CloudSource

A fantástica fábrica de soluções.

Apostila PHP

Avenida Paraná, 974, 2º andar, Centro, Foz do Iguaçu

(045) 9940-7715 | (045) 9813-0666 | (045) 9818-2568

1. Introdução

1.1. O que é PHP

PHP, um acrônimo recursivo para PHP Hypertext Preprocessor, é uma linguagem de programação script, amplamente utilizada para o desenvolvimento de sites e aplicações web. O PHP é uma linguagem interpretada, com tipagem fraca, de fácil aprendizado mas bastante poderosa.

O PHP, quando usado como linguagem web, é totalmente interpretada do lado servidor, devolvendo para o cliente somente o resultado HTML processado, impossibilitando assim o cliente saber qual é o código fonte de sua aplicação.

Uma das grandes vantagens do PHP está no fato ser ter uma curva de aprendizado baixa, tornando-se uma linguagem ideal para iniciantes, mas também conta com muitos recursos para programadores mais avançados.

1.2. O que o PHP pode fazer?

Basicamente qualquer coisa. O PHP é focado para ser uma linguagem web, portanto capaz de realizar qualquer tarefa que outra linguagem desse tipo se propões a fazer, como: Coletar dados de formulários, ler e gravar informações em banco de dados, gerar páginas com conteúdo dinâmico.

O PHP também pode ser utilizado para criar scripts para linha de comando, rodando na máquina local. E também é possivel criar aplicativos desktop com o PHP-GTK (http://gtk.php.net/).

Se tratando em linguagem para desenvolvimento web o PHP pode ser utilizado na maioria dos sistema operacionais, e também é suportado pela maioria dos servidores web como Apache, Microsoft Internet Information Server, Personal Web Server, Netscape and iPlanet Servers, Oreilly Website Pro Server, Caudium, Xitami, OmniHTTPd.

Talvez uma das maiores vantagens do PHP é o suporte a vários banco de dados e a facilidade para criar um código que faça uma consulta a algum deles, a lista de banco de dados suportada é:

- Adabas D
- dBase
- Empress
- FilePro (apenas para leitura)

- Hyperwave
- IBM DB2
- Informix
- Ingres
- InterBase
- FrontBase
- mSQL
- Direct MS-SQL
- MySQL
- ODBC
- Oracle (OCI7 e OCI8)
- Ovrimos
- PostgreSQL
- SQLite
- Solid
- Sybase
- Velocis
- Unix dbm

2. Básico

2.1. Sintaxe

O interpretador do PHP irá processar tudo que estiver dentro das tags de abertura (**<?php**) e fechamento (**?>**). Dessa maneira o PHP pode ser embutido em praticamente qualquer arquivo já que tudo que estiver fora dessas tags será ignorado pelo interpretador.

Geralmente podemos encontrar código PHP dentro de arquivos HTML, apesar de não ser a melhor forma de se faze-lo, como no exemplo abaixo.

```
<p>Isto vai ser ignorado.</p>
<?php echo 'Enquanto isto vai ser interpretado.'; ?>
<p>Isto também vai ser ignorado.</p>
```

Todos os comandos em PHP devem terminar com **;** (ponto e vírgula), exceto no uso de condicionais e laços, que será visto mais adiante.

2.2. Comentários

Os comentários não são interpretados, é uma forma de descrever qual a funcionalidade de algum trecho de código, função, classe etc. Portanto use sempre para descrever seus códigos de forma clara e consisa.

O PHP suporta vários tipos de comentários, no estilo das linguagens C, C++ e Shell Script conforme o exemplo abaixo:

```php
<?php
echo 'Isto é um teste';//Comentário de uma linha em c++

/* Este é um comentário de múltiplas linhas
       ainda outra linha de comentário */

echo 'Isto é ainda outro teste';
echo 'Um teste final';#Comentário estilo shell 1 linha
?>
```

Com os delimitadores de comentários do tipo **//** e **#** é possível comentar em apenas uma linha. Já para comentar em mais de uma linha geralmente é utilizado o delimitador do tipo **/* */**, sendo o delimitador **/*** usado para iniciar o bloco de comentário e o ***/** para fechar.

2.3. Tipos de dados

O PHP possui tipagem fraca, ou seja, o tipo de uma variável não é definida pelo programador mas sim decidido em tempo de execução.

A linguagem possui oito tipos primitivos de dados, sendo quatro do tipo básico:

- boolean (verdadeiro e falso)
- integer (para númros inteiro)
- float (para números de ponto flutuante)
- string (conjunto de caracteres

Dois do tipo composto:

- array (conjunto de dados)
- object (instância de uma classe)

E dois do tipo especial:

- resource (referencia a algum recurso externo)
- NULL (a variável não possui valor)

2.4. Tipo boolean

É o tipo de dado mais simples no PHP, guarda somente o valor verdadeiro ou falso (**TRUE** / **FALSE**).

```
<?php
$foo = True; // atribui o valor True para $foo
$bar = False; // atribui o valor True para $foo
?>
```

2.5. Tipo integer

O integer é o tipo de dados para armazenar números inteiros. Podem ser especificados em notação decimal (base 10), hexadecimal (base 16) ou octal (base 8).

Para usar a notação octal, você precisa preceder o número com um 0 (zero). Para utilizar a notação hexadecimal, preceda número com 0x.

```
<?php
$a = 1234; // número decimal
$a = -123; // um número negativo
$a = 0123; // número octal (equivalente a 83 em decimal)
$a = 0x1A; // número hexadecimal (equivalente a 26 em de
cimal)
?>
```

O tamanho de um inteiro é dependente de plataforma, sendo um número aproximado a 2 bilhões o valor mais comum (número de 32 bits com sinal). O PHP não suporta inteiros sem sinal.

O tamanho do inteiro pode ser determinado pela constante PHP_INT_SIZE, e seu o val or máximo com a constante PHP_INT_MAX desde o PHP 4.4.0 e PHP 5.0.5.

2.6. Tipo float

Tipo de dado para armazenar números de ponto flutuante (também conhecidos como “floats”, “doubles” ou “números reais”)

Podem ser definidos das seguintes formas:

```php
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

O separador decimal usado para definir um número de ponto flutuante é o ponto (.)

2.7. Tipo string

String é o tipo de dados para representar um conjunto de caracteres. Não há limites para representar esse tipo de dado no PHP, sua única limitação é a quantidade de memória do servidor.

Basicamente para se definir uma váriavel com o tipo de dado string é utilizado apóstofro ou aspas, conforme o exemplo:

```php
<?php
$a = 'exemplo de string';
$b = "outro exemplo de string";
?>
```

2.8. Tipo Array

Arrays é o tipo de dado mais poderoso do PHP, sendo um tipo de dado composto. Um array no PHP é atualmente um mapa ordenado. Um mapa é um tipo que relaciona valores para chaves.

Este tipo é otimizado de várias maneiras, então você pode usá-lo como um array real, ou uma lista (vetor), hashtable (que é uma implementação de mapa), dicionário, coleção, pilha, fila e provavelmente mais.

Como você pode ter outro array PHP como um valor, você pode facilmente simular árvores.

Um array pode ser criado com o construtor de linguagem array(). Ele pega um certo número de pares separados por vírgula chave => valor.

```
array(  chave =>  valor
     , ...
     )
// chave pode ser tanto string ou um integer
// valor pode ser qualquer coisa
```

Exemplo de definição

```php
<?php
$arr = array("foo" => "bar", 12 => true);

echo $arr["foo"]; // bar
echo $arr[12];    // 1
?>
```

Também é possível criar arrays dentro de um array

```php
<?php
$arr = array("chave" => array(6 => 5, 13 => 9, "a" => 42
));

echo $arr["chave"][6];    // 5
echo $arr["chave"][13];   // 9
echo $arr["chave"]["a"];  // 42
?>
```

Se omitir a chave quando fornece um novo item, o maior índice inteiro é obtido, e a nova chave será esse máximo + 1. Se você especificar uma chave que já possui um valor assimilada a ela, então o valor é sobrescrito.

```php
<?php
// Esse array é como ...
array(5 => 43, 32, 56, "b" => 12);

// ... este array
array(5 => 43, 6 => 32, 7 => 56, "b" => 12);
?>
```

2.9. Tipo Object

É o tipo de dado para armazenar uma instância de classe. É criado através da expressão **new**.

3. Variáveis e Constantes

3.1. Variáveis

Agora que já foi mostrado os tipos de dados que o PHP surporta, vamos aprender a definir variáveis e constante para armazenar dados para nossa aplicação.

Pense em variáveis como uma forma para armazer e representar os dados necessários para sua aplicação.

Para definir uma variável é utilizado o simbolo do cifrão (**$**), e, como dito anteriormente o PHP possui uma tipagem fraca, não sendo necessário definir qual tipo de variável sera criada, seu tipo será definido em tempo de execução.

```php
<?php
$var = 'João'; //Tipo string
$var = 1; //Tipo integer
$var = false; //Tipo boolean
?>
```

A única regra para criação de variaveis é que seu nome deve começar com um sublinhado ou uma letra, seguido de qualquer outra letra, número ou sublinhado.

O PHP interpreta os nomes das váriaveis como "**Case Sensitive**", o que implica as variáveis **$carro**, **$Carro**, **$CARRO**, **$caRro** são variáveis diferentes que podem armazenar diferestes valores.

```php
<?php
$nome = 'João'; //Variável válida
$2nome = 'Maria'; //Inválida
$_r = false; //Válida
?>
```

3.2. Constantes

Constantes funcionam como variáveis, mas possuem valor imutável, ou seja, o valor que foi definidor não pode ser alterado no decorrer da execução do programa. As constantes também são "**Case sensitive**".

```php
<?php

// Nomes de constantes válidos
define("FOO",     "alguma coisa");
define("FOO2",    "alguma outra coisa");
define("FOO_BAR", "alguma coisa mais");

// Nomes de constantes inválidas
define("2FOO",    "alguma coisa");

// Isto é válido, mas deve ser evitado:
// O PHP pode vir a fornercer uma constante mágica
// que danificará seu script
define("__FOO__", "alguma coisa");

?>
```

4. Operadores

Um operador é algo que você alimenta com um ou mais valores (ou expressões, no jargão de programação) e que devolve outro valor (e por isso os próprios construtores se tormam expressões). Assim, você pode pensar que as funções e os construtores que retornam valores (como o print) são operadores e os outros que não retornam nada (como echo) como uma outra coisa.

Há três tipos de operadores. Primeiramente, os operadores unários, que operam em apenas um valor. Por exemplo, ! (operador de negação) ou o ++ (operador de incremento). No segundo grupo estão os operadores binários, o o grupo que contém a maioria dos operadores que o PHP suporta, com uma lista completa logo abaixo na seção Precedência de operadores.

O terceiro grupo é do operador ternário: ?:. Ele pode ser usado para selecionar entre dois valores dependendo de uma terceira, em vez de selecionar duas sentenças ou encadeamentos de execução. Englobar expressões ternárias com parênteses é uma boa idéia.

4.1. Precedência de Operadores

A precedência de um operador especifica quem tem mais prioridade quando há duas delas juntas. Por exemplo, na expressão, 1 + 5 * 3, a resposta é 16 e não 18 porque o operador de multiplicação ("*") tem prioridade de precedência que o operador de adição ("+"). Parênteses podem ser utilizados para forçar a precedência, se necessário. Assim, (1 + 5) * 3 é avaliado como 18. Se a precedência do operador é igual, a associatividade da esquerda para direita é usada.

A tabela seguinte mostra a precedência dos operadores, da maior precedência no começo. Operadores com a mesma precedência estão na mesma linha, no caso a associatividade deles decidide qual ordem eles são avaliados.

Associação	Operador
não associativo	clone new
esquerda	[

Associação	Operador
não associativo	++ --
não associativo	~ - (int) (float) (string) (array) (object) (bool) @
não associativo	instanceof
direita	!
esquerda	* / %
esquerda	+ - .
esquerda	<< >>
não associativo	< <= > >= <>
não associativo	== != === !==
esquerda	&
esquerda	^
esquerda	|
esquerda	&&
esquerda	||
esquerda	? :
direita	= += -= *= /= .= %= |= ^= <<= >>=
esquerda	and
esquerda	xor
esquerda	or
esquerda	,

4.2. Operadores Aritméticos

Operadores básicos para realizar cálculos de adição, subtração, multiplicação, divisão e módulo.

Exemplo	Nome	Resultado
-$a	Negação	Oposto de $a
$a + $b	Adição	Soma de $a e $b
$a - $b	Subtração	Diferença entre $a e $b
$a * $b	Multiplicação	Produto de $a e $b
$a / $b	Divisão	Quociente de $a e $b
$a % $b	Módulo	Resto de $a dividido por $b

4.3. Operadores de atribuição

O operador básico de atribuição é "**=**". A sua primeira inclinação deve ser a de pensar nisto como "é igual". Não. Isto quer dizer, na verdade, que o operando da esquerda recebe o valor da expressão da direita (ou seja, "é configurado para").

O valor de uma expressão de atribuição é o valor atribuído. Ou seja, o valor de "**$a = 3**" é 3. Isto permite que você faça alguns truques:

```
<?php

$a = ($b = 4) + 5; /* $a é igual a 9 agora e
$b foi configurado como 4.*/

?>
```

4.4. Operadores de bit-bit

Operadores bit-a-bit permitem que você acione ou desligue bits específicos dentro de um inteiro. Se ambos os parâmetros da esquerda e da direita forem strings, esses operadores irão trabalhar nos valores ASCII dos caracteres.

```php
<?php
echo 12 ^ 9; // Imprime '5'

echo "12" ^ "9";
// Imprime o caracter de volta (backspace - ASCII 8)
// ('1' (ASCII 49)) ^ ('9' (ASCII 57)) = #8

echo "hallo" ^ "hello";
// Imprime os valores ASCII #0 #4 #0 #0 #0
// 'a' ^ 'e' = #4

echo 2 ^ "3"; // Imprime '1'
// 2 ^ ((int)"3") == 1

echo "2" ^ 3; // Imprime '1'
// ((int)"2") ^ 3 == 1

?>
```

Operadores Bit-a-bit

Exemplo	Nome	Resultado
$a & $b	E	Os bits que estão ativos tanto em $a quanto em $b são ativados.
$a | $b	OU	Os bits que estão ativos em $a ou em $b são ativados.
$a ^ $b	XOR	Os bits que estão ativos em $a ou em $b, mas não em ambos, são ativados.

Exemplo	Nome	Resultado
~ $a	NÃO	Os bits que estão ativos em $a não são ativados, e vice-versa.
$a << $b	Deslocamento à esqueda	Desloca os bits de $a $b passos para a esquerda (cada passo significa "multiplica por dois")
$a >> $b	Deslocamento à direita	Desloca os bits de $a $b passos para a direita (cada passo significa "divide por dois")

4.5. Operadores de comparação

Operadores de comparação, como os seus nomes implicam, permitem que você compare dois valores. Você pode se interessar em ver as tabelas de comparação de tipos, que tem exemplo das várias comparações entre tipos relacionadas.

Exemplo	Nome	Resultado
$a == $b	Igual	Verdadeiro (TRUE) se $a for igual a $b
$a === $b	Idêntico	Verdadeiro (TRUE) se $a for igual a $b e forem do mesmo tipo
$a != $b	Diferente	Verdadeiro (TRUE) se $a for diferente de $b
$a <> $b	Diferente	Verdadeiro (TRUE) se $a for diferente de $b
$a !== $b	Não idêntico	Verdadeiro se $a não for igual e $b ou não forem do mesmo tipo
$a < $b	Menor que	Verdadeiro se $a for menor que $b
$a > $b	Maior que	Verdadeiro se $a for maior que $b
$a <= $b	Menor ou igual	Verdadeiro se $a for menor ou igual a $b

Exemplo	Nome	Resultado
$a >= $b	Maior ou igual	Verdadeiro se $a for maior ou igual a $b

4.5. Operadores de Incremento/Decremento

O PHP suporta operadores de pré e pós-incremento e decremento no estilo C. Os operadores incremento/decremento não afetam valores booleanos. Decrementando valores NULL não há efeito também, mas incrementando resulta em 1.

Operadores de Incremento / Decremento

Exemplo	Nome	Efeito
++$a	Pré-incremento	Incrementa $a em 1 e retorna $a
$a++	Pós-incremento	Retorna $a e então incrementa $a em 1
--$a	Pré-decremento	Decrementa $a em 1 e então retorna $a
$a--	Pós-decremento	Retorna $a e então decrementa $a em 1

Alguns exemplos de incremento e decremento:

```php
<?php
echo "<h3>Pós incremento</h3>";
$a = 5;
echo "Deve ser 5: " . $a++ . "<br />\n";
echo "Deve ser 6: " . $a . "<br />\n";

echo "<h3>Pré incremento</h3>";
$a = 5;
echo "Deve ser 6: " . ++$a . "<br />\n";
echo "Deve ser 6: " . $a . "<br />\n";

echo "<h3>Pós decremento</h3>";
$a = 5;
echo "Deve ser 5: " . $a-- . "<br />\n";
echo "Deve ser 4: " . $a . "<br />\n";

echo "<h3>Pré decremento</h3>";
$a = 5;
echo "Deve ser 4: " . --$a . "<br />\n";
echo "Deve ser 4: " . $a . "<br />\n";
?>
```

4.6. Operadores Lógicos

Usandos para comparar dois valores, obtendo o resultado verdadeiro (true) ou falso (false).

Tabela de operadores lógicos no PHP

Exemplo	Nome	Resultado
$a and $b	E	Verdadeiro (TRUE) se $a e $b forem verdadeiros.
$a && $b	E	Verdadeiro (TRUE) se $a e $b forem verdadeiros.
$a or $b	OU	Verdadeiro se $a ou $b forem verdadeiros.
$a ll $b	OU	Verdadeiro se $a ou $b forem verdadeiros.

Exemplo	Nome	Resultado
! $a	NÃO	Verdadeiro se $a for falso
$a xor $b	XOR	Verdadeiro se $a ou $b são verdadeiros mas não ambos

```php
<?php

// $a será false
$a = (false && false);
// $b será false
$b = (true  && false);
// $c será verdadeiro
$c = (true && true);
// $d será verdadeiro
$d = (true  and  true);

//$e sera true
$e = false || true;
// $f sera true
$f = false or true;

?>
```

4.7. Operadores de String

Há dois operadores de string. O primeiro é o operador de concatenação ('.'), que retorna a concatenação dos seus argumentos direito e esquerdo. O segundo é o operador de atribuição de concatenação ('.='), que acrescenta o argumento do lado direito no argumento do lado esquerdo.

```php
<?php
$a = "Olá ";
$b = $a . "mundo!"; // agora $b contém "Olá mundo!"

$a = "Olá ";
$a .= "mundo!";// agora $a contém "Olá mundo!"
//é igual a $a = $a . “mundo” --> $a = “Olá “ . “mundo!”
?>
```

4.8. Operadores de Arrays

Segue a lista dos operadores de array considerando que ambas as variáveis do exemplo são do tipo array.

Exemplo	Nome	Resultado
$a + $b	União	União de $a e $b
$a == $b	Igualdade	TRUE se $a e $b tiverem os mesmos pares de chave/valor
$a === $b	Identico	TRUE se $a e $b tem os mesmos pares de chave/valor na mesma ordem e do mesmo tipo.
$a != $b	Desigualdade	TRUE se $a for diferente de $b
$a <> $b	Desigualdade	TRUE se $a for diferente de $b
$a !== $b	Não Identico	TRUE se $a não for identico a $b

O operador + acrescenta os elementos da direita no array da esquerda, contudo, chaves duplicadas NÃO são sobrescritas.

```php
<?php
$a = array("a" => "maçã", "b" => "banana");
$b = array("a" =>"pêra", "b" => "framboesa", "c" => "mor
ango");

$c = $a + $b; // Uniao de $a e $b
echo "União de \$a e \$b: \n";
var_dump($c);

$c = $b + $a; // União de $b e $a
echo "União de \$b e \$a: \n";
var_dump($c);
?>
```

Esse código vai produzir o seguinte resultado:

```
União de $a e $b:
array(3) {
  ["a"]=>
  string(5) "maçã"
  ["b"]=>
  string(6) "banana"
  ["c"]=>
  string(6) "morango"
}
União de $b e $a:
array(3) {
  ["a"]=>
  string(4) "pêra"
  ["b"]=>
  string(10) "framboesa"
  ["c"]=>
  string(6) "morango"
}
```

Exemplo com operadores de igualdade:

```php
<?php
$a = array("maçã", "banana");
$b = array(1 => "banana", "0" => "maçã");

var_dump($a == $b); // bool(true)
var_dump($a === $b); // bool(false)
?>
```

5. Estruturas de Controle

Qualquer script PHP é construído por uma série de instruções. Uma instrução pode ser uma atribuição, uma chamada de função, um 'loop', uma instrução condicional, ou mesmo uma instrução que não faz nada (um comando vazio). Instruções geralmente terminam com um ponto e vírgula. Além disso, as instruções podem ser agrupados em um grupo de comandos através do encapsulamento de um grupo de comandos com chaves. Um grupo de comandos é uma instrução também. Os vários tipos de instruções são descritos neste capítulo.

5.1. If / else

O construtor **if** é um dos recursos mais importantes de várias linguagens de programação, incluse o PHP. Ele permite a execução condicional de um trecho de código. No PHP a estrutura do **if** é similar a linguagem C.

```
if (expressao)
  codigo a ser executado
```

A “expressao” na condicional **if** será avaliada como um valor boleano (verdadeiro ou falso), e o código só será executado caso a condicional retorna um valor verdadeiro (true).

No exemplo abaixo será exibido a mensagem “a é maior que b”, caso o valor contido na variável $a for maior que o contido na variavel $b.

```
<?php
if ($a > $b)
  echo "a é maior que b";
?>
```

Geralmente é necessário executar mais de um comando dentro de uma condicional **if**, isso pode ser feito criando blocos de código com chaves **{}**.

Por exemplo, o código abaixo vai exibir a mensagem que “a é maior que b”, caso o valor contido na variável $a for maior que o contido na variavel $b, e será atribuido a variável $b o valor contido na variável $a.

```
<?php
if ($a > $b) {
  echo "a é maior b";
  $b = $a;
}
?>
```

Outra grande necessidade é executar um trecho de código quando uma expressão for atendida (true) e outra caso a mesma expressão não for atendida (false).

Pegando como exemplo o código anterior, na qual exibe a mensagem que “a é maior que b” caso a expressão ($a > $b) usada no **if** seja atendida (verdadeira). Para que seja possível executar outro comando caso a expressão não seja atendida (falso), fazemos o uso do construtor **else**.

Então para exibir uma outra mensagem que “a não é maior que b” caso a expressão não seja atendida fica assim:

```
<?php
if ($a > $b) {
  echo "a é maior b";
  $b = $a;
} else {
  echo "a não é maior b";
}
?>
```

Outra necessidade geralmente comum é testar mais de uma expressão, ou fazer varias expressões encadeadas, para isso pode ser usado o construtor **elseif**, que permite testar outra opção caso a anterior tenha falhado, exemplo:

```php
<?php
if ($a > $b) {
    echo "a é maior que b";
} elseif ($a == $b) {
    echo "a é igual a b";
} else {
    echo "a é menor que b";
}
?>
```

No exemplo anterior é testado se a variavel $a é maior que $b, caso essa expressão seja verdadeira, será exibido a mensagem “a é maior que b”. Caso essa primeira expressão falhe (falso) será testado a segunda expressão que é se a variavel $a for igual a $bPode-se utilizar vários **elseif** como no exemplo abaixo:

```
if (expressao) {
    codigo
} elseif (expressao) {
    codigo
} elseif (expressao) {
    codigo
} else {
    codigo
}
```

5.1. switch

O **switch** funciona de forma similar a uma série de **if / elseif** em uma mesma expressão. É extremamente útil para comparar varios valores sobre uma mesma expressões, como por exemplo:

```
<?php
if ($i == 0) {
    echo "i é igual a 0";
} elseif ($i == 1) {
    echo "i é igual a 1";
} elseif ($i == 2) {
    echo "i é igual a 2";
}

switch ($i) {
    case 0:
        echo "i equals 0";
        break;
    case 1:
        echo "i equals 1";
        break;
    case 2:
        echo "i equals 2";
        break;
}
?>
```

No exemplo acima, a mesma comparação feito usando **if / elseif** é feita utilizando o **switch**. Note somente o uso do construtor **break**, que é usado para interromper a execução quando cai na execução dessa condição.

5.2. for

O for é utilizado para criar laços. Os laços permitem que você execute o mesmo código repetidamente. A estrutura basica do **for** é a seguinte:

```
for (expressao; expressao2; expressao3)
    codigo a ser executado
```

A primeira expressão (expressao) é sempre executada no começo do laço.

A segunda expressão (expressao2) é sempre avaliada no começo de cada iteração. Se o resultado for verdadeiro (true) o código é executado. Caso for falso (falso) o laço é interrompido.

E finalmente a terceira expressão (expressao3) é executada no fim de cada iteração.

Cada um das expressões podem ser vazias ou conter multiplas expressões separadas por vírgulas. Na segunda expressão todas as expressões separadas por virgula são avaliadas mas é utilizado como resultado somente o resultado da última parte.

Se a segunda expressão (expressao2) for vazia, o PHP vai interpretar como verdadeira e rodar indefinidademente, pois não haverá como ser alcançado a condição de falso para interromper o loop. Nesse caso para interromper o laço pode-se utilizar o comando **break** para parar a execução do laço.

Veja alguns exemplos de utilização (todos os exemplos imprimem os números na tela de 1 até 10):

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

for ($i = 1; $i <= 10; $i++) {
    echo $i;
}

/* exemplo 2 */

for ($i = 1; ; $i++) {
    if ($i > 10) {
        break;
    }
    echo $i;
}

/* exemplo 3 */

$i = 1;
for (; ; ) {
    if ($i > 10) {
        break;
    }
    echo $i;
    $i++;
}

/* exemplo 4 */

for($i = 1, $j = 0; $i <= 10; $j += $i, print $i, $i++);
?>
```

5.2. while

O laço **while** é o tipo de laço mais simples do PHP. Sua estrutura básica é a seguinte:

```
while (expressao)
    codigo a ser executado
```

O funcionamento do laço **while** é bem simples, enquando a "expressao" a ser avaliada for verdadeira (true), o codigo do laço será executado. Quando a "expressao" retornar um valor falso (false) o laço é interrompido.

O exemplo imprime os números de 1 até 10 na tela:

```
<?php

$i = 1;
while ($i <= 10) {
    echo $i++;
}
?>
```

Outro laço que podemos realizar é o **do-while**, que é muito similar ao while. A única diferença é que a expressão é testada no final de cada iteração ao invés de ser no começo, executando assim o código do bloco pelo menos uma vez.

No exemplo abaixo é impresso na tela a variável **$i** que possui o valor 0, após isso é feito o teste para verificar se $i é maior que 0 ($i > 0). Como $i esta configurado com o valor 0 a expressão vai retornar falso encerrando assim o laço.

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

5.3. foreach

O laço **foreach** provê uma maneira fácil de iterar sobre arrays. Funciona apenas com os tipos **array** e **objeto** e ira retornar um erro se for usada com outro tipo de dado ou uma variável não iniciada.

Há dois tipos de de sintaxe para o **foreach:**

```
foreach (array as $valor)
    codigo


foreach (array as $chave => $valor)
    codigo
```

Na primeira forma do laço em cada iteração o valor do elemento atual é atribuido à variável $valor.

Na segunda forma acontece a mesma coisa apenas com a diferença que a chave do elemento atual é atribuida à variável $chave.

Exemplos:

```php
<?php
/* exemplo 1: imprime apenas valores */

$a = array(1, 2, 3, 17);

foreach ($a as $v) {
    echo "O valor atual de \$v: $v.\n";
}

/* exemplo 2: imprime o valor e a posição do array */

$a = array(1, 2, 3, 17);

$i = 0; /* apenas para fins educativos */

foreach ($a as $v) {
    echo "\$a[$i] => $v.\n";
    $i++;
}

/* exemplo 3: imprime chave e valor */

$a = array(
    "um" => 1,
    "dois" => 2,
    "tres" => 3,
    "desessete" => 17
);

foreach ($a as $k => $v) {
    echo "\$a[$k] => $v.\n";
}
?>
```

6. Funções

Uma função básicamente é um pedaço de código que realiza algum tipo de tarefa específica, podendo ser chamada em qualquer parte de seu aplicativo. Algumas das vantagens do uso de funções é clareza do código, separando assim cada parte do seu código, o que facilita o entendimento. A reutilização também é uma vantagem, muitas vezes você vai precisar executar um determinado trecho de código várias vezes, quando isso acontecer a melhor solução é criar uma função, evitando repetir todo o código e centralizando em um único lugar o que torna mais fácil as manutenções futuras.

Uma função pode ser definida com a seguinte sintaxe:

```php
<?php
function teste ($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Exemplo de função.\n";
    return $valor_retornado;
}
?>
```

Para se declarar uma função utiliza-se a palavra reservada **function** seguido pelo nome da função que você deseja criar, a nomenclatura utilizada para criar funções segue a mesma utilizada para criar nomes de variáveis. O corpo da função (o código que ela irá executar) deve ficar contido entre chaves {}. Para chama uma função basta escrever o nome da função desejada seguido de parenteses ().

Uma função pode ter 0 ou N argumentos, dependendo de sua necessidade.

Não há obrigatoriedade de ter um valor de retorno para função (**return**).

Exemplo de função:

```php
<?php
//Define a função teste sem argumentos
function teste ()
{
    echo "Olá mundo.";
}

//Executa a função
teste(); //Irá imprimir Olá mundo.


//Define uma função com argumento
function teste ($complemento)
{
    echo "Olá " . $complemento;
}

//Executa a função
teste("mundo"); //Irá imprimir Olá mundo

//Executa a função
teste("batata"); //Irá imprimir Olá batata
?>
```

Para se retornar valores é utilizado a palavra reservada **return**, qualquer tipo de dado pode ser retornado de uma função. Ao se utilizar o return faz a execução da fução ser iniciada.

Ao retornar um valor de uma função ele não é impresso em tela, a função apenas devolve o valor de retorno. O valor de retorno pode ser atribuido a uma variável ou usado diretamente em uma expressão.

```php
<?php
function quadrado ($num)
{
    return $num * $num;
}
echo quadrado (4);   // imprime '16'.
?>
```

O PHP ainda conta com várias funções internas para facilitar o desenvolvimento de suas aplicações. Uma lista completa dessas funções pode ser encontrada no próprio site do PHP http://www.php.net/manual/pt_BR/funcref.php

7. Orientação a objetos

O PHP começou como uma linguagem procedural sendo introduzido suporte a objetos a partir da versão 4. Claro que como não foi uma linguagem construida sobre o paradigma de orientação a objetos seu suporte não foi perfeito, mas tudo isso já é passado tendo em conta que a partir do PHP 5 toda o modelo de objetos foi reescrito tendo assim uma melhor performance e suporte a mais funcionalides.

A Programação Orientada a Objetos utiliza os conceitos fundamentais que todos sabemos: objetos e atributos, todos e partes, classes e membros.

Na compreensão do mundo real, as pessoas empregam constantemente três métodos de organização, sempre presentes em todos os seus pensamentos:

- Diferenciação, baseado na experiência de cada um, de objetos particulares e seus atributos - quando distinguem uma árvore, e seu tamanho ou relações espaciais, dos outros objetos;

- Distinção entre objetos como um todo e entre suas partes componentes - por exemplo, quando separam uma árvore dos seus galhos;

- Formação de, e distinção entre, as diferentes classes de objetos - por exemplo, quando formam uma classe de todas as árvores, uma outra classe de todas as rochas e distinguem-nas.

A Programação Orientada a Objetos se apóia nestes três métodos usuais de organização. Programação Orientada a Objetos é a programação implementada pelo envio de mensagens a objetos.

Cada objeto irá responder às mensagens conhecidas por este, e cada objeto poderá enviar mensagens a outros, para que sejam atendidas, de maneira que ao final do programa, todas as mensagens enviadas foram respondidas, atingindo-se o objetivo do programa.

Programação Orientada a Objetos, técnicas e artefatos ditos “orientados a objetos” incluem linguagens, sistemas, interfaces, ambientes de desenvolvimento, bases de dados, etc.

7.1 Classe

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.

Uma pseudo variável, $this, está disponível quando um método é chamado dentro de um contexto de objeto.

```php
<?php

//define uma classe
class A
{
    //define um método
    function ola()
    {
        echo "Olá mundo.";
    }
}


//instancia classe A e chama método
$instancia_a = new A();

$instancia_a->ola();

?>
```

7.2 Visibilidade

Ao criarmos uma classe vamos definir variaveis e funções, chamados de propriedades e métodos. Mas nem sempre queremos deixar esses métodos e propriedades acessiveis (ou visiveis) para serem acessados por outra classe, para isso podemos definir a visibilidade de cada método e propriedade.

A visibilidade de uma propriedade ou método pode ser definida prefixando a declaração com as palavras-chave: 'public','protected' ou 'private'. Itens declarados como public podem ser acessados por todo mundo. Protected limita o acesso a classes herdadas (e para a classe que define o item). Private limita a visibilidade para apenas a classe que define o item.

Com **public** o método ou propriedade não terão restrição de acesso, sendo acessivel por qualquer elemento interno ou externo.

Com **protected** o método ou propriedade só serão acessiveis dentro da classe ou classes que herdarem esses métodos.

E com **private** os elementos só serão visiveis dentro da classe.

Exemplo de visibilidade:

```php
<?php
/**
 * Define MinhaClasse
 */
class MinhaClasse
{
    public $publica = 'Publica';
    protected $protegida = 'Protegida';
    private $privada = 'Privada';

    public function imprimeAlo()
    {
        echo $this->publica;
        echo $this->protegida;
        echo $this->privada;
    }
}

$obj = new MinhaClasse();
echo $obj->publica; // Funciona
echo $obj->protegida; // Erro Fatal
echo $obj->privada; // Erro Fatal
$obj->imprimeAlo(); //Mostra Public, Protected e Private
```

7.3 Propriedades

As variáveis de uma classe são chamadas de propriedades (properties), suportam todos os tipos de dados já citados anteriormente. Podem ser definidas usando as palavras reservadas de visibilidade **public, protected** e **private**.

```php
<?php
/**
 * Define MinhaClasse
 */
class MinhaClasse
{
    public $publica = 'Publica';
    protected $protegida = 'Protegida';
    private $privada = 'Privada';
}
```

7.4 Abstração de classes

PHP 5 introduz métodos e classes abstratos. Não é permitido criar uma instância de uma classe que foi definida como abstrata e qualquer classe que contenha pelo menos um método abstrato deve também ser abstrata.

Métodos definidos como abstratos simplesmente declaram a assinatura do método, eles não podem definir a implementação.

Quando uma classe herda uma classe abstrata, todos os métodos marcados como abstratos na declaração da classe pai devem ser definidos na classe filha; além disso, esses métodos devem ser definidos com a mesma (ou menos restrita) visibilidade.

Por exemplo, se um método abstrato é definido como protected, a implementação da função deve ser definida ou como protected ou public, mas não private.

Além disso, as assinaturas dos métodos devem coincidir, ou seja, as induções de tipos e o número de argumentos requeridos devem ser os

mesmos. Isto também se aplica aos construtores a partir do PHP 5.4. Antes do 5.4 as assinaturas dos construtores poderiam ser diferentes.

```php
<?php
abstract class ClasseAbstrata
{
// Força a classe que estende ClasseAbstrata a definir esse método
    abstract protected function pegarValor();

    // Método comum
    public function imprimir() {
        print $this->pegarValor();
    }
}

class ClasseConcreta1 extends ClasseAbstrata
{
    // obrigado definir esse método
    protected function pegarValor() {
        return "ClasseConcreta1";
    }
}

?>
```

7.4 Herança

A herança funciona de forma a permitir que métodos e propriedades definidas em uma classe possa ser herdada por outra.

Se prestarmos atenção um carro e uma moto possuem características bem parecidas. Ambos possuem rodas ou até mesmo marchas, que são características semelhantes de qualquer veículo. Sendo assim podemos dizer que carro é um veículo. Assim como uma moto também é um veículo. Portanto ambos – carro e moto – herdam características do objeto veículo.

No PHP para dizer que uma classe herda os métodos e propriedades de outra usamos a palavra reservada **extends**, como no exemplo:

```
<?php
class Veiculo
{
     public $marcha;

     public $quantidadeRodas;

     public function passarMarcha()
     {
          // código
     }

     public function andar()
     {
          // código
     }
}

class Carro extends Veiculo
{
     public function __construct()
     {
          $this->quantidadeRodas = 4;
     }
}

$carro = new Carro();
$carro->andar();
?>
```

8. Banco de dados

O PHP oferece suporte a vários banco de dados, mas essa parte tera foco no banco de dados MySQL. Mas entendendo como funciona o processo de conexão com a base de dados e consultas fica fácil de aplicar a outros banco de dados.

Para se conectar com o banco de dados MySQL podemos utilizar a extensão **mysqli** ou **PDO_MySQL**.

Basicamente, para ter acesso as informações em um banco de dados, temos os seguintes passos:

1. Conectar ao servidor MySQL;
2. Selecionar o banco de dados;
3. Enviar a consulta;
4. Receber os dados;
5. Processar, caso tenha necessidade, e imprimir na tela;

Para se conectar ao servidor e selecionar o banco de dados fazemos o uso da classe **mysqli**, ela possui a seguinte definição:

```
mysqli(endereco, usuario, senha, banco_de_dado, porta)
```

Sendo:

• **endereco**: Nome ou IP onde está localizado o servidor MySQL;
• **usuario**: Usuário para se conectar ao servidor MySQL;
• **senha**: Senha para se conectar ao servidor MySQL;
• **banco_de_dados**: Nome da base de dados que será utilizada;
• **porta**: Porta do serviço MySQL, pode ser deixado em branco e nesse caso é utilizado a porta padrão do serviço 3306

Exemplo de conexão com servidor e banco de dados:

```php
<?php
//conecta com servidor e banco de dados
$mysqli = new mysqli("localhost", "usuario", "senha", "b
anco");

//verifica se houve algum erro ao conectar
if ($mysqli->connect_errno) {
    echo "Erro ao conectar com mysql: " .
           $mysqli->connect_error;
}
?>
```

Após a conexão podemos fazer o uso do método **query** para realizar as consultas no banco de dados.

```php
<?php
//conecta com servidor e banco de dados
$mysqli = new mysqli("localhost", "usuario", "senha", "b
anco");

//verifica se houve algum erro ao conectar
if ($mysqli->connect_errno) {
    echo "Erro ao conectar com mysql: " .
           $mysqli->connect_error;
}

$consulta = $mysqli->query("SELECT * FROM cidade");

while ($resultado = $consulta->fetch_assoc())
{
    print_r($resultado);
}
?>
```

O comando **query** também pode ser utilizado para INSERIR, ATUALIZAR e DELETAR informações além de recuperar.

Já o comando **fetch_assoc()** utilizado sobre o resultado da consulta, retorna um array dos resultados. Também pode ser obtido um objeto com os resultados da consulta, para isso basta utilizar o **fetch_object()**

Mais informações sobre métodos podem ser encontrados na documentação oficial do PHP (http://php.net)

9. Funcionalidades Web

9.1. Variáveis superglobais

No PHP existe uma série de variáveis pré-definidas que são chamadas de “superglobais”, disponíveis em todos os escopos para todo o script, sendo as principais delas:

$GLOBALS: Um array associativo contendo referências para todas as variáveis que estão atualmente definidas no escopo global do script. O nome das variáveis são chaves do array.

$_SERVER: $_SERVER é um array contendo informação como cabeçalhos, paths, e localizações do script. As entradas neste array são criadas pelo servidor web. A lista completa de chaves possíveis está em: http://www.php.net/manual/pt_BR/reserved.variables.server.php

$_GET: Um array associativo de variáveis passadas para o script atual via o método HTTP GET.

$_POST: Um array associativo de variáveis passados para o script atual via método HTTP POST.

$_FILES: Um array associativo de items enviado através do script atual via o método HTTP POST.

$_COOKIE: Um array associativo de variáveis passadas para o atual script via HTTP Cookies.

$_SESSION: Um array associativo contendo variáveis de sessão disponíveis para o atual script.

$_REQUEST: Um array associativo que por padrão contém informações de $_GET, $_POST and $_COOKIE.

$_ENV: Um array associativo de variáveis passadas para o script atual via o método do ambiente.

9.2. Sessões

Sessões permite que o PHP consiga armazenar os dados através de requisições ou acessos subsequentes.

Para se iniciar uma sessão fazemos o uso da função **session_start()**, para registrar uma variável usamos o **session_register()** ou da variável superglobal **$_SESSION[]**.

Para melhor compreenção podemos analizar o exemplo a seguir:

```php
<?php
// pagina1.php

session_start();

echo 'Bem vindo a pagina #1';

$_SESSION['favcolor'] = 'verde';
$_SESSION['animal']   = 'gato';
$_SESSION['time']     = time();

echo '<br /><a href="pagina2.php">pagina 2</a>';

?>
```

```php
<?php
// pagina2.php

session_start();

echo 'Bem vindo a pagina #2<br />';

echo $_SESSION['favcolor']; // verde
echo $_SESSION['animal'];   // gato
echo date('Y m d H:i:s', $_SESSION['time']);

echo '<br /><a href="pagina1.php">pagina 1</a>';
?>
```

9.3. GET

O PHP possui a variável superglobal **$_GET** que contém um array associativo de todas as variáveis passadas através de uma requisição HTTP GET.

```php
<?php
echo 'Olá ' . $_GET["nome"];
// ou print_r para imprimir todas as chaves e valores
print_r($_GET);
?>
```

9.4. POST

Da mesma forma que o **$_GET** temos a superglobal **$_POST** que contém um array associativo de todas as variáveis passadas através de uma requisição HTTP POST.

```php
<?php
echo 'Olá ' . $_POST["nome"];
// ou print_r para imprimir todas as chaves e valores
print_r($_POST);
?>
```

9.5. COOKIES

Cookies são um mecanismo para guardar dados no navegador remoto possibilitando o acompanhamento ou identificação de usuários que retornam. Você pode criar cookies usando a função setcookie() ou setrawcookie(). Os cookies são uma parte do cabeçalho HTTP, logo setcookie() precisa ser chamada antes que qualquer outro dado seja enviado ao navegador.

Qualquer cookie enviado por você para o cliente automaticamente será incluido na auto-global $_COOKIE.

Para setar um cookie utilizamos a função **setcookie()**

```
bool setcookie ( string $name [, string $value [, int $expire
= 0 [, string $path [, string $domain [,bool $secure = false
[, bool $httponly = false ]]]]] )
```

Se existe saída antes da chamada dessa função, setcookie() irá falhar e retornará FALSE. Se a função setcookie() for executada com sucesso, ela retornará TRUE. Isso não indica que o usuário aceitou o cookie.

Parâmetros da função:

name

O nome do cookie.

value

O valor do cookie. Esse valor é guardado no computador do cliente; não guarde informação sensível. Supondo que o **name** seja ***'nomedocookie'***, o valor pode ser lido través de *$_COOKIE['nomedocookie']*

expire

O tempo para o cookie expirar. Esse valor é uma timestamp Unix, portanto é o número de segundos desde a época (epoch). Em outras palavras, você provavelmente irá utilizar isso com a função time() mais o número de segundos que você quer que ele expire. Ou você pode utilizar a função mktime(). ***time()+60*60*24*30*** irá configurar o cookie para expirar em 30 dias. Se configurado para 0, ou omitido, o cookie irá expirar ao fim da sessao (quando o navegador fechar).

Nota:

Você pode ver que o parâmetro **expire** recebe uma timestamp Unix, ao contrário do formato de data ***Wdy, DD-Mon-YYYY HH:MM:SS GMT***, isso se dá porque o PHP faz essa conversão internamente.

path

O caminho no servidor aonde o cookie estará disponível. Se configurado para ***'/'***, o cookie estará dosponível para todo o **domain**. Se configurado para o diretório ***'/foo/'***, o cookie estará disponível apenas dentro do diretório ***/foo/*** e todos os subdiretórios como***/foo/bar*** do **domain**. O valor padrão é o diretório atual onde o cookie está sendo configurado.

domain

O domínio para qual o cookie estará disponível. Configurando o domínio para ***'www.exemplo.com'*** fará com que o cookie esteja disponível no subdomínio ***www*** e nos subdomínios superiores. Cookies disponíveis para um domínio inferior, como ***'example.com'*** estarão disponíveis para subdomínios superiores, como ***'www.exemplo.com'***. Browsers antigos ainda implementam a » RFC 2109 e podem requerer um **.** no início para funcionar com todos os subdomínios.

secure

Indica que o cookie só podera ser transimitido sob uma conexão segura HTTPS do cliente. Quando configurado para `TRUE`, o cookie será enviado somente se uma conexão segura existir. No lado do servidor, fica por conta do programador enviar esse tipo de cookie somente sob uma conexão segura (ex respeitando *$_SERVER["HTTPS"]*).

httponly

Quando for `TRUE` o cookie será acessível somente sob o protocolo HTTP. Isso significa que o cookie não será acessível por linguagens de script, como JavaScript. É dito que essa configuração pode ajudar a reduzir ou identificar roubos de identidade através de ataques do tipo XSS (entretanto ela não é suportada por todos os browsers), mas essa informação é constantemente discutida. Foi adicionada no PHP 5.2.0. `TRUE` ou `FALSE`

Exemplo de como setar um cookie:

```php
<?php
$value = 'alguma coisa de algum lugar';

setcookie("CookieTeste", $value);
/* expira em 1 hora */
setcookie("CookieTeste", $value, time()+3600);

/* Imprimindo valor do cookie */

// Mostra um cookie individual
echo $_COOKIE["CookieTeste"];
echo $HTTP_COOKIE_VARS["CookieTeste"];

/* Outra maneira de depurar(debug)/
testar é vendo todos os cookies */
print_r($_COOKIE);
?>
```

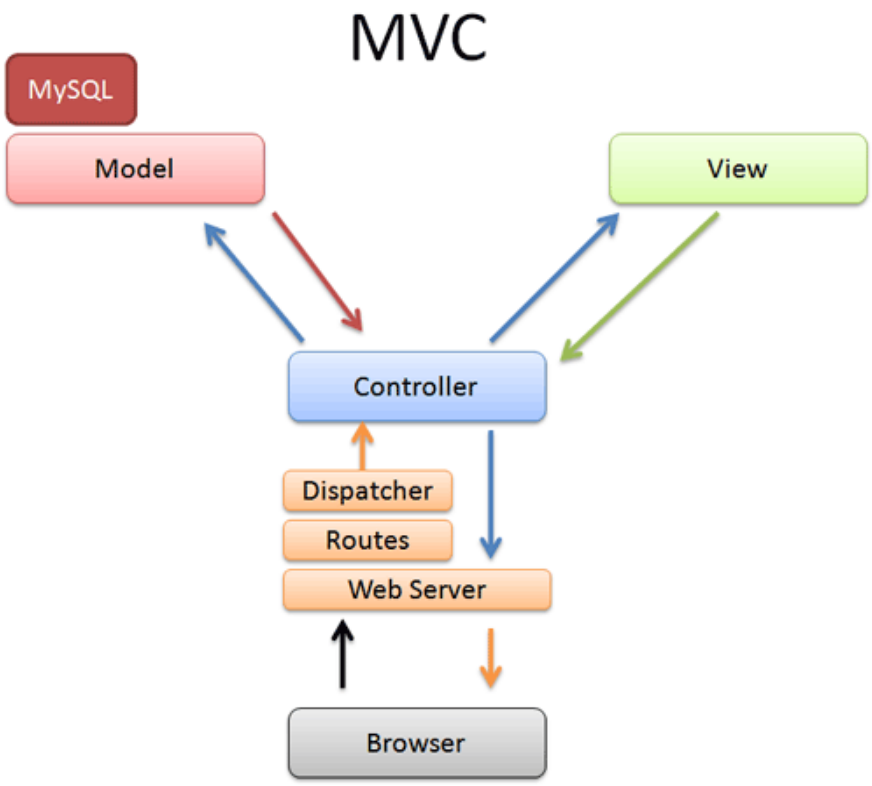
10. Kohana

O Kohana é um framework open source construído sobre o PHP5, orientado a objetos e com a estrutura MVC. Possui licença BSD, que permite que você o framework livremente para qualquer tipo de aplicação open source, comercial ou pessoal.

Possui uma baixa curva de aprendizado, é facilmente configurável, possui classes para tratar erros, tratar entrada de dados, cookies, formulários, geradores de HTML. A camada de banco de dados já provê proteção contra SQL Injection.

10.1. Padrão MVC

O Kohana foi feito sobre o padrão de projeto MVC, o que permite uma programação mais fluída. A principio a mudança de paradigma da programação estruturada para a orientada a objetos pode assustar, ainda mais já adotando a filosofia do MVC para criação dos projetos, mas isso facilita o desenvolvimento e manutenção do sistema.



O MVC ou Model, View, Controller visa a separação de responsabilidades no desenvolvimento da aplicação, código enxuto (DRY) e reuso do código.

O Model é responsável pela comunicação com banco de dados, obtenção de dados, validação, etc.

A View é responsável pela exibição dos dados para o usuário final, incluindo informações processadas, imagens, css, js.

A Controller cuida de toda a parte lógica da aplicação, processar os dados obtidos da Model enviar para a View.

10.2. Instalação

A instalação do Kohana é bem simples, basta fazer o download da última versão disponível no site (http://kohanaframework.org/). Extrair o conteúdo do zip, e colocar numa pasta dentro de seu servidor web.

Abra o arquivo **application/bootstrap.php** e faça as seguintes alterações:

Procure pela linha onde é definido o timezone e mude para o timezone certo (ex. America/Sao_Paulo)

```
// Example of changing timezone from Chicago to Sao
Paulo, Brazil
date_default_timezone_set('America/Sao_Paulo');
```

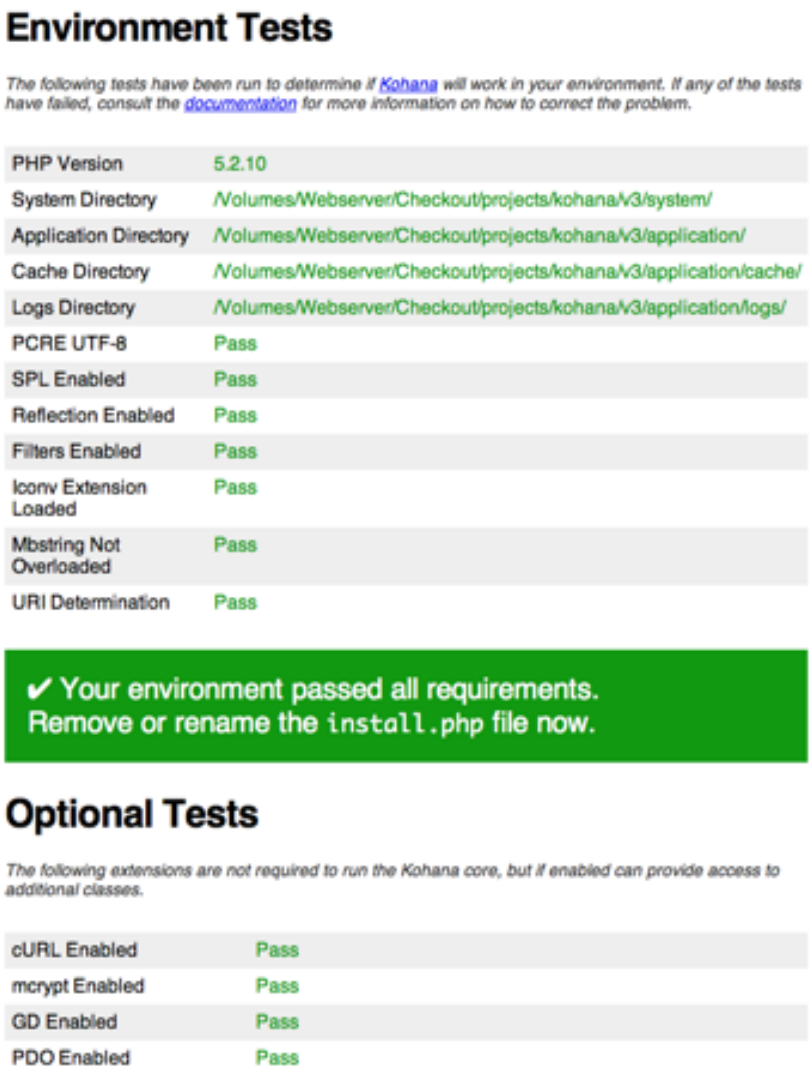
Configure o **base_url** para a pasta onde está o framework kohana.

```
//se a pasta onde estiver o kohana for meusite
Kohana::init(array(
    'base_url'   => '/meusite',
));
```

Defina permissões de leitura e escrita para as pastas **application/cache** e **application/logs**.

```
sudo chmod 777 -R application/cache
sudo chmod 777 -R application/logs
```

Abra seu site em algum navegador, o Kohana deverá apresentar uma tela de instalação listando todos os requisitos para o funcionamento do framework. Após confirmar que está tudo certo apague ou renomeie o arquivo **install.php**



Environment Tests

The following tests have been run to determine if Kohana will work in your environment. If any of the tests have failed, consult the documentation for more information on how to correct the problem.

PHP Version	5.2.10
System Directory	/Volumes/Webserver/Checkout/projects/kohana/v3/system/
Application Directory	/Volumes/Webserver/Checkout/projects/kohana/v3/application/
Cache Directory	/Volumes/Webserver/Checkout/projects/kohana/v3/application/cache/
Logs Directory	/Volumes/Webserver/Checkout/projects/kohana/v3/application/logs/
PCRE UTF-8	Pass
SPL Enabled	Pass
Reflection Enabled	Pass
Filters Enabled	Pass
Iconv Extension Loaded	Pass
Mbstring Not Overloaded	Pass
URI Determination	Pass

✔ Your environment passed all requirements.
Remove or rename the `install.php` file now.

Optional Tests

The following extensions are not required to run the Kohana core, but if enabled can provide access to additional classes.

cURL Enabled	Pass
mcrypt Enabled	Pass
GD Enabled	Pass
PDO Enabled	Pass

Após essas etapas provavelmente você verá uma tela de boas vindas:

```
hello, world!
```

10.3. Convenção e estilo de codificação

A idéia de adotar um framework vai além de agilizar o trabalho, mas também de adotar as melhores práticas para o desenvolvimento, por isso é importante se atentar aos padrões de codificação para escrever um código mais limpo e claro de ler e entender.

10.3.1. PHP-FIG e PSR's

Cada framework tem sua própria filosofia e melhores práticas de codificação, mas no ano de 2009 foi criado um grupo de desenvolvedores de vários frameworks para definir boas práticas de programação para a linguagem.

Não é um grupo oficial do PHP, mas é formado por representantes de vários frameworks atuais, o nome do grupo é PHP-FIG, sendo FIG um acrônimo para (Framework Interoperability Group).

Até o momento foram criadas quatro PSR, que são as melhores práticas para o desenvolvimento utilizando PHP, sendo elas:

PSR-0: Focado em manter uma convenção de arquivo, classe e namespace para o carregamento automático do código (autoloading) (https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-0.md**)**

PSR-1: Focado em definir os elementos padrões de codificação que são necessários para assegurar um elevado nível de interoperabilidade técnica entre o código PHP compartilhado. (https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md)

PSR-2: Focado no padrão de codificação utilizando PHP para padronização do código (https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md)

PSR-3: Define uma interface padrão para uma interface de log (https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-3-logger-interface.md)

10.3.1. Padrão de codificação no Kohana

Atualmente o Kohana não faz parte do grupo PHP-FIG, mas possui uma documentação bem detalhada do seu padrão de codificação e segue alguns padrões do PSR.

Nomes de classes e locais de arquivos

As classes no Kohana devem tem a primeira letra maiuscula e utilizar underscore(_) para separar as palavras.

Todas as classes devem ficar na pasta **classes.**

Exemplo:

Nome da classe	localização
Controller_Template	classes/Controller/Template.php
Model_User	classes/Model/User.php
Model_BlogPost	classes/Model/BlogPost.php
Database	classes/Database.php
Database_Query	classes/Database/Query.php
Form	classes/Database/Query.php

Uso de chaves em estruturas de controle

As chaves devem ser colocados em suas próprias linhas, indentado com o comando da estrutura de controle.

```php
// Forma correta
if ($a === $b)
{
    ...
}
else
{
    ...
}

// forma errada
if ($a === $b) {
    ...
} else {
    ...
}
```

Uso de chaves em classes

A excessão da regra de como devem ficar as chaves é na declaraçnao da classe onde a chave começa ao lado do nome da classe e não em sua própria linha.

```php
// Correto
class Abacate {

// Incorreto
class Abacate
{
```

Chaves vazias

Em caso de utilizar chaves vazias não coloque nenhum caracter entre eles (espaco ou nova linha)

```
// Correto
class Abacate {}

// Incorreto
class Abacate { }
```

Arrays

Os arrays podem ser definidos tanto em uma linha como em multiplas linhas.

```
array('a' => 'b', 'c' => 'd')

array(
    'a' => 'b',
    'c' => 'd',
)
```

Abrindo parenteses

A abertura de parenteses vai na mesma linha, sendo fechado no mesmo nível de declaração.

```
// Correto
array(
    ...
)

// Incorreto:
array
(
    ...
)
```

Convenção de nomes de classes

O Kohana utiliza underscore (_) no nome das classes e não o camelCase.

```
// Controller class, uses Controller_ prefix
class Controller_Apple extends Controller {

// Model class, uses Model_ prefix
class Model_Cheese extends Model {

// Regular class
class Peanut {
```

Quando for instanciar uma nova classe não utilize parenteses se não for passado nenhum parametro para o construtor da classe.

```
// Correto:
$db = new Database;

// Incorreto:
$db = new Database();
```

Funções e métodos

Devem ser todas em letras minusculas e utilizar underscore para separar as palavras.

```
function drink_beverage($beverage)
{
```

Variáveis

Mesmo padrão de métodos, deve-se utilizar letras minusculas e underscore.

```
// Correto:
$foo = 'bar';
$long_example = 'uses underscores';

// Incorreto:
$weDontWantThis = 'understood?';
```

Indentação

A indentação deve utilizar tabs e nunca espaços

Concatenação de string

Não utiliza espaços entre o operador de concatenação

```
// Correto:
$str = 'one'.$var.'two';

// Incorreto:
$str = 'one'. $var .'two';
$str = 'one' . $var . 'two';
```

Operadores de comparação

Utilizar **OR** e **AND** para comparação

```
// Correto:
if (($foo AND $bar) OR ($b AND $c))

// Incorreto:
if (($foo && $bar) || ($b && $c))
```

Utilizar **elseif** ou invés de **else if**

```
// Correto:
elseif ($bar)

// Incorreto:
else if($bar)
```

Constantes

Utilizar sempre caracteres maiusculos

```
// Correto:
define('MY_CONSTANT', 'my_value');
$a = TRUE;
$b = NULL;

// Incorreto:
define('MyConstant', 'my_value');
$a = True;
$b = null;
```

10.4. Configuração

A maior parte da configuração de ambiente do Kohana é feita no arquivo bootstrap, localizado em **application/bootstrap.php**

As principais configurações que devem ser feitas são timezone e locale

```
// Configuracao do timezone.
date_default_timezone_set('America/Chicago');

// Configuração do locale.
setlocale(LC_ALL, 'en_US.utf-8');

// Enable the Kohana auto-loader.
spl_autoload_register(array('Kohana', 'auto_load'));

// Enable the Kohana auto-loader for unserialization.
ini_set('unserialize_callback_func',
'spl_autoload_call');
```

Outra parte importante para configurar é **base_url**, que determina em qual pasta está seu servidor.

```
Kohana::init(array('
    base_url' => '/kohana/'
));
```

E finalmente os módulos e as rotas. O Kohana oferece alguns módulos padrão que podem ser habilitados no arquivo bootstrap. Para ativar um módulo basta descomentar a linha correspondente.

```
Kohana::modules(array(
      // 'auth'       => MODPATH.'auth',       // Basic
authentication
      // 'cache'      => MODPATH.'cache',      // Caching with
multiple backends
      // 'codebench'  => MODPATH.'codebench',  // Benchmarking tool
      // 'database'   => MODPATH.'database',   // Database access
      // 'image'      => MODPATH.'image',      // Image
manipulation
      // 'minion'     => MODPATH.'minion',     // CLI Tasks
      // 'orm'        => MODPATH.'orm',        // Object
Relationship Mapping
      // 'unittest'   => MODPATH.'unittest',   // Unit testing
      // 'userguide'  => MODPATH.'userguide',  // User guide and
API documentation
      ));
```

E no final do arquivo a configuração da rota padrão do framework.

```
Route::set('default', '(<controller>(/<action>(/<id>)))')
      ->defaults(array(
            'controller' => 'welcome',
            'action'     => 'index',
      ));
```

10.5. Fluxo da requisição

O fluxo começa no arquivo **index.php**, nessa etapa são definidos os caminhos da aplicação, dos modulos e do sistema (**APPPATH**, **MODPATH**, **SYSPATH**)

Então entra em ação o **bootstrap.php**, carregando a classe Kohana, então é chamado o método **Kohana:init** responsável por iniciar a aplicação, configurar URL padrão (**base_url**), gerenciador de erros, cache e log.

Nesta etapa também são carregados os módulos, o controle de rotas e a classe de request.

A classe **Request** (requisição) inicia, verifica se a rota existe, redireciona para o controller apropriado e devolve a reposta novamente para **index**, que exibe o resultado.

10.6. Rotas

O Kohana fornece um sistema de roteamento poderoso, permitindo fazer uma ligação com qualquer fragmento de URL com uma controller e action.

A rota padrão (default) do Kohana fica no arquivo bootstrap.php, cada rota no sistema deve ter um nome unico.

```
Route::set('default', '(<controller>(/<action>(/<id>)))')
->defaults(array(
    'controller' => 'Welcome',
    'action'     => 'index',
));
```

Este trecho de código cria a rota **default**, ao utilizar o método **Route::set** o primeiro parametro é o nome da rota, esse nome deve ser unico no sistema, e o segundo paramento é qual o padrão de URL que será casado neste exemplo é (<controller>(/<action>(/<id>))).

Esse padrão vai pegar o valor passado no token <controller> e procurar por um controller no sistema com o mesmo nome, e , fazer o mesmo com <action> e procurar nessa controller esse método.

Ainda é utilizado o **defaults** para setar um valor padrão para a controller e para a action caso não seja passado na URL.

Caso o valor da URL seja opcional ou não for presenta na rota, pode-se utilizar o **defaults** para definir valores padrões.

No caso **controller** e **action** devem sempre possuir um valor padrão.

O Kohana também converte automaticamente o nome do controller para o padrão de nomes, por exemplo:

/blog/ver/11 irá procurar pela controller **Controller_Blog.php** na pasta **classes/Controller/Blog.php** e executar a ação **action_ver()**

10.7. Módulos

O Kohana fornece alguns módulos básicos para o desenvolvimento de aplicações web, mas nada impede que você crie seus módulo ou baixe módulos prontos e utilize no framework. Um lugar recomendado pelo próprio Kohana para procurar por módulos é no endereço https://github.com/kolanos/kohana-universe/tree/master/modules/.

Os módulos devem ser ativados no método **Kohana::modules** através de um array associativo com 'nome' => 'caminho'. O nome não é importante, serve apenas para identificar o módulo, mas o caminho deve ser correto e apontar para onde esta os arquivos do módulo.

O método **Kohana::modules** fica no arquivo **bootstrap.php** e só pode ser chamado uma vez.

```
Kohana::modules(array(
    'auth'       => MODPATH.'auth',       // Basic authentication
    'cache'      => MODPATH.'cache',      // Caching with multiple
backends
    'codebench'  => MODPATH.'codebench',  // Benchmarking tool
    'database'   => MODPATH.'database',   // Database access
    'image'      => MODPATH.'image',      // Image manipulation
    'orm'        => MODPATH.'orm',        // Object Relationship
Mapping
    'oauth'      => MODPATH.'oauth',      // OAuth authentication
    'pagination' => MODPATH.'pagination', // Paging of results
    'unittest'   => MODPATH.'unittest',   // Unit testing
    'userguide'  => MODPATH.'userguide',  // User guide and API
documentation
    ));
```

Para ativar um módulo basta descomentar ou adicionar no método **Kohana:modules**

10.8. Sessões

O Kohana também fornece uma classe para facilitar o uso de sessões.

Para acessar a instância da sessão pode ser utilizado o método **Session::instance**.

```
// Pega a instancia da classe Session
$session = Session::instance();
```

Todos os dados da sessão podem ser obtidos como array com o método **Session::as_array**.

```
// pega todos os dados da sessao como array
$data = $session->as_array();
```

Para setar um novo valor da sessão pode ser usado o método **set**

```
// armazenar um novo valor
$session->set($chave, $valor);
// ou
Session::instance()->set($chave, $valor);

// Exemplo, armazenando id de um usuario
$session->set('user_id', 10);
```

Para pegar um valor já setado usamos o **get**

```
// Pegar um valor da sessao
$data = $session->get($chave, $valor_padrao);

// Pegar o id do usuario
$user = $session->get('user_id');
```

E, para deletar usamos o **delete**

```
// Deletar um valor da sessao
$session->delete($chave);


// Deletar o id do usuario (user_id)
$session->delete('user_id');
```

10.9. Banco de dados e Model

O suporte a banco de dados vem desabilitado no Kohana, para habilita-lo é necessário ativar no módulo de **database** e **orm** no arquivo **bootstrap.php**.

Após habilitar o módulo de bando de dados e ORM é necessário criar um arquivo de configuração com os dados de acesso ao banco de dados.

Um modelo do arquivo pode ser encontrado na pasta **modules/database/config/database.php**. Não altere esse arquivo, copie ele para a pasta **application/config** e então configure seu banco de dados.

```
<?php defined('SYSPATH') OR die('No direct access allowed.');

return array
(
      'default' => array
      (
            'type'       => 'MySQL',
            'connection' => array(

                  'hostname'   => 'localhost',
                  'database'   => 'kohana',
                  'username'   => 'root',
                  'password'   => 'senha111',
                  'persistent' => FALSE,
```

Com o acesso do Kohana ao banco de dados configurado já é possivel fazer as classes da Model (responsável pelas operações com o banco de dados).

Para criação dos Models o Kohana estabelece algumas regras:

• O nome da tabela deve ter o mesmo nome do Model só que no plural. Ex. **Model_Blog** a tabela deverá ser **blogs** para **Model_Post** a tabela devera ser **posts**.

• O plural das palavras são em inglês, caso você tenha alguma dúvida com o nome escolhido para a tabela pode usar o comando abaixo para verificar qual será o plural ou singular correspondente:

```
echo Inflector::plural('person'); //imprime people

echo Inflector::singlular('people'); //imprime person
```

Umas das vantagens do Kohana é o poder do ORM que permite operações de crud de uma maneira simples, bastando herdar (**extends**) a classe **ORM** para sua classe **Model**, como no exemplo:

```
class Model_Member extends ORM
{
    ...
}
```

Podemos carregar instancia da Model dessas formas

```
$user = ORM::factory('User');
// Ou
$user = new Model_User();
```

Para inserir um valor é bem simples, basta que sua Model herde a classe ORM. Exemplo:

```
// pegando a instancia
$usuario = ORM::factory('User');

// definindo dados
$usuario->first_name = 'Fulano';
$usuario->last_name = 'de tal';

// salvando
$usuario->save();
```

Para procurar um objeto podemos utilizar o método **ORM::find**, ou passa o ID do objeto diretamente para o construtor da classe ORM.

```
// Procurar usuário com ID 20
$user = ORM::factory('User')
    ->where('id', '=', 20)
    ->find();
// Ou no construtor
$user = ORM::factory('User', 20);
```

Para recuperar todos os objetos da tabela usamos o **ORM::find_all.**

```
// Retorna todos os usuario no banco
$users = ORM::factory('User')->find_all();
```

Para atualizar ou deletar basta recuperar o objeto com uma busca, e em seguida alterar o que for necessario e salva. Ou deletar o objeto.

```
// Procurar usuário com ID 20
$user = ORM::factory('User')
    ->where('id', '=', 20)
    ->find();

// Atualiza objeto (altera informações e
salva)
$user->first_name = "Ciclano";
$user->last_name = "de tal";
$user->save();

// Deletar
$user->delete();
```

10.10. Controller

As controller da sua aplicação devem ficar na pasta **application/ controller**, só lembre que o nome da controller e da action devem casar com a URL definida como rota.

Exemplo a URL:

```
/blog/all
```

Será redirecionada para a controller e action:

```
Controller_Blog::action_all
```

Para definir uma controller basta criar o arquivo dentro da basta **application/controller** e extender a classe **Controller** do Kohana, ficando assim:

application/controller/hello.php

```
<?php
defined('SYSPATH') OR die('No Direct Script Access');

class Controller_Hello extends Controller
{
    ...
}
```

E a partir dai é possível escrever as ações (actions) necessárias, toda a parte lógica da aplicação.

10.11. View e Formulários

As **views** devem apenas conter o que será exibido para o usuário, nada de fazer conexão com o banco de dados, a parte lógica da aplicação e exibir os resultados.

Na view vai apenas arquivos estaticos como HTML, CSS, arquivos javascript, imagens. E variáveis pré processadas pela controller como nome do usuario a exibir, idade, lista de produtos, etc.

As view também tem uma pasta especifica e devem ficam em **application/view**.

Na view não é necessário extender nenhuma classe, pode-ser escrever como se fosse um arquivo HTML.

Como exemplo vamos criar uma view chamada **"digaoi"** que exibe a mensagem "olá mundo" dentro da pasta **"ola"**

application/view/ola/digaoi.php

```
<h2>Olá Mundo</h2>
```

Para exibir essa view chamamos o método **View::factory** e apontamos para a nossa recém criada view.

```
<?php
defined('SYSPATH') OR die('No Direct Script Access');

class Controller_Hello extends Controller
{
    $view = View::factory('ola/digaoi');
    $this->response->body($view);
}
```

Bem simples, mas também tem o caso de passar uma variavel já processada para exibir na view, nesse caso basta utilizar o método **bind** ou **set**.

Vamos trocar a mensagem de “olá mundo” para exibir um olá e o nome passado, como “ola fulado”, “ola marcio”, etc.

A estrutura do método bind é essa:

```
$view->bind('ref', $var);
```

Sendo **ref** o nome que a variável terá na view, e **$var** sendo a variável em si, que contém os valores que você deseja exibir na tela.

Exemplo de uso do bind na controller:

```
<?php
defined('SYSPATH') OR die('No Direct Script Access');

class Controller_Hello extends Controller
{
    $view = View::factory('ola/digaoi')
      ->bind('nome', 'João');
    $this->response->body($view);
}
```

E na view:

```
<h2>Olá <?php echo $nome; ?></h2>
```

O Kohana possui várias classes auxiliares, uma delas é para criar formulários, o que facilita a vida do desenvolvedor poupando de criar várias tags HTML.

A classe **Form** é para ser usada na view de sua aplicação, ou seja, na parte responsável pelos dados a serem impressos na tela.

Criando um formulário na view:

```
<h2> Exemplo de formulario </h2>

<?php

echo Form::open();

echo Form::label('autor', 'Nome do autor');
echo Form::input('autor');

echo Form::label('mensagem', 'Corpo da Mensagem');
echo Form::textarea('mensagem');

echo Form::submit('submit', 'Enviar');

echo Form::close();

?>
```

Outros tipo de campos suportador pelos formulários são button, checkbox, file, password, radio, select.

Mais informações podem ser obtidas na documentação em http://kohanaframework.org/3.0/guide/api/Form

11. Gallery

O Gallery é uma aplicação web, de código fonte aberto (open source) para organização de fotos, albuns, videos.

Atualmente está na versão 3.0.9, é totalmente escrito em PHP com o framework Kohana, o site oficial do projeto é (http://galleryproject.org/) e o repositório no GitHub em https://github.com/gallery/gallery3/

11.1. Como se envolver no projeto

Como em todo projeto primeiro é necessário conhecer a ferramenta, os integrantes, qual os problemas atuais da ferramenta e qual o futuro. Para isso a maioria dos projetos open source fornece uma lista de emails, canais de IRC, forums, etc.

No Gallery essa lista pode ser obtida em http://codex.galleryproject.org/Gallery3#Getting_Involved_With_Development

É importante participar ativamente na comunidade, não só com contribuições de código mas estar a par das discuções da comunidade.

A lista de Bug tracking fica em http://sourceforge.net/apps/trac/gallery/

11.2. Ferramenta

O Gallery 3 é construido sobre o framework Kohana na versão 2.3, os conceitos de design da ferramenta é praticamente identico a forma que o Kohana funciona. Gallery 3 tem a seguinte estrutura:

application: Aplicação minima exigida para o funcionamento do Kohana.

system: É onde fica o Kohana framework e as bibliotecas do core

themes: Onde ficam os temas para o gallery

modules: Onde ficam todos os módulos do gallery. O módulo principal é o modules/gallery que carrega os outros módulos que estão intalados e ativos.

lib: Bibliotecas, arquivos de CSS, Javascript usados por módulos ou temas.

installer: Instalador do Gallery

11.2. Padrão de codificação

O Gallery possui seu próprio padrão de codificação e para que o código criado por você seja aceito na comunidade é importante se atentar aos padrões estabelecidos.

A lista completa dos padrões está em http://codex.galleryproject.org/Gallery3:Coding_Standards

A seguir segue as recomendações básicas estabelecidas pela comunidade:

Indetanção

Deve-se utilizar dois espaços para indentação. O uso de TABs é proibido.

```
function my_function() {
  // 2 espaços
  code;
  code;

  if (indent_another_level) {
    // outros 2 espaços
    more_code;
    more_code;
  }
}
```

Tamanho da linha

Evite escrever linhas com mais de 100 caracteres em seus arquivos.

Padrão de nomes

Os padrões de nomes para arquivos, classes, variáveis, contantes, tabelas em banco de dados, Model seguem o padrão estabelecido pelo Kohana Framework.

11.2. Módulo no Gallery

Um módulo no gallery 3 é um conjuntos de models, views, controller, bibliotecas e arquivos de configuração reunidos em uma uma pasta dentro de **gallery3/modules**. Todos esses elementos citados são usados para adiconar um recurso especifico ao Gallery. Somente existe um arquivo obrigatório ao criar um módulo par ao Gallery chamado de **module.info** ele contem o nome do módulo, descrição e versão.

Vamos pegar como exemplo um módulo **hello_world,** o unico arquivo obrigatório é o module.info

```
hello_world
└── module.info
```

Seu conteúdo pode ser algo como:

```
name = "Hello world!"
description = "Este módulo nao faz nada!"
version = 1
```

Com isso o módulo já podera ser ativado ou desativado no menu **Admin > Modules**, mas não irá fazer nada. Para adicionar uma funcionalidade ao módulo é necessário adiconar uma controller, utilizar alguns dos ganchos (hooks) disponíveis na ferramenta.

Por exemplo, para exibir um “Hello world” no topo de cada álbum podemos utilizar o hook theme, na pasta **helpers** e criar um arquivo chamado **hello_world_theme.php**.

```
<?
class hello_world_theme {
  static function album_top($theme) {
    return "Hello world";
  }
}
```

Algumas considerações:

O Gallery espera que o módulo que implementa o hook theme fique localizado na pasta **modules/<nome_do_modulo>/helpers/ <nome_do_modulo>_theme.php**

Sempre que o Gallery exibir um álbum ele tentará chamar **hello_world_theme::album_top** no topo da página e **hello_world_theme::album_bottom** no final.

Poderiamos ter um módulo mais complexo como esse por exemplo:

```
modules
└── search
    ├── controllers
    │   └── search.php
    ├── helpers
    │   ├── search_event.php
    │   ├── search_installer.php
    │   ├── search.php
    │   ├── search_task.php
    │   └── search_theme.php
    ├── models
    │   └── search_record.php
    ├── module.info
    └── views
        ├── search.html.php
        └── search_link.html.php
```

Este módulo tem uma controller. Possui também hook de evento no arquivo **search_event.php**, um hook de installer no **search_installer.php**, uma tarefa administrativa no arquivo **search_task.php** e um hook theme no arquivo **search_theme.php.**

Implementa também uma model (search_record.php) e duas views.

Uma lista completa de hooks e um guia prático de desenvolvimento pode ser encontrado aqui: https://docs.google.com/document/pub?id=1DQ1Tz177aX5ZsSgEbjYtYOBufJ0SYhgYvfJtA6snn2Y#h.j5rr3jqbb5fz