ao uso do padrão de arquitetura MVC, para organiza o código em camadas, também facilitando a manutenção.

O paradigma de orientação a objetos permite maior organização do código, especialmente quando combinado com o MVC. Também permite uma maior segurança, basta lembrar da visibilidade de propriedades e métodos, onde podemos manter acessível somente dentro de uma classe algumas propriedades e métodos.

Também temos estruturas que ajudam a manter o código da forma que o programador definiu, como é o caso das interfaces, que definem apenas a assinatura dos métodos, que serão implementados nas classes que as implementam.
E a herança, que define que uma classe pai contém código genérico comum e nas classes filhas tem código especializado e herdam o código genérico da classe pai. Isso otimiza o código reaproveitando código já definido na classe pai, sem precisar criar o mesmo código novamente.

E assim temos muitas outras estruturas que tornam o código melhor, sem contar as boas práticas, convenções e padrões de projeto.

Outro detalhe muito importante é que os principais aplicativos atuais usam orientação a objetos e MVC (frameworks, CMS, etc). Quem não conhece tem mais dificuldade para aprender.

Forum de discussão

Lembre que foi criado um fórum de discussão no repositório do livro

https://github.com/ribafs/phpoo-livro/discussions/

Este é um bom lugar para trocar ideias sobre este livro:
- tirar dúvidas
- feedbacks sobre o livro
- sugestões
- críticas construtivas

Felicidades!