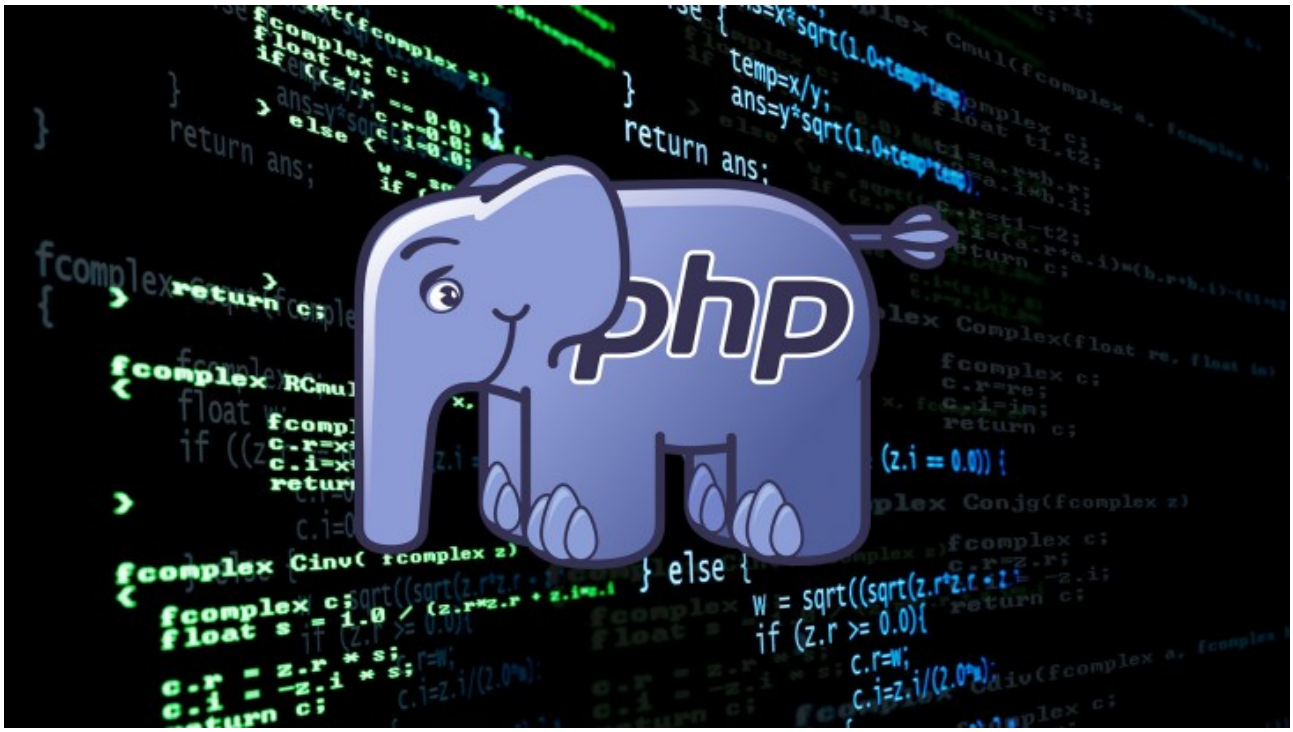
Programando para Backend com PHP

E BootStrap para o Frontend



Ribamar FS
Editora Clube de Autores

Fortaleza, 16 de maio de 2019

Ficha Catalográfica

S725c

Programação para o backend com PHP / Ribamar
Ferreira de Sousa. - Fortaleza: Clube de Autores, 2019

202p.

1. Programação para o backend 2. PHP!. Algo do frontend. I.
Título.

Normalização Bibliográfica

Lúcia Maria Piancó Chaves – DNOCS-CGE/MD
Margarida Lídia de Abreu Vieira – DNOCS-CGE/MD

Apresentação

Agradecimentos

Índice

Uma frase motivacional em cada capítulo

Adicionar ao livro somente conteúdo simples, testado e de preferência exclusivo e meu.

Aprendendo Programação Backend com PHP

Não passar simplesmente a documentação que tenho, mas enxugar, atualizar, testar e otimizar. Ex.: segurança web, não passar tudo e passar somente o que considero que vá acrescentar algo que não tem por aí, algo exclusivo e que seja importante.

- Ferramentas

Backend
- Introdução
- PHP estruturado
  - Regras para nomes:
    - arquivos
    - classes
    - métodos/funções e propriedades
    - constantes
    - bancos, tabelas, campos, etc
  - Sistema de Login
- Bancos de Dados
  - MySQL
    - Introdução
    - Administração
    - MySQL com PHP usando PDO
  - PostgreSQL
    - Introdução
    - Administração
    - PostgreSQL com PHP usando PDO
  - SQlite
    - Introdução
    - Administração
    - SQLite com PHP usando PDO
- PDO - PHP Data Objects
- PHP Orientado a Objetos
  - Novidades do PHP
  - Paginação
  - CRUD
  - Grid Editável
  - PHP com AJAX
  - Aplicativos de exemplo
  - Coding Standards
  - Padrões de Projeto
    - Explicando em detalhes ActiveRecord com exemplo
    - Namespace
    - Autoload com PSR-4
    - Rotas
  - Arquitetura de Software MVC

Segurança
- Introdução
- Otimizações
- Backup e sua importância

Um pouco de História

Versões e lançamentos

PHP 1 - 1995 - Rasmus Lerdorf libera o código fonte para o público
PHP 2 - 1997
PHP 3 - 1998
PHP 4 - 2000
PHP 5 - 2004
PHP 6 - Não existiu
PHP 7 - 2015
PHP 7.1 - 2016
PHP 7.2 - 2017
PHP 7.3 - 2018

Algumas Características do PHP
- open source e free para uso
- multiplataforma: Windows, Linux, Mac OS X, FreeBSD, OpenBSD, etc. Suporta vários servidores web: Apache, Nginx, IIS, etc
- linguagem interpretada pelo servidor web
- suporta os paradigmas: funcional e orientado a objeto
- sintaxe parecida com a do C
- não fortemente tipada (até agora)
- suporta diversos SGBDs: mysql, postgresq, sqlite, ODBC, SQLServer e Oracle
- conta com diversas bibliotecas nativas: filesystem, imagens, strings, arrays, pdf, xml, segurança, sessões, cookies, uso na linha de comando/CLI, zip, criptografia, data e hora, gráficos, matemática, e-mail, etc.
- suporta menipulação de erros com exceções
- a linguagem mais usada para a web atualmente
- a linguagem web mais fácil de aprender
- uma documentação oficial muito boa (php.net)
- a linguagem que mais se encontra documentação em geral na internet

Sistema de distribuição de software Open Source

O movimento de produção e distribuição de software como free e open source (grátis e de código fonte aberto) é o movimento mais generoso que já vi criado pelo ser humano. Pessoas criam software e compartilham com outros da forma mais generosa possível. Me parece que isso foi a base da grande revolução tecnológica que vivemos.

Então veio o protocolo TCP/IP, também aberto e não proprietário, que permite a comunicação como temos ainda hoje.

Então a IBM criou o computador pessoal, padrão IBM PC e abriu suas especificações para que qualquer fabricante pudesse fabricar acessórios para o mesmo e também montá-los e logo isso se tornou padrão mundial até hoje. Antes eu tinha um TK-90X, cujo BASIC não conversava com o do

clega, que tinha um MSX. Mas ao redor do mundo todos os computadores pessoais passaram a usar os mesmos softwares e logos todos tinham a mesma cultura.

Então vieram os celulares. Quando o Google criou o sistema operacional Android e abriu para que qualquer fabricante de smartphone pudesse usá-lo. Graças a isso hoje o Android é o mais usado do mundo e se pode ter um celular com bons recursos e por um preço menor.

Apenas para citar alguns destaques.

Introdução ao 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.

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 <?php e final ?> que permitem que você entre e saia do "modo PHP". Caso o script contenha somente código PHP não devemos usar a tag de fechamento.

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 navegador. O navegador recebe os resultados da execução desse script, mas não sabe qual era o código fonte.

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.

Créditos: https://secure.php.net/manual/pt_BR/index.php

O PHP é uma linguagem interpretada que, assim como o JavaScript, foi desenvolvida para tornar as páginas HTML dinâmicas, com maior interação com o usuário.

A diferença entre o PHP e o JavaScript é que o PHP é uma linguagem server-side, ou seja, necessita de um servidor para ser interpretada. É multiplataforma, podendo ser usada em servidores com diversos sistemas operacionais.

Com o PHP você pode gerar imagens, arquivos PDF e arquivos XML, permitindo que sejam baixados para o seu hd e muito, muito mais.

Uma das maiores vantagens do PHP é seu suporte a diversos banco de dados. Fazer o CRUD (Create, Read, Update and Delete – criar, ler, atualizar e apagar) em banco de dados é extremamente simples com o PHP.

Algumas grandes empresas utilizam o PHP nas suas páginas, como o Facebook por exemplo e a Wikipedia. Os CMS (Content Management System – Sistema de gerenciamento de conteúdo) como

o WordPress, Joomla e Drupal têm seu backend escrito em PHP, assim como vários frameworks, como CakePHP, Laravel, Zend e Synfony.

Que podemos fazer com PHP

Entre as inúmeras coisas que podemos fazer com PHP, destaco:

- Podemos gerar páginas e arquivos dinamicamente
- Podemos criar, abrir, ler, escrever neles e fechar arquivos no servidor.
- Podemos coletar dados de um form web com informações do usuário, email, telefone, etc.
- Podemos enviar e receber emails para usuários de seu website.
- Você pode enviar e receber cookies para rastrear o visitante do seu website.
- Você pode armazenar, excluir e modificar informações em seu banco de dados.
- Você pode restringir acessos não autorizados para seu website.
- Você pode criptografar dados para uma segura transmissão pela internet.
- E fazer muito mais

Vantagens do PHP sobre outras linguagens

Se você está familiarizado com outras linguagens server-side como ASP.NET ou Java, você irá perceber o que torna o PHP especial. Aqui estão algumas vantagns para que escolha o PHP.

- **Fácil de aprender:** PHP é fácil de aprender e usar. Para programadores iniciantes, que justamente iniciaram no desenvolvimento web, PHP é frequentemente considerada como a escolha da linguagem preferida para aprender.
- **Open source:** PHP é um projeto open-source. Ele é desenvolvido e mantido por uma grande comunidade ao redor do mundo que fazem seu código fonte gratuitamente disponível para download e uso.
- **Portabilidade:** PHP roda em várias plataformas como Microsoft Windows, Linux, Mac OS, FreeBSD, OpenBSD, etc. E ele é compatível com todos os principais servidores web usados hoje como Apache, Nginx, IIS, inclusive atualmente conta com um embutido.
- **Rápida Performance:** Scripts escritos em PHP usualmente executam ou rodam mais rápido que os escritos em outras linguagens de script como ASP, Ruby, Python, Java, etc.
- **Vasta Comunidade:** Como PHP é suportado por uma comunidade mundial, encontrar ajuda ou documentação relativa ao PHP online é extremamente fácil, tanto na ótima documentação oficial (php.net), quanto na internet em geral.

Objetivos principais da programação web

- Criação de sites, do tipo blog ou portais. Também temos muitos outros como forums, lojas virtuais, e-learning, etc

- Criação de aplicativos, tipo os de intranet, cadastros, controle de estoque, etc.

- Criação de aplicativos ou jogos mobile (do tipo híbrido, sendo o "frontend" usando as tecnologias web e o "backend" a tecnologia nativa, como Android/Java, WindowsPhoje, etc)

- Praticamente qualquer tipo de programa para outra finalidade. Também pode substituir boa parte dos programas desktop

Para tais finalidades existe atualmente ferramentas que ajudam acriar tais peças de código com grande facilidade e com boa qualidade e recursos, que são os CMSs, os Frameworks e as IDEs especializadas.

Para trabalhar em PHP com grandes aplicativos a IDE mais eficiente que tenho conhecimento é o Eclipse para PHP (http://eclipse.org/pdt), pois foi a que melhor completou o código das bibliotecas de grandes aplicativos.

Podem ser usados com pouco conhecimento e ainda assim nos permitir criar bons sites, aplicativos e até jogos com tais ferramentas.

Mas é muito importante lembrar que quanto mais conhecer as linguagens e ferramentas básicas, como o HTML, CSS, JS, PHP, MySQL entre outras, quanto mais as conhecermos melhor será o resultado do nosso trabalho.

Programação no Backend

Elementos da programação no Backend

- projeto
- documentação
- organização e prioridades
- linguagem do lado server
- framework
- banco de dados
- segurança
- servidor de hospedagem/nuvem
- versionamento/git
- composer/packagist/github
- conhecimento de frontend

Como funcionam os sistemas client-serve com PHP

Cliente
Navegador ou aplicativo do celular num computador/celular do usuário
Chamado Frontend

Servidor
Programa em PHP (com Apache/Nginx ou outro e MySQL no computador servidor)
Chamado BackEnd

Tipos de arquivos solicitados pelo navegador:

Arquivos diferentes de PHP

Quando o cliente/navegador solicita um arquivo do tipo diferente de php (html, htm, txt, imagem qualquer, etc) do Apache, o Apache não processa nada ele devolve o arquivo como está para o navegador que o solicitou. Exemplo:

https://ribafs.org/portal/images/csuperior.png

Esta imagem chegará ao navegador que a slolicitou diretamente do Apache ou de outro servidor web.

Arquivos em PHP (terminados em .php)

Quando o cliente solicita um arquivo .php então o Apache processará todo o seu conteúdo e devolverá somente o resultado em html, css e javascript.

Algumas vezes solicitamos uma url que não mostra o arquivo, neste caso será aberto o index default do servidor, que nos casos de servidores com apache é o index.php.

Se chamar http://ribafs.org, veja o que ele processará:

http://ribafs.org/portal/

Ainda assim o arquivo não aparece, mas ele está processando um index.php, quer ver?

http://ribafs.org/portal/index.php

= Caso o arquivo em php contenha algo de uma mibblioteca interna, como é o caso de processamento de bancos de dados, do sistema de arquivos, ou outro, então o Apache procurará pela respectiva biblioteca e passará o processamento para ela. Quando receber o resultado, então converterá e enviará para o navegador. Uma biblioteca assim muito usada é a PDO que trabalha com bancos de dados.

Visualizando o Código Fonte de Sites

Cada vez que abrimos um site da web ou mesmo em nosso servidor local, ou em HTML estático (sem servidor) podemos visualiar o código fonte do site que estamos vendo com a combinação de teclas:

Ctrl+U

Mas lembre de que se a página for resuultante de arquivo php, não verá nada de PHP aqui. Todo o código php é processado e chega somente o resultado no cliente, em html, css e js. Nunca o Ctrl+U mostra php. Experimente com alguns sites localmente ou em servidor.

Iniciando na Programação com PHP

Para poder programar com PHP precisamos de um servidor web. Diferente de HTML, CSS, JavaScript, XML, jQuery e Bootstrap, pois nenhum destes precisa de servidor web. Para estes podemos criar um arquivo em qualquer diretório e ele funciona.

É importante saber que atualmente o PHP deva ser a linguagem mais popular para programar para a web. Acredito que a facilidade de programar, os recursos oferecidos, a documentação farta em seu site (http://php.net), a grande quantidade de boas ferramentas desenvolvidas com ele (Frameworks, CMS) e a grande aceitação da comunidade foram responsáveis pela popularidade.

Vamos mostrar na tela apenas uma frase 'Olá Mundo':

```
<?php
print “Olá Mundo!”;
?>
```

Se testar isso com C, C++ ou Java vai perceber a simplicidade do PHP.

Tags:

```
<?php
//
?>
```

comentários
final de linha ;

variáveis (caixas, endereços de memória contendo valores)

escopo de variáveis

constantes
ROOT_LOCATION = '/root';

Operadores
https://www.php.net/manual/pt_BR/language.operators.precedence.php

=, ==, ===, !=, !==

$a e !$a (boleano NOT)

>, <, <=, >=,

&, &&, |, ||

Operadores Lógicos	Exemplo	Nome	Resultado
$a and $b	E		Verdadeiro (TRUE) se tanto $a quanto $b são verdadeiros.
$a or $b	OU		Verdadeiro se $a ou $b são verdadeiros.
$a xor $b	XOR		Verdadeiro se $a ou $b são verdadeiros, mas não ambos.
! $a	NÃO		Verdadeiro se $a não é verdadeiro.

$a && $b E Verdadeiro se tanto $a quanto $b são verdadeiros.
$a || $b OU Verdadeiro se $a ou $b são verdadeiros.

Representações de FALSE e TRUE
- TRUE
- 1
-

FALSE

- o próprio booleano FALSE
- o inteiro 0 (zero)
- o ponto flutuante 0.0 (zero)
- uma string vazia e a string "0"
- um array sem elementos
- um objeto sem variáveis membros (somente PHP 4)
- o tipo especial NULL (incluindo variáveis não definidas)
- o objeto SimpleXML criado de tags vazias

Qualquer outro valor é considerado TRUE (incluindo qualquer resource).

Testar
NULL
!NULL

TRUE, FALSE
EMPTY, NULL,
object, resource, array

Tipos de dados
Booleanos
Inteiros
Números de ponto flutuante
Strings
Arrays
Objetos
Recursos
NULL
Callbacks / Callables

Combinar null com outros (Ver postgresql)

incremento
decremento
pré e pós
$a = 4;
$b = 3;
++$a (primeiro incremente e depois atribua)
print $a; // 5
$b++ primeiro atribua e depois incremente

print $b; // 3
precedência de operadores

strings

Aspas
" e '

echo e print

Em múltiplas linhas
echo "Developers, developers, developers, developers, developers,
developers, developers, developers, developers!

- $author.";

HEREDOC
echo <<<_END
Debugging is twice as hard as writing the code in the first place.
Therefore, if you write the code as cleverly as possible, you are,
by definition, not smart enough to debug it.
- $author.
_END;

arrays

estruturas de controle

funções

includes

expressões

Orientação a Ojetos

Constantes
const MINHA_CONST = 'valor';

Para chamar

function teste(){
    echo self::MINHA_CONST;
}
Para ver como isso funciona precisamos criar um arquivo no diretório web e chamar este arquivo no navegador via localhost. Nosso diretório web é c:\xampp\htdocs (no windows). No Linux é algo como /var/www/html

Vamos criar neste diretório o arquivo ola.php, contendo:

```
<?php
print “Olá Mundo!”;
?>
```

E chamar no navegador com:

http://localhost/ola.php

Então veremos a nossa frase Olá Mundo! No navegador.

PHP com HTML

Vale lembrar que o PHP sempre fica do lado do servidor. Ele processa informações solicitadas pelo visitante e as devolve. O servidor web processa todo o código PHP e devolve tudo no formato HTML, juntamente com CSS e JavaScript, que ficam do lado do cliente.

Qualquer arquivo que contenha código PHP precisa ter a extensão .php. Sendo assim, o interpretador do PHP não olhará para outros tipos de arquivos. E quando verificar arquivos .php somente processará os trechos entre <?php e ?>, o que estiver fora destas TAGs ele não olha, simplesmente devolve ao navegador.

O navegador interpreta todo código em HTML pelas tags antes de mostrar na tela.

Separador de Instruções

Observe que ao final de cada linha temos um ponto e vírgula:

print "Como vai?";

Instrução – é um comando único que damos ao PHP, no caso aqui o “print”.

Exemplo: print “Olá”;

O PHP permite escrever mais de uma instrução por linha, mas devemos evitar, por conta de que fica obscuro o código.

Comentários

Em PHP existem três formas de se escrever comentários:

// - este é comentário oriundo do C++ e para apenas uma linha

/*

Este é o comentário oriundo do C

e para

múltiplas linhas

*/

- Este é o comentário oriundo do shell, para uma única linha e que deve ser evitado, pois está em processo de obsolescência, então deve ser evitado.

Estruturas de Controle

Vejamos um pequeno trecho de código mais útil e interessante. Suponha que seu colégio tenha suas 4 notas dos 4 bimestres e precise fazer o cálculo para saber se você passou. Ele tem uma fórmula que é somar as 4 notas e dividir tudo por 4 para tirar a média. Veja:

Vamos criar um arquivo notas.php com o conteúdo abaixo ou testar diretamente no php-console, sem as tags inicial e final.

IF - ELSE

<?php

```
$aluno = '<b>Elias EF</b>';

$nota1 = 10;
$nota2 = 9;
$nota3 = 7;
$nota4 = 5;

$resultado = ($nota1 + $nota2 + $nota3 + $nota4) / 4;

if( $resultado >= 7 ){
    print "O aluno $aluno foi aprovado, com nota $resultado";
```

```
}else{
    print 'O aluno não passou!';
}
?>
```

Variáveis
Como o nome sugere seu valor varia. Vejamos.

Uma variável tem:

nome - $aluno, que é o nome precedido de $.

valor - 'Elias EF'

tipo – string, que é delimitado por " ou "" (acontece que o PHP não declara tipos de variáveis)

Tipos de Dados

O tipo de dados de uma variável será reconhecido quando o código for executado:

string, inteiro, float/double, boleano, array, objeto e resource.

Nomes de Variáveis

- O nome das variáveis precisa começar com uma letra ou pelo símbolo de sublinhado
- O nome de variáveis pode conter somente caracteres alfa-numéricos e sublinhado: a-z, A-Z, 0-9 e _ .
- Não pode conter espaços. Quando várias palavras podemos usar o sublinhado para separá-las: $minha_variavel.

As variáveis podem ter seu valor alterado durante a execução do script, daí seu nome.

Atribuição

Se eu quiser guardar a primeira nota na variável $nota1, então preciso atribuir 10 para $nota1, assim:

$nota1 = 10;

Veja que usamos o sinal de igual para atribuir e terminamos com ponto e vírgula.

Posso atribuir um valor diretamente para uma variável, mas veja que também posso atribuir uma fórmula bem grande para a variável, assim:

$resultado = ($nota1 + $nota2 + $nota3 + $nota4) / 4;
Parecido com uma calculadora o PHP fará os cálculos e guardará o valor na variável $resultado.

Na próxima linha ele vai testar o valor do $resultado. (if($resultado >= 7){)

Se for maior ou igual a 7 ele mostrará a primeira mensagem.

Caso contrário então mostrará a segunda mensagem. (}else{)

Vejamos mais um exemplo prático e interessante de uso do PHP

IF – ELSE IF – ELSE

```
if (expressao){
    executado de true e encessa o teste.
}else if (expressao2){
    executado se true e encessa o teste.
}else if (expressao3){
    executado se true e encessa o teste.
}else if (expressaoN){
    executado se true e encessa o teste.
}else{
    executado se false
}
```

Lembrando que somente entra no próximo teste, se o anterior for falso.

```
$hora = 14;
if($hora <= 12){
print “Está de manhã”;
}else if ($hora > 12 && $hora < 18){
print “Está de tarde”;
}else{
print “Está de noite”;
}
```

if - else

O **if** testa uma expressão

Caso a expressão seja verdadeira, todo o código após o **if** será executado.

Caso não seja verdadeira o código após o **else** será executado.

if – eles if – else

O **if** testa uma expressão

Caso a expressão seja verdadeira, todo o código após o **if** será executado.

Caso a expressão seja falsa então o else if testará uma nova expressão

Caso a nova expressão seja verdadeira o código após o else if será executado
Caso a nova expressão seja falsa, o código após o else ou o novo else if será

executado

Criando Strings

Strings pequenas, com apenas uma linha podem ser mostradas com print ou echo, mas textos com várias linhas podem ser armazenados com o recurso chamado Heredoc:

Armazenando o texto numa variável

```
$str = <<<EOD
Exemplo de uma string
distribuída em várias linhas
utilizando a sintaxe heredoc.
EOD;
```

Ou simplesmente mostrando

```
echo <<<show
<a href="index.php">Início</a>
<h3>Edite Seus Dados</h3>
<form action="update.php" method="post">
<table width="500" border="1">
<tr>
<th scope="row">Nome</th>
<td><input type="text" name="nome" value="$nome"></td>
</tr>
<tr>
<th scope="row">Nascimento</th>
<td><input type="text" name="data_nasc" value=$data_nasc></td>
</tr>
<tr>
<th scope="row">E-mail</th>
<td><input type="text" name="email" value="$email"></td>
</tr>
</table>
</form>
</div>
show;
```

Mais uma estrutura interessante do PHP - Switch

switch – Este testa uma variável contra alguns valores. Parece com um bloco de if else if. Quando encontra o valor correto executa o código seguinte e sai da iteração.

```
$opcao = 3;
switch($opcao)
{
case 1:
      print 1;
      break;
case 2:
      print 2;
      break;
case 3:
      print 3;
      break;
case 4:
```

```
        print 4;
        break;
case 5:
        print 5;
        break;
default:
        print 'Nenhuma das anteriores';
        break;
}
```

Obs.: Caso removamos um dos comandos break o código continua no próximo case.

Laços: for, while, forwach

Estruturas que resolvem vários problemas com pouco esforço.

for

```
for($x = 1;$x <= 10;$x++){
        print $x;
}
for($x = 1;$x <= 10;$x++){
        print $x.'<br>';
}
for($x = 0;$x < 10;$x++){
        print $x;
}
```

Existem variações de uso do for. No caso abaixo, como não temos teste do valor, o incremento continua indefinidamente:

```
for($x=0; ;$x++){
        print $x;
}
```

Nestes casos precisamos inserir um teste no corpo do for, assim:

```
for($x=0; ;$x++){
        if($x>10){
                break;
        }
        print $x;
}
```

Continue

Este comando sai do laço e volta na próxima iteração, ou um certo número de comandos para sair do laço. Veja

```
for ($i = 0; $i < 10; ++$i) {
	if ($i < 5){
		continue;
	}
	print "$i<br>";
}
```

Break

Interrompe o laço, entregando o processamento para a linha seguinte ao laço.

```
for ($x = 0; $x < 5; ++$x) {
	if ($x == 2) {
		break;
	}
	print "$x<br>";
}
```

Experimente.

foreach

Laço através de um array.

```
$arr = array(1, 2, 3, 4);
foreach ($arr as $value) {
	$value = $value * 2;
	print $value;
}
// $arr is now array(2, 4, 6, 8)
unset($value); // Limpar a variável, que armazena o valor
```

Colher Chave e Valor

```
$arr = array(1, 2, 3, 4);
foreach ($arr as $chave => $value) {
	$value = $value * 2;
	print 'Chave: '.$chave.' - Valor: '.$value.'<br>';
}
unset($valor); // Limpar a variável, que armazena o valor
```

Existe muito mais recursos, mas me parece que já dá para perceber isso com estes exemplos mostrados.

Qualquer dúvida sobre qualquer comando do PHP vá até o site oficial em http://php.net e faça uma busca pelo comando, que verá a sintaxe e muitos exemplos de uso.

Atribuição

Atribuir um valor para uma variável é entregar um valor para ela guardar.

Exemplo

$variavel = 5;

Estamos acima dizendo que a variavel guarde o valor 5.

Comparação

Comparação é o que fazemos em uma expressão geralmente em if.

Exemplo

if($x==3)

Estou querendo dizer acima: se $x for igual a 3.

Veja que quando queremos atribuir usamos apenas um sinal de igualdade:

$x = 3;

Quando queremos comparar usamos dois sinais de igualdade: if($x == 3)

Com = armazenamos um valor em uma variável.

Com == eu testo se uma variável é igual a um valor.

Filtrando dados no PHP

Uma das maiores forças do PHP é a sua facilidade de aprendizado e de uso. Infelizmente este mesmo benefício tem trabalhado contra o PHP quando muitos novatos esquecido de usar algumas medidas de segurança.

Existe no PHP muitas classes de validação. A extensão filter do PHP tem muitas das funções necessárias para checar tipos de entrada de usuários.

Lista de filtros

```
<table>
<tr><td>Nome do filtro</td><td>ID do filtro</td></td>
<?php
    foreach(filter_list() as $id=>$filter){
        echo '<tr><td>'.$filter.'</td><td>'.filter_id($filter).'</td></td>';
    }
?>
</table>
```

Filrando Variáveis

```
<?php

/*** an integer to check ***/
$int = 1234;
/*** validate the integer ***/
echo filter_var($int, FILTER_VALIDATE_INT);

// Retorna 1234, pois este é int. Caso contrário retornaria false

?>
```

Validando Inteiros

O filtro FILTER_VALIDATE_INT também permite especificar uma faixa para validar o inteiro.

```
<?php
/*** an integer to check ***/
$int = 42;
/*** lower limit of the int ***/
$min = 50;
/*** upper limit of the int ***/
$max = 100;

/*** validate the integer ***/
echo filter_var($int, FILTER_VALIDATE_INT, array("options" =>
array("min_range"=>$min, "max_range"=>$max)));
?>
```

```
Retorna false, pois 42 não está entre 50 e 100
Outro exemplo:

<?php
/*** an integer to check ***/
$int = -2;

/*** lower limit of the int ***/
$min = -10;

/*** upper limit of the int ***/
$max = 100;

/*** validate the integer ***/
echo filter_var($int, FILTER_VALIDATE_INT, array("options" =>
array("min_range"=>$min, "max_range"=>$max)));
// Retorna int(-2), pois -2 esta entre -10 e 100
?>

Exemplo com apenas min

<?php
/*** an integer to check ***/
$int = 12;

/*** lower limit of the int ***/
$min = 10;

/*** validate the integer ***/
echo filter_var($int,
FILTER_VALIDATE_INT,array('options'=>array('min_range'=>$min)));
// Retorna 12, pois está na faixa de 10 até...
?>
```

Validando Boolean

```
<?php

/*** test for a boolean value ***/
echo filter_var("true", FILTER_VALIDATE_BOOLEAN);

// Retorna 1, pois true é boolean
?>
```

Valores válidos para true:

- 1
- "1"
- "yes"
- "true"
- "on"

- TRUE

Valores válidos para false:

- 0
- "0"
- "no"
- "false"
- "off"
- ""
- NULL
- FALSE

Validando float

```
<?php

/*** an FLOAT value to check ***/
$float = 22.42;

/*** validate with the FLOAT flag ***/
if(filter_var($float, FILTER_VALIDATE_FLOAT) === false)
    {
    echo "$float is not valid!";
    }
else
    {
    echo "$float is a valid floating point number";
    }

// 22.42 é Float válido
?>
```

Validando RegEx

```
<?php

/*** an email address ***/
$email = "ribafs@gmail.com";

/*** the pattern we wish to match against ***/
$pattern = "/^\S+@[\w\d.-]{2,}\.[\w]{2,6}$/iU";

/*** try to validate with the regex pattern ***/
if(filter_var($email, FILTER_VALIDATE_REGEXP,
array("options"=>array("regexp"=>$pattern))) === false) {
        /*** if there is no match ***/
        echo "Sorry, no match";
        }else{
        /*** if we match the pattern ***/
        echo "The email $email address is valid";
```

```
        }
?>
```

Existem muitas outras validações/filtros

Validação de E-mails

```
<?php

/*** an email address ***/
$email = "kevin@foo.bar.net";

/*** try to validate the email ***/
if(filter_var($email, FILTER_VALIDATE_EMAIL) === FALSE)
        {
    /*** if it fails validation ***/
        echo "$email is invalid";
        }
else
        {
    /*** if the address passes validation ***/
        echo "$email is valid";
        }
?>
```

Saneando Strings

```
<?php

/*** a string with tags ***/
$string = "<script>foo</script>";

/*** sanitize the string ***/
echo filter_var($string, FILTER_SANITIZE_STRING);
// Limpa as tags, deixa somente foo
?>
```

Outro exemplo

```
<?php

/*** a string with tags ***/
$string = "<script>\"'foo'\"</script>";

/*** sanitize the string ***/
echo filter_var($string, FILTER_SANITIZE_STRING);

?>
```

Saneando E-mail

```
<?php

/*** an email address ***/
$email = "kevin&friends@(foo).ex\\ample.com";

/*** sanitize the email address ***/
echo filter_var($email, FILTER_SANITIZE_EMAIL);

?>
```

Saneando Inteiros

```
<?php

/*** an interger ***/
$int = "abc40def+;2";

/*** sanitize the integer ***/
echo filter_var($int, FILTER_SANITIZE_NUMBER_INT);

?>
```

Ver
https://phpro.org/tutorials/Filtering-Data-with-PHP.html
Para mais detalhes

Dicas de PHP

Usando $_POST para testar se o submit foi enviado de um form usando POST

```
if($_POST){

    // set product property values
    $product->name = $_POST['name'];
    $product->price = $_POST['price'];
    $product->description = $_POST['description'];
    $product->category_id = $_POST['category_id'];

    // create the product
    if($product->create()){
        echo "<div class='alert alert-success'>Product was
created.</div>";
    }

    // if unable to create the product, tell the user
    else{
        echo "<div class='alert alert-danger'>Unable to create
product.</div>";
    }
}
```

Expressão condicional

```
// get ID of the product to be edited
$id = isset($_GET['id']) ? $_GET['id'] : die('ERROR: missing
ID.');
```

Passando informações via GET

```
      // get passed parameter value, in this case, the record ID
      // isset() is a PHP function used to verify if a value is
there or not
      $id=isset($_GET['id']) ? $_GET['id'] : die('ERROR: Record ID
not found.');
```

Uso do try catch

```
      include 'config/database.php';

      // read current record's data
      try {
         // prepare select query
         $query = "SELECT id, name, description, price, image FROM
products WHERE id = ? LIMIT 0,1";
```

```
        $stmt = $con->prepare( $query );

        // this is the first question mark
        $stmt->bindParam(1, $id);

        // execute our query
        $stmt->execute();

        // store retrieved row to a variable
        $row = $stmt->fetch(PDO::FETCH_ASSOC);

        // values to fill up our form
        $name = $row['name'];
        $description = $row['description'];
        $price = $row['price'];
        $image = $row['image'];
    }

    // show error
    catch(PDOException $exception){
        die('ERROR: ' . $exception->getMessage());
    }
```

Checar uso do POST

```
    // check if form was submitted
    if($_POST){

        try{

            // write update query
            // in this case, it seemed like we have so many fields
to pass and
            // it is better to label them and not use question
marks
            $query = "UPDATE products
                        SET name=:name, description=:description,
price=:price
                        WHERE id = :id";

            // prepare query for excecution
            $stmt = $con->prepare($query);

            // posted values
            $name=htmlspecialchars(strip_tags($_POST['name']));


$description=htmlspecialchars(strip_tags($_POST['description']));
            $price=htmlspecialchars(strip_tags($_POST['price']));
```

```
            // bind the parameters
            $stmt->bindParam(':name', $name);
            $stmt->bindParam(':description', $description);
            $stmt->bindParam(':price', $price);
            $stmt->bindParam(':id', $id);

            // Execute the query
            if($stmt->execute()){
                echo "<div class='alert alert-success'>Record was
updated.</div>";
            }else{
                echo "<div class='alert alert-danger'>Unable to
update record. Please try again.</div>";
            }

        }
```

Dicas sobre o php.ini ou ini_set()

```
ini_set('max_execution_time', '-1');// Ilimitados
ini_set('max_input_time', 120);// s
ini_set('max_input_nesting_level', 64);//s
ini_set('memory_limit', '-1');//Ilimitada
```

Expressões Regulares em PHP

Caracteres em Branco e Caracteres de Escape
Assim como no PHP, em expressões regulares nós temos também que nos utilizar de caracteres de escape, a saber: barra invertida (). Segue, então, a lista com os caracteres de espaço em branco e alguns caracteres que necessitam de escape.

1. \t – Caracter de tabulação
2. \n – Nova linha
3. \f – Avanço de página
4. \r – Retorno de carro
5. \. – Qualquer caractere
6. \\ – Uma barra invertida literal
7. \- – Um hífen literal

… e com alguma garimpada pela rede você consegue encontrar uma lista mais completa!

Vamos agora aprender intervalo de caracteres

1. [a-z] - Qualquer letra minúscula
2. [A-Z] - Qualquer letra maiúscula
3. [a-zA-Z] - Qualquer letra maiúscula ou minúscula
4. [0-9] - Qualquer número
5. [0-9.-] - Qualquer número, ponto ou sinal de subtração

Bom… Tudo muito legal, muito simples… Mas isto serve apenas para combinação de UM caractere.

1. ^[a-z][0-9]$

Neste caso teríamos uma expressão de apenas dois caracteres em que o primeiro tem necessariamente que ser uma letra minúscula e o segundo ser um número.

Classes de Caracteres Pré-definidas
Para evitar chateação, foram criadas as classes de caracteres pré-definidas que já vêm junto com o interpretador de ER que você estiver utilizando (lembrando que estão presentes apenas no método POSIX).

1. [[:alpha:]] // Qualquer letra (alfabético)
2. [[:digit:]] // Qualquer número (dígito)
3. [[:alnum:]] // Qualquer letra ou número (alfanumérico)
4. [[:space:]] // Qualquer caractere de espaço
5. [[:upper:]] // Qualquer letra maíuscula
6. [[:lower:]] // Qualquer letra minúscula
7. [[:punct:]] // Qualquer caractere de pontuação
8. [[:xdigit:]] // Qualquer dígito hexadecimal (Equivalente a: [0-9a-fA-F])

[[:blank:]] // Espaço e Tab
[[:cntl:]] // Caractere de controle

Ocorrências Múltiplas

Agora vamos pra parte em que realmente começa a fazer sentido usar-se expressões regulares.

1. ^[[:alpha:]]{3}$ // Qualquer palavra de três letras
2. ^a{4}$ // Só fecha com a expressão 'aaaa'
3. ^a{2,4}$ // Fecha com 'aa', 'aaa' e 'aaaa'
4. ^a{2,}$ // Fecha com 'aa', 'aaa', 'aaaa', 'aaaaa' e assim por diante

Até aqui… Tudo tranquilo? Então vamos em frente!
Há outros modos de representar a repetição de caracteres!

1. ? - Uma ocorrência ou nenhuma (Equivale a {0,1})
2. * - Nenhuma ocorrência, uma ocorrência, duas ocorrências e por aí vai (Equivale a {0,})
3. + - Uma ou mais ocorrências (Equivale a {1,})

Pipe é ou

r|s é equivalente a [rs]

Entendendo as Expressões Regulares por Rafael Jaques

Funções

ereg_replace - Busca e substitui o que casar com a expressão.
eregi_replace - Igual a ereg_replace, porém é case-insensitive.
preg_match_all - Retorna as ocorrências de uma string que casarem com a expressão.
preg_match - Verifica se uma string casa com a expressão.
preg_replace - Busca e substitui, retornando todas as opções.
preg_split - Divide uma string utilizando uma expressão.

Exemplo

```
<?php
$cep = '22710-045';
$names = array('Diogo', 'Renato', 'Gomes', 'Thiago', 'Leonardo');
$text = 'Lorem ipsum dolor sit amet, consectetuer adipiscing.';

// Validação de CEP
$er = '/^(\d){5}-(\d){3}$/';
if(preg_match($er, $cep)) {
    echo "O cep casou com a expressão.";
}
// Resultado: O cep casou com a expressão.

// Busca e substitui nomes que tenham "go", case-insensitive
$er = '/go/i';
$pregReplace = preg_replace($er, 'GO', $names);
print_r($pregReplace);
// Resultado: DioGO, Renato, GOmes, ThiaGO, Leonardo

// Busca e substitui nomes que terminam com "go"
```

```
$er = '/go$/';
$pregFilter = preg_filter($er, 'GO', $names);
print_r($pregFilter);
// Resultado: DioGO, ThiaGO

// Resgatar nomes que começam com "go", case-insensitive
$er = '/^go/i';
$pregGrep = preg_grep($er, $names);
print_r($pregGrep);
// Resultado: Gomes

// Divide o texto por pontos e espaços, que podem ser seguidos por
espaços
$er = '/[[:punct:]\s]\s*/';
$pregSplit = preg_split($er, $text);
print_r($pregSplit);
// Resultado: Array de palavras

// callback, retorna em letras maiúsculas
$callback = function($matches) {
    return strtoupper($matches[0]);
};

// Busca e substitui de acordo com o callback
$er = '/(.*)go$/';
$pregCallback = preg_replace_callback($er, $callback, $names);
print_r($pregCallback);
// Resultado: DIOGO, Renato, Gomes, THIAGO, Leonardo
?>
```

Tratamento de Erros em PHP

```
<?php
// Adicionar ao início do arquivo para exibir mensagens de erro
error_reporting(E_ALL);
ini_set('display_errors', 1);


Não mostrar notices
error_reporting(E_ALL & ~E_NOTICE);


ini_set('error_reporting', E_ALL & ~E_NOTICE & ~E_WARNING & ~E_STRICT);
ini_set('display_errors','On');
```

Adicionar as linhas acima ao configuration.php, logo abaixo de <?php

===
Para não mostrar mensagens de erro adicione ao index.php do raiz logo após o JEXEC:

```
ini_set('error_reporting', E_ALL ^ E_DEPRECATED);
ini_set('display_errors','Off');
```

Criar páginas de erro usando o .htaccess

Adicionar ao .htaccess do raiz do servidor:

```
ErrorDocument 400 /errors.php
ErrorDocument 403 /errors.php
ErrorDocument 404 /errors.php
ErrorDocument 405 /errors.php
ErrorDocument 408 /errors.php
ErrorDocument 500 /errors.php
ErrorDocument 502 /errors.php
ErrorDocument 504 /errors.php
```

Criar o
errors.php

```
<?php
$status = $_SERVER['REDIRECT_STATUS'];
$codes = array(
400 => array('400 Bad Request', 'The request cannot be fulfilled due to bad syntax/Sintaxe errada.'),
403 => array('403 Forbidden', 'The server has refused to fulfil your request. O Servidor recusou sua solicitação'),
404 => array('404 Not Found', 'The page you requested was not found on this server. Página não encontrada'),
405 => array('405 Method Not Allowed', 'The method specified in the request is not allowed for the specified resource/Método da requisição não permitido.'),
```

```
408 => array('408 Request Timeout', 'Your browser failed to send a request in the time allowed by the server/Tempo esgotado'),
500 => array('500 Internal Server Error', 'The request was unsuccessful due to an unexpected condition encountered by the server. Erro interno'),
502 => array('502 Bad Gateway', 'The server received an invalid response while trying to carry out the request. Resposta inválida do servidor'),
504 => array('504 Gateway Timeout', 'The upstream server failed to send a request in the time allowed by the server. Tempo esgorado do servidor'),
);

$title = $codes[$status][0];
$message = $codes[$status][1];
if ($title == false || strlen($status) != 3) {
$message = 'Please supply a valid HTTP status code.';
}

echo '<h1>Ôpa! '.$title.' detectado!</h1>
<p>'.$message.'</p>';

print '<h3>Desculpe. Acesse a index do site</h3>'.
'<a href="/index.php">index.php</a>
Ou contacte via ribafs @ gmail.com.';
```

Principais vantagens da POO

A programação orientada a objetos traz uma ideia muito interessante: a representação de cada elemento em termos de um objeto, ou classe. Esse tipo de representação procura aproximar o sistema que está sendo criado ao que é observado no mundo real, e um objeto contém características e ações, assim como vemos na realidade. Esse tipo de representação traz algumas vantagens muito interessantes para os desenvolvedores e também para o usuário da aplicação. Veremos algumas delas a seguir.

A reutilização de código é um dos principais requisitos no desenvolvimento de software atual. Com a complexidade dos sistemas cada vez maior, o tempo de desenvolvimento iria aumentar exponencialmente caso não fosse possível a reutilização. A orientação a objetos permite que haja uma reutilização do código criado, diminuindo o tempo de desenvolvimento, bem como o número de linhas de código. Isso é possível devido ao fato de que as linguagens de programação orientada a objetos trazem representações muito claras de cada um dos elementos, e esses elementos normalmente não são interdependentes. Essa independência entre as partes do software é o que permite que esse código seja reutilizado em outros sistemas no futuro.

Outra grande vantagem que o desenvolvimento orientado a objetos traz diz respeito a leitura e manutenção de código. Como a representação do sistema se aproxima muito do que vemos na vida real, o entendimento do sistema como um todo e de cada parte individualmente fica muito mais simples. Isso permite que a equipe de desenvolvimento não fique dependente de uma pessoa apenas, como acontecia com frequência em linguagens estruturadas como o C, por exemplo.

A criação de bibliotecas é outro ponto que é muito mais simples com a orientação a objetos. No caso das linguagens estruturadas, como o C, temos que as bibliotecas são coleções de procedimentos (ou funções) que podem ser reutilizadas. No caso da POO, entretanto, as bibliotecas trazem representações de classes, que são muito mais claras para permitirem a reutilização.

Entretanto, nem tudo é perfeição na programação orientada a objetos. A execução de uma aplicação orientada a objetos é mais lenta do que o que vemos na programação estruturada, por exemplo. Isso acontece devido à complexidade do modelo, que traz representações na forma de classes. Essas representações irão fazer com que a execução do programa tenha muitos desvios, diferente da execução sequencial da programação estruturada. Esse é o grande motivo por trás da preferência pela linguagem C em hardware limitado, como sistemas embarcados. Também é o motivo pelo qual a programação para sistemas móveis como o Google Android, embora em Java (linguagem orientada a objetos), seja feita o menos orientada a objetos possível.

No momento atual em que estamos, tecnologicamente essa execução mais lenta não é sentida. Isso significa que, em termos de desenvolvimento de sistemas modernos, a programação orientada a objetos é a mais recomendada devido as vantagens que foram apresentadas. Essas vantagens são derivadas do modelo de programação, que busca uma representação baseada no que vemos no mundo real.
https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264

Neste artigo, quero mostrar algumas diferenças entre a programação estruturada (PE) e a programação orientada a objetos (POO). Logo quando pensamos em criar um programa, temos que analisar qual programação é a mais adequada para um problema em questão. Tanto a programação PE ou a programação OO possuem seus pontos altos e baixos, porém a orientada a objetos tem ganhado a preferência dos desenvolvedores para os novos projetos. Graças a facilidade de

manipular os dados pois trabalhamos com classes, objetos, herança, encapsulamento de regras de negócios, estruturas de dados, etc., onde cada classe tem seu objetivo específico, assim fazer qualquer alteração em seu código se torna muito mais fácil, mais rápido e sem “danos” em classes dependentes da mesma.

A programação estruturada é formada apenas por três estruturas, que são sequência, onde uma tarefa é executada logo após a outra, decisão quando um teste lógico é executado ou não, e iteração que a partir do teste lógico algum trecho do código pode ser repetido finitas vezes. Seus códigos ficam em um mesmo bloco, sendo mais difícil e demorado fazer uma alteração, pois teremos que olhar se nenhum outro código depende daquele, fazendo uma análise mais detalhada. É fácil de entender, sendo usada em cursos introdutórios a programação. No final deste artigo, vemos um exemplo de programação estruturada, um programa bem simples usado somente para exemplo. O mesmo exemplo está disponível para visualização, abaixo do exemplo da PE, onde criamos mais pacotes e classes como boas práticas para programar em OO.

Na imagem da programação estruturada, fica claro quando falamos em declarar variáveis, métodos e funções em uma mesma página. Na orientada a objeto vemos as classes, pacotes, e o que caracteriza uma POO.
Imagine fazer uma alteração em um formulário na programação estruturada, onde teremos que ver o que será afetado, onde teremos que reestruturar e assim olhar código por código e ver se nada foi afetado com a alteração. Na orientada a objetos as rotinas e funções estão em objetos separados, encapsulados, facilitando as alterações e atualizações. Procuramos a classe onde o método esta definido, e assim alterando somente aqueles métodos.

Na fábrica de software nosso foco é Java, e essa linguagem é toda orientada a objetos, pois não existe uma linguagem de PE para Java, onde aprendemos a lidar com classes, métodos, herança, polimorfismo entre outros que caracterizam uma POO. Na minha graduação em Tecnologia em Redes, estamos aprendendo a linguagem C estruturada, onde vemos as vantagens e desvantagens em cima da POO.

Minha opinião é que a melhor forma de programar é a orientada a objeto, pois fica bem mais fácil de manusear os códigos, através da herança podemos usar variáveis e métodos já declaradas em outras classes, fica fácil de dar manutenção no programa, fácil para outros programadores possam entende o raciocínio lógico e também alterar os códigos e em questões de segurança é difícil do código ser copiado graças ao encapsulamento das classes
http://fabrica.ms.senac.br/2013/04/programacao-estruturada-versus-programacao-orientada-a-objetos/

Performance

- Usar bancos de dados somente quando necessário
- Otimizar, minimizar e reduzir consultas ao banco
- Da mesma forma muito cuidado com serviços externos, trate com cuidado estes
- Muito cuidado com os loops, especialmente testar se não é infinito
- Evite função recursiva, função que chama função pode cair em loop infinito
- Mantenha o código simples e organizado, evitando duplicações - KISS
- Tenha um bom controle de erros e trate ao máximo com try/catchs
- Tenha cuidado com todas as entradas de dados de formulários: validação de tamanho, tipo de dados, requerido, etc
- Crie sempre o melhor software que pode
- Use orientação a objetos, bons padrões, MVC
- Modularize, componentize seu aplicativo
- Tentar criar um código o mais flexível possível e evitar engessar alguams partes
- Boa documentação do código com comentários para partes do código que demandem
- Use nomes coerentes para arquivos, tabelas, campos, variáveis, funções e classes
- Lembre de criar cada classe em um arquivo separado e que cada classe seja somente sobre um único assunto, problema

Performance do PHP
https://kinsta.com/pt/blog/benchmarks-definitivos-do-php/