PHP Básico, 10 encontros

Por **Hábner Nascimento**

Contato: 81 9 9575.7394

Nosso cronograma

Coisas legais em PHP
Mitos e verdades
Ambiente WEB

Por: Hábner Nascimento
habner@treinamentoinfoco.com.br

Sobre o PHP

- PHP é interpretado
- PHP é open source
- Pode ser embutido no HTML
- Conecta aos pricipais bancos de dados como MySQL, Oracle e SQL Server
- Pode ser usado em todos os principais sistemas operacionais
- Gera várias saídas como XML, PDF, Flash em tempo de execução

Um pouco de história...

PHP/FI (Personal home page tools/Form interpreter)

- Criado em 1994 por Rasmus Lerdof, a primeira versão era um conjunto de binários CGI (common gateway interface) escrito em C. Ele fez isso para acompanhar visitas de seu currículo online, ele nomeou o conjunto de script de “Personal Home Page Tools” mais conhecido como “PHP Tools”, em junho de 1995 Rasmus liberou o código fonte do PHP Tools, nessa versão já era possível criar sites dinâmicos e conexão com banco de dados.

PHP/FI 2.0

- Em setembro de 1995 Rasmus expandiu e mudou o nome para PHP FI abreviação de “Form Interpreter”, tinha variáveis estilo Perl, interpretação automática de variáveis de formulário, em Outubro o código foi todo reescrito. A linguagem foi desenvolvida para ser parecida com C

PHP 3.0

- Ainda em 1997, dois israelenses, Andi Gutmans e Zeev Zuraski, ao utilizar o PHP/FI 2.0 como base de um projeto de eCommerce na Universidade, resolveram reformula-lo por completo, lançando seu sucessor na comunidade, o PHP 3.0.
- Uma das primeiras iniciativas foi mudar o seu próprio nome para P HP H ypertext P reprocessor.
- Zend Technologies (http://www.zend.com).

PHP 4.0

- Em 1998, começou a ser desenvolvido um novo núcleo do PHP (introduzido na versão 3.0). Este foi chamado de Zend Engine em função da empresa de Zeev e Andi.
- Lançado oficialmente em maio de 2002, veio com uma série de novas funcionalidades incluindo o laço para coleções foreach e o gerenciamento de sessão HTTP (session). A versão 4 trousse consigo na base de uns 15 milhões de domínios.

>= PHP 5.0

- Atualmente na versão 5.5 o PHP traz novidades e cada vez mais responsabilidades para os desenvolvedores e provando que pode ser usada em sistemas robustos e com grande performance.
- Uma orientação a objeto mais robusta(iniciada no PHP4).
- Melhor suporte a XML.
- Namespace
- Goto

>= PHP 7.2

- Suporte a mais 2 novos tipos de dados
- Sobrescrita em métodos abstratos
- Number_format não retorna mais zero negativo (-0)
- Chaves numéricas no casting de array/object
- Descontinuação
  - `__autoload` (prefira usar `spl_autoload_register`)
  - `$php_errormsg` (prefira usar `error_get_last()`)
  - `create_function()`
  - `mbstring.func_overload`
  - `(unset)` cast
  - `parse_str()` sem um segundo argumento
  - `gmp_random()` (prefira usar `gmp_random_bits()` e `gmp_random_rage()`)
  - `each()`

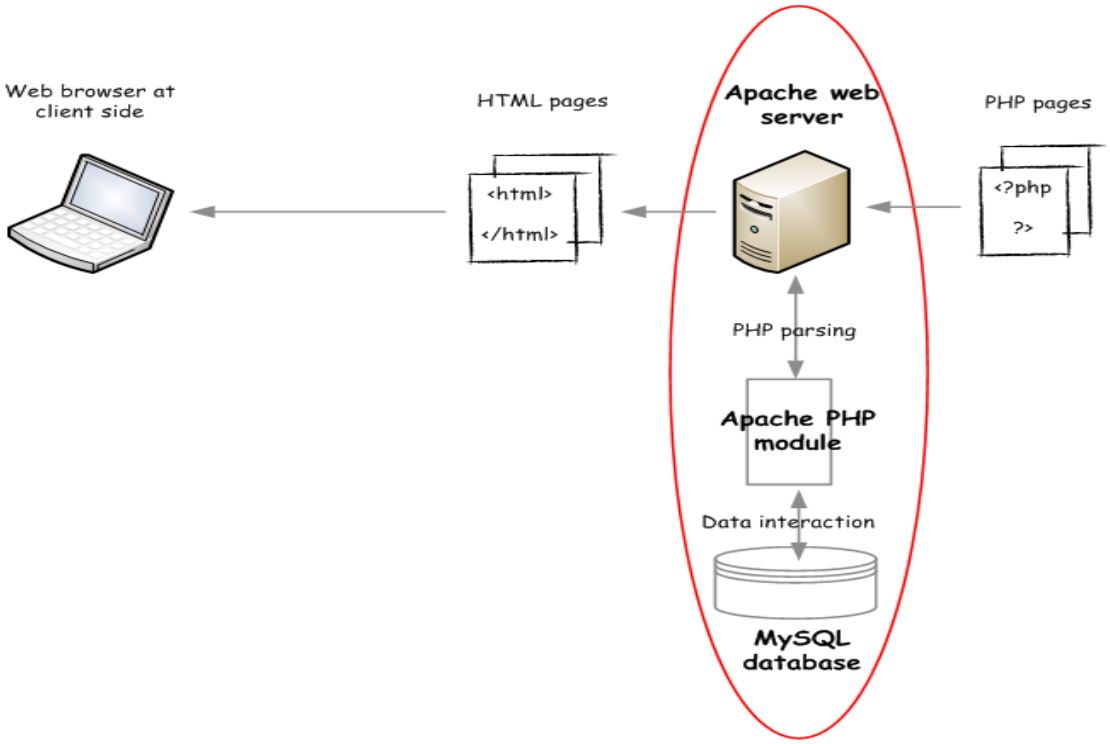
- `assert()` com uma string no argumento
- O argumento `$errcontext` de `set_error_handler()` foi descontinuado.

https://www.treinaweb.com.br/blog/novidades-do-php-7-2/

Ambiente Web

Muitos sistemas estão migrando para o ambiente web devido a facilidade de acesso em tempo real e atualização.

Como o PHP funciona?



Como podemos observar, o HTML é a principal forma de comunicação utilizada na internet!

Vamos **c**onhecer um pouco de HTML, afinal de contas ela é a interface fundamental para iteração com nosso usuário final!

Para o ambiente web funcionar precisamos de dois elementos: o cliente (navegador) e o servidor (onde tem o apache e o php instalados), como o PHP é uma linguagem de programação web, tudo será feito em html e será visualizado o resultado no cliente, ok?
Nesse ponto vamos instalar o apache...
...dessa forma o apache diz o seguinte, tudo que estiver na pasta c:\xampp\htdocs será executado no cliente através de localhost

HTML

HTML (abreviação para a expressão inglesa HyperText Markup Language, que significa Linguagem de Marcação de Hipertexto) é uma linguagem de marcação utilizada para produzir páginas na Web. Documentos HTML podem ser interpretados por navegadores.

Os elementos de entrada são definidos por tags, que iniciam com “<” e finalizam com “>”.

A tag principal é a input, com ela podemos criar 90% dos formulários da maioria dos sistemas, ela é composta pelos atributos “name” e “value”, onde name é o identificador desse campo que recuperaremos logo mais.

Exemplo:

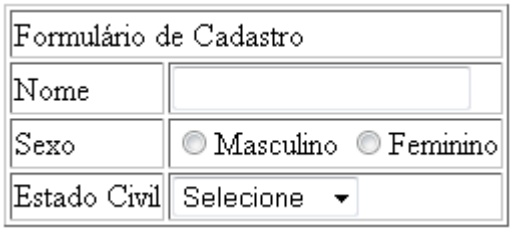
```
<input type=”text” name=”nome”>
<input type=”radio” name=”nome”>
<input type=”checkbox” name=”nome”>
```

Os types da tag input pode ter os valores text, checkbox, radio, hidden

text: campo aberto para digitação
checkbox: Marca e desmarca
radio: Grupo de campos onde qualquer um é marcado mas apenas um é marcado
hidden: Campo invisível no formulário

Exercício

Este belo código acima após interpretado pelo navegador reproduzirá a seguinte saída



Nosso primeiro script PHP, no arquivo 1

```
<input type=”text” name=”nome”>
```

No arquivo 2

```
<?php
        Echo $_GET[‘nome’];
```

Coisas legais em PHP
Configuração PHP
Primeiros scripts

Por: Hábner Nascimento
habner@treinamentoinfoco.com.br

Configuração PHP é algo tão importante quanto programar, certa vez um profissional me questionou algumas "fraquezas" do php, simplesmente perguntei você conhece o **php.ini**? Claro que sabia a resposta, e era não.

Quando o Apache é iniciado as diretivas descritas no php.ini são levadas em consideração.

Algumas relevâncias descritas no **php.ini**

session.save_path string, nessa diretiva você define onde os arquivos de sessão serão gravados

```
; The file storage module creates files using mode 600 by default.
; You can change that by using
;
;     session.save_path = "N;MODE;/path"
;
; where MODE is the octal representation of the mode. Note that this
; does not overwrite the process's umask.
; http://php.net/session.save-path
;session.save_path = "/tmp"

session.save_path = "/var/www/sessao"

; Whether to use cookies.
; http://php.net/session.use-cookies
session.use_cookies = 1
```

Nesse caso todas as $_SESSION serão gravadas no diretório /var/www/sessao, não esqueça de dar permissão de escrita!

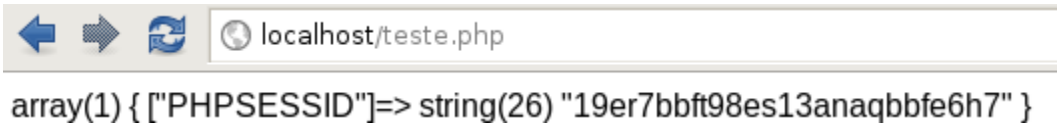
session.name string, essa diretiva sinaliza o nome do cookie que terá o identificador da sessão

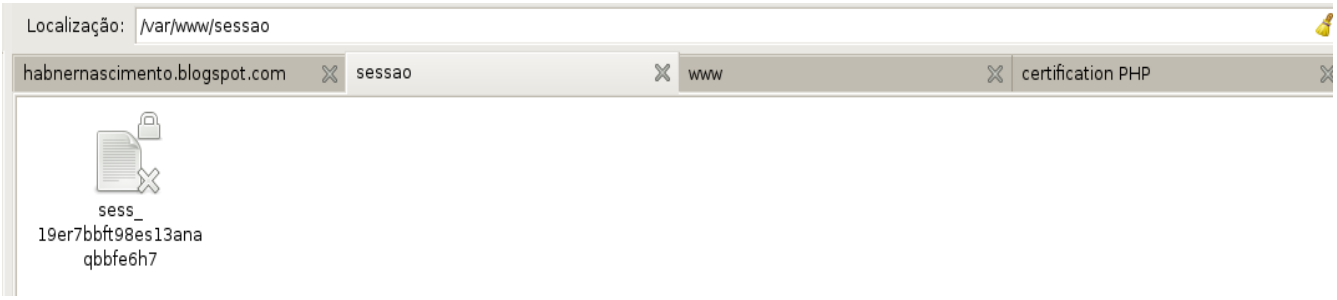
```
; Name of the session (used as cookie name).
; http://php.net/session.name
session.name = PHPSESSID
```

executando o script abaixo para verificar o resultado

```
teste.php
1  <?php
2
3  session_start();
4  $_SESSION['teste'] = sha1(222);
5
6  var_dump($_COOKIE);
```

executando **teste.php** teremos como resultado





Lembrando que cada vez que se alterar o **php.ini** tem que se reiniciar o apache/IIS para levantar as configurações novamente.

post_max_size integer, limita o tamanho máximo de dados que podem ser tranferidos via método post. Por exemplo:

O valor padrão dessa diretiva é **8M**, se tivermos um formulário para cadastro de imagens que suporte um envio de 10 imagens simultaneamente teremos uma mensagem de erro informando o motivo.

session.auto_start boolean, o default é 0 (disabled) isto significa que antes de trabalhar com $_SESSION deve-se starta-la usando a função session_start(); caso o auto_start esteja **1** (enabled) não precisaremos usar o session_start ok?

A diretiva ***include_path*** string é uma das melhores! Mas, veremos ela mais à frente.

Guarde essa diretiva, iremos usar mais à frente. **track_errors = Off**

file_uploads boolean or integer simplesmente habilita ou desabilita o upload de arquivos para programas php.

A diretiva ***display_errors*** *habilita ou desabilita a exibição de erros na tela do usuário*

```
<?php
error_reporting  (E_ALL);
ini_set ('display_errors', true); //aqui assume true/false/1/0
include('./test_file.php');
```

Vamos testar!?

Aprendendo a comunicar-se

J*á aprendemos que a melhor forma em termos de interface de comunicação é usar HTML, precisamos aprender a estrutura abaixo*

O método POST também um método do HTTP, é acionado por meio de um formulário HTML através do atributo method=post incluída na tag <form>

http://www.comocriarsites.com/html/como-funciona-os-metodos-get-e-post-diferencas/

```
<form method="post" action="cadastrar.php">
        <input type="text" name="vNome" />
        <input type="submit" value=">> Enviar" />
</form>
```

O método GET

GET, um dos métodos do HTTP, é acionado por meio de um formulário HTML através do atributo method="get" incluída na tag <form>. Por meio desse método, os dados do formulário são transmitidos ao software servidor e este por sua vez, armazena os dados temporariamente numa variável de contexto denominada QUERY_STRING.

```
<form method="get" action="cadastrar.php">
        <input type="text" name="vNome" />
        <input type="submit" value=">> Enviar" />
</form>
```

O método GET envia os dados ao servidor pela url exemplo:

http://projeto.com.br/cadastrar.php?vNome=Sarah

A diretiva **disable_functions =** é muito útil, ela permite que você desabilite algumas funções.

```
<?php
        mkdir('disab');
        scandir('.');
```

No php.ini fica assim

disable_functions = scandir,mkdir

Saída:

Warning: mkdir() has been disabled for security reasons in /var/www/teste.php on line 5 Warning: scandir() has been disabled for security reasons in /var/www/teste.php on line 6

Lembrando que após cada mudança no php.ini terá que reiniciar o serviço do Apache

Praticando...

1° Configure o php para não mostrar os erros e não aceitar criação e remoção de arquivos e diretórios.

Coisas legais em PHP
Tipo de dados
Variáveis e constantes

Por: Hábner Nascimento
php@treinamentoinfoco.com.br

Ao c**o**ntrário do que se pensa o PHP organiza seus tipos primitivo da seguinte forma...

group: ***scalar***
- boolean
- integer
- float/double
- string

group: ***composto***
- array
- object
- callable
- iterable

group: ***special***
- resource
- NULL

É **i**mportante saber disso pois tanto nesse material quanto no site oficial do php (php.net) se vê muito usado o termo “esse campo aceita apenas tipos do grupo composto” e assim por diante, para mim o conhecimento desses grupos é um diferencial para os programadores.

Destrinchando um pouco cada tipo teremos

boolean esse é um tipo simples e é representado por **TRUE** ou **FALSE**, lembrando que o tipo boolean é case-insensitive

```
$boleano = TRUE; //FALSE, true, false
```

Para converter explicitamente usa-se **bool** ou **boolean**.

Atenção nesse momento pois alguns dos valores abaixo são considerados false

- the boolean `FALSE` itself
- the integer 0 (zero)
- the float 0.0 (zero)
- the empty string, and the string "0"
- an array with zero elements

- the special type NULL (including unset variables)
- SimpleXML objects created from empty tags

Qualquer outro valor será considerado **TRUE...** *-1* é considerado *true* ou *false*? Resolva as questões abaixo

```
<?php
var_dump((bool) "");        //
var_dump((bool) 1);         //
var_dump((bool) -2);        //
var_dump((bool) "foo");     //
var_dump((bool) 2.3e5);     //
var_dump((bool) array(12)); //
var_dump((bool) array());   //
var_dump((bool) "false");   //
?>
```

integer assume qualquer valor inteiro positivo ou negativo, o inteiro pode ser expressado nas bases decimal, hexadecimal, octal ou binário.

Para usar a notação octal precisa apenas preceder o número com 0 (zero), lembre-se que não pode usar aspas ou apóstrofos, ok? Pois assim será considerado literal.

```
<?php
$a = 1234; // decimal number
$a = -123; // a negative number
$a = 0123; // octal number (equivalent to 83 decimal)
$a = 0x1A; // hexadecimal number (equivalent to 26 decimal)
$a = 0b11111111; // binary number (equivalent to 255 decimal)
?>
```

Estrutura do integer

```
decimal     : [1-9][0-9]*
              | 0

hexadecimal : 0[xX][0-9a-fA-F]+

octal       : 0[0-7]+

binary      : 0b[01]+

integer     : [+-]?decimal
              | [+-]?hexadecimal
              | [+-]?octal
              | [+-]?binary
```

O tamando de um inteiro depende da plataforma, para saber com precisão basta verificar o valor da constante `PHP_INT_MAX` **e** `PHP_INT_MIN`

Atenção redobrada para tipo octal e hexadecimal

```
<?php
        var_dump(01090); // 010 octal = 8 decimal
?>
```

Caso o inteiro seja muito grande ele muda automaticamente de tipo, por isso cuidado e atenção.

Example #2 Integer overflow on a 32-bit system

```
<?php
$large_number = 2147483647;
var_dump($large_number);                     // int(2147483647)

$large_number = 2147483648;
var_dump($large_number);                     // float(2147483648)

$million = 1000000;
$large_number =  50000 * $million;
var_dump($large_number);                     // float(50000000000)
?>
```

float/double pode ser chamado de float, double ou real numbers, um float pode ser declarado como

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

O tamando de um float depende da plataforma, mas, acredite é bem grande!

String é uma série de caracter, um caracter assume um byte, o PHP suporta um conjunto de 256 caracteres e portanto não oferece suporte nativo a Unicode. Uma string pode ter até 2GB de tamanho.

Podemos especificar uma string de quatro maneiras, apóstrofo, aspas, heredoc e nowdoc(PHP 5.3)

Vejamos

Usando apóstrofos você sinaliza uma string no sentido literal, ou seja caso tenha uma variável dentro da string ela não será interpretada

```
$name = 'Sarah';
echo 'my name is $name';
```

saída:

my name is $name

Quando usamos **aspas** o php interpreta as variáveis caso haja, nesse caso

```
$name = 'Sarah';
echo “my name is $name”;
```

saída:
my name is Sarah

A sintaxe HereDoc e NowDoc são bem parecidas a diferença é que a nowdoc é como se estivesse usando apóstrofos e a heredoc não.

Usando heredoc

```
$name = 'Sarah';
$text = <<< DELIMITADOR
meu texto aqui dentro junto
com minhas variáveis,
a variável \$name tem o valor $name.
DELIMITADOR;

echo $text;
```

Logo a saída será

meu texto aqui dentro junto com minhas variáveis, a variável $name tem o valor Sarah.

Muito importante** na linha que fecha o heredoc (**DELIMITADOR;**) só pode ter esse código de fechamento.

Usando nowdoc

```
$name = 'Sarah';
$text = <<< 'DELIMITADOR'
meu texto aqui dentro junto
com minhas variáveis,
a variável \$name tem o valor $name.
DELIMITADOR;

echo $text;
```

Logo a saída será

meu texto aqui dentro junto com minhas variáveis, a variável \$name tem o valor $name.

OK?

Array em PHP é um mapa ordenado, é uma associação de **chave** e **valor** nas próximas lições falaremos exclusivamente sobre array.

Object esse também é outro tópico abordado mais a frente e tem uma aula exclusiva para esse assunto.

Resource é uma variável especial, é tudo aquilo externo que o php usa para realizar tal ação. A exemplo disso são as dlls que também estão descritas no **php.ini** onde pode ser habiiltada e desabilitada.

NULL é um valor especial que não representa valor. Uma variável é considerada null se for atribuido a constante null, se não tiver valor definido e quando usar a função unset().

Variáveis são representadas por um dollar ($), logo após deve vir um underscore ou uma letra para só assim vir um número, nunca usar caracteres especiais ou começar uma variável com número, as variáveis são case-sensitive.

```
$_nome; //correta
$nome;//correta
$1;//incorreta
```

*A variável $this é predefinida pelo PHP para simbolizar objetos.

```
<?php
	$var = 'Bob';
	$Var = 'Joe';
	echo "$var, $Var";      // outputs "Bob, Joe"

	$4site = 'not yet';     // invalid; starts with a number
	$_4site = 'not yet';    // valid; starts with an underscore
	$täyte = 'mansikka';    // valid; 'ä' is (Extended) ASCII 228.
?>
```

Temos duas formas de atribuição a por valor e a por referência, por padrão todas as variáveis são atribuidas por valor, para atribuir por referência basta usar o sinal & antes de definir a variável.

```
<?php
	$foo = 'Bob';              // Assign the value 'Bob' to $foo
	$bar = &$foo;              // Reference $foo via $bar.
	$bar = "My name is $bar";  // Alter $bar...
	echo $bar;
	echo $foo;                 // $foo is altered too.
?>
```

Saída

My name is Bob
My name is Bob

O que não podemos passar por referência

```
<?php
        $foo = 25;
        $bar = &$foo;      // This is a valid assignment.
        $bar = &(24 * 7);  // Invalid; references an unnamed expression.

        function test()
        {
          return 25;
        }

        $bar = &test();    // Invalid.
?>
```

No PHP **não é necessário** inicializar ou declarar uma variável, porém é considerado uma **boa prática**. Caso não seja inicializado dependendo do contexto algumas variáveis podem assumir o valor default FALSE para bool, inteiros e float para zero, string fica vazio e array fica sem elementos.

Constantes são identificadores para valores simples, ou que não sofram alteração ao decorrer da execução do programa. Por padrão uma constante é case-sensitive e por convenção são declaradas em maiúsculo. O nome de uma constante segue a mesma regra para nomes de variáveis.

Defini-se uma constante usando a função

bool define (string $name , mixed $value [, bool $case_insensitive = false])

nesse caso

```
// Valid constant names
define("FOO",     "something");
define("_FOO2",    "something else");
define("FOO_BAR", "something more");

// Invalid constant names
define("2FOO",    "something");
```

Você deve ter observado que a função define tem dois parâmetros obrigatórios e um terceiro opcional, bem, este último diz se a constante vai ser sensitive ou insensitive, caso queira deixar a constante insensitive basta colocar o terceiro parâmetro como true.

Por que é importante saber os grupos e tipos primitivos do PHP? Por causa do resultado

abaixo.

Warning: Constants may only evaluate to scalar values in /var/www/teste.php on line 5 Notice: Use of undefined constant _FOO2 - assumed '_FOO2' in /var/www/teste.php on line 7 _FOO2

resultado do **code**

```
define("_FOO2",    array());
echo _FOO2;
```

Com isso entendemos que apenas tipos escalares podem ser colocados como valores de constantes.

Usamos a função **bool defined (string $name)** para verificar se uma constante já foi definida.

Além disso existem oito constantes mágicas, recebem esse nome por ser constantes variáveis!

A few "magical" PHP constants

Name	Description
__LINE__	The current line number of the file.
__FILE__	The full path and filename of the file. If used inside an include, the name of the included file is returned. Since PHP 4.0.2, **__FILE__** always contains an absolute path with symlinks resolved whereas in older versions it contained relative path under some circumstances.
__DIR__	The directory of the file. If used inside an include, the directory of the included file is returned. This is equivalent to ***dirname(__FILE__)***. This directory name does not have a trailing slash unless it is the root directory. (Added in PHP 5.3.0.)
__FUNCTION__	The function name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the function name as it was declared (case-sensitive). In PHP 4 its value is always lowercased.
__CLASS__	The class name. (Added in PHP 4.3.0) As of PHP 5 this constant returns the class name as it was declared (case-sensitive). In PHP 4 its value is always lowercased. The class name includes the namespace it was declared in (e.g. ***Foo\Bar***). Note that as of PHP 5.4 __CLASS__ works also in traits. When used in a trait method, __CLASS__ is the name of the class the trait is used in.
__TRAIT__	The trait name. (Added in PHP 5.4.0) As of PHP 5.4 this constant returns the trait as it was declared (case-sensitive). The trait name includes the namespace it was declared in (e.g. ***Foo\Bar***).
__METHOD__	The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).
__NAMESPACE__	The name of the current namespace (case-sensitive). This constant is defined in compile-time (Added in PHP 5.3.0).

http://www.php.net/manual/en/reserved.constants.php

Exercício com include e require

Coisas legais em PHP
Operadores

Por: Hábner Nascimento
php@treinamentoinfoco.com.br

See more http://br2.php.net/manual/en/language.operators.precedence.php

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

Operadores **a**ritméticos

+$a	Identity	Conversion of *$a* to int or float as appropriate.
-$a	Negation	Opposite of *$a*.
$a + $b	Addition	Sum of *$a* and *$b*.
$a - $b	Subtraction	Difference of *$a* and *$b*.
$a * $b	Multiplication	Product of *$a* and *$b*.
$a / $b	Division	Quotient of *$a* and *$b*.
$a % $b	Modulo	Remainder of *$a* divided by *$b*.
$a ** $b	Exponentiation	Result of raising *$a* to the *$b*'th power. Introduced in PHP 5.6.

Operador de Atribuição

O operador básico de atribuição é **=**, dessa forma podemos atribuir o valor da direita ao elemento da esquerda.

$a = ($b = 4) + 5; //nessa forma o valor de $a será 9

Para atribuir o valor de uma chave de um array usa-se o operador **=>.**

Você ainda pode combinar alguns operadores

```
$a = 3;
$a += 5; // sets $a to 8, as if we had said: $a = $a + 5;
$b = "Hello ";
$b .= "Sarah!";
```

Operador de comparação

Comparison Operators		
Example	Name	Result
$a == $b	Equal	**TRUE** if *$a* is equal to *$b* after type juggling.
$a === $b	Identical	**TRUE** if *$a* is equal to *$b*, and they are of the same type.
$a != $b	Not equal	**TRUE** if *$a* is not equal to *$b* after type juggling.
$a <> $b	Not equal	**TRUE** if *$a* is not equal to *$b* after type juggling.
$a !== $b	Not identical	**TRUE** if *$a* is not equal to *$b*, or they are not of the same type.
$a < $b	Less than	**TRUE** if *$a* is strictly less than *$b*.
$a > $b	Greater than	**TRUE** if *$a* is strictly greater than *$b*.
$a <= $b	Less than or equal to	**TRUE** if *$a* is less than or equal to *$b*.
$a >= $b	Greater than or equal to	**TRUE** if *$a* is greater than or equal to *$b*.
$a <=> $b	Spaceship	An integer less than, equal to, or greater than zero when *$a* is respectively less than, equal to, or greater than *$b*. Available as of PHP 7.

Praticando um pouco

```php
<?php
// Integers
echo 1 <=> 1; // 0
echo 1 <=> 2; // -1
echo 2 <=> 1; // 1

// Floats
echo 1.5 <=> 1.5; // 0
echo 1.5 <=> 2.5; // -1
echo 2.5 <=> 1.5; // 1

// Strings
echo "a" <=> "a"; // 0
echo "a" <=> "b"; // -1
echo "b" <=> "a"; // 1

echo "a" <=> "aa"; // -1
echo "zz" <=> "aa"; // 1

// Arrays
echo [] <=> []; // 0
echo [1, 2, 3] <=> [1, 2, 3]; // 0
echo [1, 2, 3] <=> []; // 1
echo [1, 2, 3] <=> [1, 2, 1]; // 1
echo [1, 2, 3] <=> [1, 2, 4]; // -1
```

```php
// Objects
$a = (object) ["a" => "b"];
$b = (object) ["a" => "b"];
echo $a <=> $b; // 0

$a = (object) ["a" => "b"];
$b = (object) ["a" => "c"];
echo $a <=> $b; // -1

$a = (object) ["a" => "c"];
$b = (object) ["a" => "b"];
echo $a <=> $b; // 1

// not only values are compared; keys must match
$a = (object) ["a" => "b"];
$b = (object) ["b" => "b"];
echo $a <=> $b; // 1

?>
```

Operador de controle de erros

Usa-se o operador @ para omitir erros, ele deve ser colocado antes da expressão.

```
<?php

$my_file = @file ('non_existent_file') or
    die ("Failed opening file: error was '$php_errormsg'");
?>
```

Porém para a mensagem de erro funcionar a diretiva track_errors tem que estar On (track_errors = On)

Operador de execução

O PHP suporta um operador de execução: acentos graves (``). Note que não são apóstrofes! O PHP tentará executar o conteúdo dos acentos graves como um comando do shell; a saída será retornada (isto é, ela não será simplesmente descarregada para a saída; ela pode ser atribuída a uma variável). A utilização do operador contra-apóstrofo é idêntica a função shell_exec().

```
<?php
$output = `dir`;
echo "<pre>$output</pre>";
?>
```

Nota:
O operador de execução fica desabilitado quando safe mode está ativo ou shell_exec() está desabilitado.

Operadores de Incremento/Decremento

O PHP suporta operadores de pré e pós-incremento e decremento no estilo C.

Nota: 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 um, e então retorna $a.
$a++	Pós-incremento	Retorna $a, e então incrementa $a em um.
--$a	Pré-decremento	Decrementa $a em um, e então retorna $a.
$a--	Pós-decremento	Retorna $a, e então decrementa $a em um.

Aqui está um script de exemplo simples:

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

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

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

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

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.

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
$a = "Olá ";
$b = $a . "mundo!"; // agora $b contém "Olá mundo!"

$a = "Olá ";
$a .= "mundo!";     // agora $a contém "Olá mundo!"
?>
```

Operadores de array

Operadores de array		
Exemplo	Nome	Resultado
$a + $b	União	União de $a e $b.
$a == $b	Igualdade	**TRUE** se $a e $b tem os mesmos pares de chave/valor.
$a === $b	Identidade	**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 não é igual a $b.
$a <> $b	Desigualdade	**TRUE** se $a não é igual a $b.
$a !== $b	Não identi-dade	**TRUE** se $a não é identico a $b.

```
<?php

$a = array("a" => "maçã", "b" => "banana");
$b = array("a" =>"pêra", "b" => "framboesa", "c" => "morango");

$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);
?>
```

Quando executado, o script produz uma saída assim:

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"
}
```

Operadores de tipo

instanceof é usado para determinar se um variável do PHP é uma objeto instânciado de uma certa classe:

Exemplo #1 Usando instanceof com classes

```
<?php
class MyClass
```

```
{
}

class NotMyClass
{
}
$a = new MyClass;

var_dump($a instanceof MyClass);
var_dump($a instanceof NotMyClass);
?>
```

O exemplo acima irá imprimir:

```
bool(true)
bool(false)
```

instanceof pode também ser usado para determinar se uma variável é um objeto instânciado de uma classe que herda de uma classe pai:

Exemplo #2 Usando instanceof com herança

```
<?php
class ParentClass
{
}

class MyClass extends ParentClass
{
}

$a = new MyClass;

var_dump($a instanceof MyClass); // bool(true)
var_dump($a instanceof ParentClass); // bool(true)
```

Coisas legais em PHP
Estrutura de controle

Por: Hábner Nascimento
habner@treinamentoinfoco.com.br

Daqui para frente veremos que todo o fluxo do sistema será condicionado, dessa forma basta entender apenas que, por padrão para algo acontecer ele deve ser verdadeiro.

IF

O if é mais importante em muitas linguagens, inclusive no PHP

```
<?php
        if (<condição>)
                echo 'esse código será executado';
```

Nota: O uso de chaves é aconselhado, mas, dessa forma também funciona, lembre-se que apenas a próxima linha será executada como parte da expressão do if, sendo assim a partir da segunda linha será executado independente do if.

```
<?php
        if (<condição>)
                echo 'esse código será executado';
        echo 'esse código não faz parte do if';
```

IF ELSE

Sempre que tivermos apenas duas condições poderemos usar if...else, caso a expressão não satisfaça o if será usado o else. Por exemplo para saber um sexo

```
if ($sexo == 'M') {
        echo 'Masculino';
} else {
        echo 'Feminino';
}
```

ELSEIF / ELSE IF

Usa-se quando tivermos mais de uma opção para expressão, exemplo

```
if ($idade <= 18) {
        echo 'Adolescente';
} elseif ($idade > 18 && $idade < 65) {
        echo 'Adulto';
} eles {
        echo 'Idoso';
}
```

(SINTAXE ALTERNATIVA)

Sempre veremos mais de uma forma de se fazer as coisas no PHP, a exemplo disso é a sintaxe alternativa que ela oferece.

O que muda é apenas as chaves de abertura que são substituidas por “:” e as chaves de fechamento são substituidas por endif; endwhile; endfor; endforeach; endswitch;

WHILE

Como disse anteriormente, para que as condições ocorram bastam apenas ser verdadeiras, observe

```
$a=0;
while ($a <= 7) {
        echo $a++;
}
```

o que acontece se fizermos

```
while (true) {
        echo 'aqui';
}
```

e agora

```
while (true) {
        echo 'aqui';
        break;
}
```

DO...WHILE

Essa estrutura foge à regra, pois ele primeiro executa o bloco para depois verificar a condição.

FOR

Essa é uma das estruturas que mais gosto, pois além de receber 3 parâmetros, todos são opcionais, vejamos

Exemplo de funcionamento

```
for (<executa apenas no inicio>; <condicao>; <executa após cada loop>)

for ($a=0 ;$a<10 ; ++$a) {
        echo $a.'<br>';
}

$a=0;
for ( ;  ; ) {
        if ($a>10) break;
        echo $a.'<br>';
        ++$a;
}
```

FOREACH

Usado para percorrer **arrays** e **objetos**.

```
foreach (array_expression as $value)
   statement
foreach (array_expression as $key => $value)
   statement
```

SWITCH

Substitui muitos elseif, pois testa-se a mesma variável sem precisar repeti-la

```
$s = 'M';
switch ($s) {
        case 'M':
                echo 'Masculino';
        break;
        case 'F':
                echo 'Feminino';
        break;
        default:
                echo 'Sexo inválido';
}
```

BREAK

CONTINUE

Coisas legais em PHP
Função

Por: Hábner Nascimento
habner@treinamentoinfoco.com.br

Funções são pequenos trechos de código que possivelmente se repitiria, então a ideia foi colocar tudo em um lugar onde caso seja necessário sofrer alteração que seja em um único lugar, função ganha o nome de método quando falamos de orientação a objeto. Você pode definir suas próprias funções

```
function minha_funcao ($arg_1, $arg_2, /* ..., */ $arg_n) {
        $valor_retornado = false;
        echo "Exemplo de função.\n";
        return $valor_retornado;
}
```

Os valores podem ser passados às funções através de argumentos, veremos aqui por enquanto apenas passagem de parâmetro por valor, veja o exemplo:

```
function minha_funcao ($arg_1, $arg_2) {
        return $arg_1.' - '.$arg_2;
}
echo minha_funcao(1, 9); // result 1 – 9
```

```
function minha_funcao ($arg_1, $arg_2) {
        for ($x=0; $x<func_num_args(); $x++) {
                echo func_get_arg($x).' - ';
        }
}
minha_funcao(1, 9, 7, 5, 4); // result 1 - 9 - 7 - 5 - 4 -
```

ATENÇÃO: você pode usar func_get_args(); // retorna um array com todos os argumentos, fazemos isso como exercício!

Argumentos passados por referência

Por padrão eles são passados por valor, caso coloque o & na frente do parâmetro, significa que o mesmo está apontando para variável passada como argumento

```
<?php
function add_some_extra(&$string)
{
   $string .= ' e alguma coisa mais.';
}
$str = 'Isto é uma string,';
add_some_extra($str);
echo $str;    // imprime 'Isto é uma string, e alguma coisa mais.'
?>
```

Valores padrão de argumentos

```
<?php
function cafeteira ($tipo = "cappuccino")
{
   return "Fazendo uma xícara de café $tipo.\n";
}
echo cafeteira ();
echo cafeteira ("expresso");
```

Coisas legais em PHP
Falando de string

Por: Hábner Nascimento
habner@treinamentoinfoco.com.br

O usuário é a pessoa que quer ver a destruição do seu sistema, sempre pense nisso. Qual a maneira de prevenir que o usuário destrua seu trabalho: - filtrando todas as entradas de dados e configurando corretamente o PHP para que o número de erros sejam os menores possíveis.

Uma das formas de “quebrar” o sistema é colocar aspas na entrada dos dados, por exemplo:

na entrada de um sistema esperamos sempre os dados corretos, mas se os dados não forem o que esperávamos ser ? Como por exemplo no login ao invés de colocar “habner” colocar “h'abner” o que será que aconteceria?

```
$login  = “h'abner”;
$senha = “senha”;
```

mysql_query(“select * from usuario where login='$login' and senha='$senha' ”**)**;

select * from usuario where login='h'abner' and senha='senha' ”

Esse apóstofro(ou aspas simples) ocasionaria um erro de sql, isso é perigoso por que pode expor a sentença sql completa e consequentemente o nome da tabela e dos campos da tabela que está sendo pesquisada. Como o PHP nos ajuda a resolver isso ? Colocando o caractere de escape (\ barra) através da função **addslashes**. Vejamos;

mysql_query(**addslashes(**“select * from usuario where login='$login' and senha='$senha' ”**)**);

select * from usuario where login='h\'abner' and senha='senha' ”

A função **addslashes** adiciona barra invertida (\) antes dos caracteres aspas

simples, aspas duplas e a própria barra invertida, porém, se o parâmetro **magic_quotes_sybase** estiver ativado no php.ini (magic_quotes_gpc = On) em vez de barra invertida, serão inseridas aspas simples sempre que o caracter for também aspas simples.

Caracteres ASCII, a função **chr** recebe um parâmetro de 0 a 255 e retorna o alfabeto em minúsculo. A função **ord** faz o inverso da função **chr** recebe uma letra e retorna o caracter ASCII correspondente.

```
<?php
$principal = 'Treinamento IN FOCO - PHP Zend';
$total      = strlen($principal);
echo $principal.'<br />';

for ($x=0; $x < $total; $x++) {
        echo chr(ord($principal[$x])).'('.ord($principal[$x]).') ';
}
// SAÍDA
Treinamento IN FOCO - PHP Zend
T(84) r(114) e(101) i(105) n(110) a(97) m(109) e(101) n(110) t(116) o(111) (32) I(73) N(78) (32) F(70) O(79) C(67) O(79) (32) -(45) (32) P(80) H(72) P(80) (32) Z(90) e(101) n(110) d(100)
```

Formatando um número para valor monetário, a função **number_format** pode receber um, dois ou quatro parâmetros mas nunca três, da seguinte forma:

Com um parâmetro:

```
<?php
$valor = 50099;
echo number_format($valor);
// a saída será 50,099 lembrando que essa vírgula é a configuração do separador de milhar definida pelo ambiente(a função localeconv contém formatação numérica e monetária local).
```

Com dois parâmetros:

```
<?php
$valor = 50099;
echo number_format($valor, 2);
// a saída será 50,099.00 lembrando que a vírgula e o ponto são configurações do ambiente, o segundo parâmetro é a quantidade de casas decimais que o número formatado vai conter.
```

Com quatro parâmetros:

```
<?php
$valor = 50099;
echo number_format($valor, 2, ',', '.');
// a saída será 50.099,00 o terceiro parâmetro é o separador de decimal, e o quarto parâmetro é o separador de milhar.
```

Complementando uma string ou qualquer dado que possa ser convertido em string com outra string, a função **str_pad** aceita até quatro parâmetros sendo que os dois primeiros são obrigatórios, ela pode acrescentar uma string no início, final ou ambos os lados de uma string.

```
<?php
$nome = 'PHP';
$nome = str_pad($nome, 4);
// caso apenas os dois parâmetros sejam definidos a função acrescenta espaço a direita

$nome = str_pad($nome, 8, 'Zend');// será adicionado a direita a palavra Zend
$nome = str_pad($nome, 30, 'Treinamento IN FOCO - ', STR_PAD_LEFT);
var_dump($nome);
// será adicionado a esquerda a palavra 'Treinamento IN FOCO - '.
```

O quarto parâmetro pode ser:

STR_PAD_LEFT – adiciona o conteúdo a esquerda
STR_PAD_RIGHT – adiciona o conteúdo a direita
STR_PAD_BOTH – adiciona o conteúdo em ambos os lados

caso esse quarto parâmetro não seja informado o padrão é STR_PAD_RIGHT.

Resultado do script acima:

string(30) "Treinamento IN FOCO - PHP Zend"

Mecanizando o que é manual e chato, **str_repeat**(texto, número_repetições), ao invés de colocarmos manualmente um mesmo caracter diversas vezes a ponto de perdermos a conta, essa função nos ajuda com isso.

```
<?php
$cinquenta_espacos = str_repeat(' ', 50); // o que quero que repita, e quantas vezes.
var_dump($cinquenta_espacos);// saída: string(50) " "
```

Localizando e substituindo, a função **str_replace** tem os três parâmetros obrigatórios que são: string pesquisada, string a substituir e string principal;

```
<?php
$principal = 'Treinamento PHP Zend';
var_dump(str_replace('Treinamento', 'Treinamento IN FOCO ', $principal));
// SAÍDA
string(29) "Treinamento IN FOCO PHP Zend"

$principal = array('php Zend', 'php & php');
var_dump(str_replace('php', 'PHP', $principal));
// SAÍDA
array(2) { [0]=> string(8) "PHP Zend" [1]=> string(9) "PHP & PHP" }

$principal = array('php Zend', 'php & php');
var_dump(str_replace(array('php', '&'), 'PHP', $principal));
// SAÍDA
array(2) { [0]=> string(8) "PHP Zend" [1]=> string(11) "PHP PHP PHP" }
```

Embaralhando tudo aleatóriamente, a função **str_shuffle** embaralha aleatóriamente a string informada(ou um tipo que possa ser convertido em string), interessante para senhas com o próprio login do usuário(mas misturando também com números e caracteres especiais =)).

```
<?php
$principal = 'PHP Zend';
var_dump(str_shuffle($principal));// saída string(8) "edPZHn P"
```

Contando as palavras de um texto, **str_word_count** conta a quantidade de palavras de um texto, retorna um array começando do índice zero onde o valor de cada elemento é uma palavra do texto, retorna um array associativo onde os índices são as posições das palavras no texto e o valor de cada elemento é uma palavra do texto;

```
<?php
$principal = 'Treinamento IN FOCO - php Zend';

var_dump(str_word_count($principal));// saida: int(6)
var_dump(str_word_count($principal, 1));
// saida:
        array(6) {
          [0]=>  string(11) "Treinamento"
          [1]=>  string(2) "IN"
          [2]=>  string(4) "FOCO"
          [3]=>  string(1) "-"
          [4]=>  string(3) "php"
          [5]=>  string(4) "Zend"
        }

var_dump(str_word_count($principal, 2));
```

```
// saida:
        array(6) {
          [0]=>  string(11) "Treinamento"
          [12]=>  string(2) "IN"
          [15]=>  string(4) "FOCO"
          [20]=>  string(1) "-"
          [22]=>  string(3) "php"
          [26]=>  string(4) "Zend"
        }
```

"**R**etirando" código malicioso de um texto, **strip_tags** essa função tenta retirar todo código HTML e PHP de um texto, porém com a opção de não retirar as tags que você quiser;

```
<?php
$principal = '<html><b>Treinamento IN FOCO - php Zend</b></html>';
var_dump(strip_tags($principal));
// SAÍDA
string(30) "Treinamento IN FOCO - php Zend"

var_dump(strip_tags($principal, '<b>'));
// SAÍDA
string(37) "<b>Treinamento IN FOCO - php Zend</b>"
```

Entendido **?**

Pegando o tamanho de uma string(ou um tipo que possa ser convertido em string), **strlen** possui apenas um parâmetro e retorna a quantidade de bytes que é o tamanho do valor informado.

```
<?php
$principal = 9;
var_dump(strlen($principal)); // produz a saída: int(1)
```

***** você vai notar que existem funções com underscore(o famoso underline) e funções que se escrevem tudo junto. Enfim voltemos ao resumo...

Retornando a posição de um valor no texto, a função **strpos** retorna a primeira posição de um determinado texto dentro de outro texto, caso o texto pesquisado não seja encontrado a função retorna **false** e caso o texto encontrado esteja no início retorna **zero** o que numa comparação você tem que usar idêntico e não igual para verificar se o texto foi encontrado**;**

```
<?php
$principal = 'php Zend';
var_dump(strpos($principal, 'php')); // saída: int(0)
```

```
$principal = 'php Zend';
var_dump(strpos($principal, 'phpp')); // saída: bool(false)
```

Diferença entre os operadores de comparação: igualdade e idêntico

Igualdade:

$valor = = $a; // retorna verdadeiro se $valor for igual a $a.

Idêntico:

$valor = = = $a; // retorna verdadeiro se $valor for igual a $a e o tipo também.

Estou escrevendo isso por que na conversão para booleano os seguintes valores serão convertidos em false:

- inteiro igual a 0 (zero)
- ponto flutuante igual a 0.0
- string vazia (“”)
- string contendo o valor 0 (zero)
- array vazio (sem algum elemento)
- objeto vazio
- null (nulo)

Lembrando que todas as condições são booleanas...

Invertendo uma string, nunca imaginaria que existisse uma função assim, não pelo nome mas pela utilidade, a finalidade da função **strrev** é retornar a string informada invertida (espelhada).

```
<?php
$principal = 'Treinamento IN FOCO - php Zend';
var_dump(strrev($principal));
// SAÍDA – é um espelho literalmente.
string(30) "dneZ php - OCOF NI otnemanierT"
```

Maiúsculo e minúsculo, as funções **strtoupper** e **strtolower** retornam a string informada maiúscula e minúscula respectivamente...

```
<?php
$principal = 'PHP Zend';
var_dump(strtoupper($principal));
echo '<br />';
var_dump(strtolower($principal));

string(8) "PHP ZEND"
string(8) "php zend"
```

Legal né ?

Retornar pedaços de um texto, “sim isso é possível” com a função **substr** que recebe dois e três parâmetros;

```
<?php
$principal = 'PHP Zend';
var_dump(substr($principal, 4)); // se definido apenas o início, retorna até o final do texto
// saída: string(4) "Zend"

var_dump(substr($principal, 0, 3));
// saída: string(3) "PHP"
var_dump(substr($principal, -4)); // se negativo começa pelo final
// saída: string(4) "Zend"
```

Eliminando dados nas extremidades com a função **trim,** lembre-se que todas as funções até agora não usam referência (&) isso quer dizer que depois que usar a função o texto ou a variável continua do mesmo jeito, apenas o retorno da função é que vem mudado.

```
<?php
$principal = 'PHPPHP Zend';
var_dump(trim($principal, "PHP"));
// SAÍDA
string(5) " Zend"

$principal = ' PHPPHP Zend'; // começa com um espaço
var_dump(trim($principal, "PHP"));
// SAÍDA
string(12) " PHPPHP Zend"

$principal = '  PHP Zend                  ';
var_dump(trim($principal));
// SAÍDA
string(8) "PHP Zend"
```

Lembre-se que retira dados da extremidade, caso o segundo argumento da função não seja passado ele assume o seguinte valor padrão (“ “, “\t”, “\n”, “\r” e “\0”).

Retornar a primeira letra de uma frase maiúscula ou a primeira letra de cada palavra de uma frase em maiúsculo é simples, basta utilizar as funções **ucfirst** e **ucwords** respectivamente;

```
<?php
$principal = 'Treinamento IN FOCO - pHP Zend';
var_dump(ucfirst($principal));
// SAÍDA
string(30) "Treinamento IN FOCO - pHP Zend"

$principal = 'Treinamento IN FOCO - pHP Zend';
var_dump(ucwords($principal));
// SAÍDA
string(30) "Treinamento IN FOCO - PHP Zend"
```

Quebrar um texto ao meio é simples basta apenas utilizar a função **wordwrap** dos quatro parâmetros apenas o primeiro é obrigatório, essa função insere na string informada o caractere de quebra de linha \n (ou um outro caractere) a cada n caracteres (o padrão é 75);

```
<?php
$principal = 'Treinamento IN FOCO - PHP Zend';
var_dump(wordwrap($principal, 5, '<br />'));
// SAÍDA
    string(50) "Treinamento
    IN
    FOCO
    - PHP
    Zend"
```

Código fonte

```
string(50) "Treinamento<br />IN<br />FOCO<br />- PHP<br />Zend"
```

Você observou que a cada 5 caractere mandamos inserir a tag
 (quebra de linha) porém o PHP não quebrou a palavra, antes, respoitou-as no tamanho, porém o quarto parâmetro indica que a quebra deve ser forçada, então;

```
<?php
$principal = 'Treinamento IN FOCO - PHP Zend';
var_dump(wordwrap($principal, 5, '<br />', true));
// SAÍDA
    string(57) "Trein
        ament
        o IN
        FOCO
        - PHP
        Zend"
```

A diferença é que se tiver que quebrar uma palavra ele quebra no meio mesmo.
string(57) "Trein
ament
o IN
FOCO
- PHP
Zend"

Coisas legais em PHP
Falando de array

Por: Hábner Nascimento
habner@treinamentoinfoco.com.br

A forma como o PHP trata array deve ser tratada com minuosidade, pois há alguns detalhes que podem confudir você.

Assim como as variáveis você também não precisa declarar um array (depende das configurações do PHP, se o E_NOTICE não estiver habilitado no error_reporting), mas o que seria um array**?** seria uma coleção de chaves e valores, lembrando que as chaves só podem pertencer ao **grupo escalar**(string, boolean, integer, float) e os valores de um array pode ser qualquer um dos **tipos que o PHP possui**.

Você pode criar um array de duas formas: pelo construtor **array()**

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

// a partir do PHP 5.4
$array = [
    "foo" => "bar",
    "bar" => "foo",
];
?>
```

$arr = array(); // array vazio

$arr = array(1 => array('valor_1', 'valor_2')); // o valor do array pode ser outro array

$arr = array('valor_1', 'valor_2'); // posso colocar apenas valores sem as chaves

Caso você não especifique a chave, o PHP coloca o último inteiro mais um**.**

Adicionalmente, as seguites coerções ocorrerão a chave:

- Strings contendo inteiros válidos, serão convertidos para o tipo inteiro. Por exemplo, a chave *"8"* será, na verdade, armazenada como *8*. Entretanto, *"08"* não será convertido, por não ser um inteiro decimal válido.

- Floats também são convertidos para inteiros, isso significa que a parte fracionada será removida. Por exemplo, a chave *8.7* será na verdade armazenada como *8*.
- Booleanos são convertidos para inteiros, igualmente, por exemplo, a chave *true*, será na verdade armazenada como *1* e a chave *false* como *0*.
- Null será convertido para uma string vazia, por exemplo, a chave *null* na verdade será armazenada como *""*.
- Arrays e objetos *não podem* ser usados como chaves. Fazer isso resultará em um aviso: *Illegal offset type*.

```php
<?php
$array = array(
    1    => "a",
    "1"  => "b",
    1.5  => "c",
    true => "d",
);
var_dump($array);
?>
```

```
array(1) {
  [1]=>
  string(1) "d"
}
```

```
/**
* caso não seja informado um valor para chave, o PHP busca o maior índice do array e
* soma mais 1, se o array estiver sendo criado ou até então tiver apenas
* índices associativos o índice máximo será -1, desta forma o primeiro índice será 0 (zero)
*/

        $arr[]   = 'valor_1'; // índice zero
        $arr[]   = 'valor_2'; // índice um
        $arr[1] = array('valor_1', 'valor_2'); // sobrescreve o índice um

        Somando dois arrays, com o operador +

        <?php
        $arr_1 = array(4 => 'treinamento', 'IN FOCO');
        $arr_2 = array(5 => 'treinamento', 'PHP', 'Zend');
        var_dump($arr_1 + $arr_2);

        // SAÍDA
        array(4) {
                [4]=>  string(11) "treinamento"
                [5]=>  string(7) "IN FOCO"
                [6]=>  string(3) "PHP"
                [7]=>  string(4) "Zend" }
```

* se os valores se repitirem, será considerado apenas a chave e o valor do primeiro array.

Combinando dois arrays com a função **array_combine,** a partir de dois arrays se cria um, onde o valor do primeiro array será o índice e o valor do segundo array será o valor desse novo array, mas não se esqueça que os dois arrays tem que ter o mesmo tamanho, caso contrário a função retornará **false;**

```
<?php
$arr_1 = array('Instituição', 'Curso');
$arr_2 = array('Treinamento IN FOCO', 'PHP Top');
var_dump(array_combine($arr_1, $arr_2));
// SAÍDA
array(2) {
        ['Instituição']=>  string(13) "Treinamento IN FOCO"
        ['Curso']=>  string(5) "PHP Top"
}
```

O PHP facilita o trabalho que seria manual, vejamos; para saber quantas vezes um valor aparece num array utilizamos a função **array_count_values** que retorna um array onde os índices são os valores do array e o valor é a quantidade de vezes que esse valor aparece no array.

```
<?php
$arr_1 = array('Instituição', 'Curso', 'treinamento IN FOCO', 'PHP Zend', 'Curso');
var_dump(array_count_values($arr_1));
// SAÍDA
array(4) {
  ["Instituição"]=>  int(1)
  ["Curso"]=>  int(2)
  ["treinamento IN FOCO"]=>  int(1)
  ["PHP Zend"]=>  int(1)
}
```

Como criar um array de 30 posições com o mesmo valor? Seria:

```
$arr_1 = array('hábner', 'hábner'...); // repetir o valor 'hábner' trinta vezes?
```

A função **array_fill** recebe 3 parâmetros, o primeiro é a posição inicial do índice do array, o segundo é quantas posições devem ser criadas, o terceiro é o valor de todas as posições;

```
<?php
var_dump(array_fill(7, 4, 'hábner'));
// saída
array(5) {
        [7]=>    string(7) "hábner"
        [8]=>    string(7) "hábner"
        [9]=>    string(7) "hábner"
        [10]=>   string(7) "hábner"  }
```

Troca de valores com a função **array_flip** o que é valor torna-se índice e o que for índice torna-se valor, porém os valores tem que ser do grupo escalar (pois o índice só pode ser de um dos tipos do grupo escalar), se tiver mais de um mesmo valor no array, a chave sempre será sobrescrita pelo último valor.

```
<?php
echo '<pre>';
var_dump(array_flip(array('nome' => 'Hábner', 10, 'a' => 10, 5, 8)));
// SAÍDA
array(4) {
        ["Hábner"]=>   string(4) "nome"
        [10]=>  string(1) "a"
        [5]=>    int(1)
        [8]=>    int(2)
}
```

Verificando se a chave de um array existe com a função **array_key_exists** ainda bem que o nome da maioria das funções são sugestivas, essa função retorna um booleano (**true** se encontrou e **false** se não encontrou) essa função serve para você saber se um índice existe antes de inserir outro

```
<?php
echo '<pre>';
$arr_1 = array('nome' => 'especialista', 'php');
var_dump(array_key_exists(0, $arr_1));
// SAÍDA
bool(true)
```

Retornando todas as chaves ou as chaves de um critério aplicado em um array com a função **array_keys** que recebe dois parâmetros mas apenas o primeiro é obrigatório.

```
<?php
        echo '<pre>';
        $arr_1 = array('nome' => 'especialista', 'php', 'curso' => 'php zend', 'php zend');
        var_dump(array_keys($arr_1));
        // SAÍDA
        array(4) {
          [0]=>  string(4) "nome"
          [1]=>  int(0)
          [2]=>  string(5) "curso"
          [3]=>  int(1)
        }
```

O valor do novo array são todas as chaves do array passado como argumento, e como disse você pode retornar apenas as chaves de todos os valores de um array;

```
<?php
echo '<pre>';
$arr_1 = array('nome' => 'especialista', 'php', 'curso' => 'php zend', 'php zend');
var_dump(array_keys($arr_1, 'php zend'));
// SAÍDA
array(2) {
  [0]=>  string(5) "curso"
  [1]=>  int(1)
}
```

Completando um array com a função **array_pad,** útil para você que deseja completar um array com determinado número de elementos, sendo:

```
<?php
$arr_1 = array('nome' => 'especialista PHP');
var_dump(array_pad($arr_1, 2, 'habner'));
// o primeiro parâmetro é o array, o segundo é o tamanho que o array deve ter, sendo que se o tamanho atual do array for maior do que o parâmetro de tamanho nada será feito, e o terceiro argumento é o valor desse array .
// SAÍDA
array(2) {
        ["nome"]=>     string(16) "especialista PHP"
        [0]=>    string(6) "habner"
}
```

Trabalhando com funções que usam parâmetros referênciais.

Retornando e eliminando o último elemento de um array com a função **array_pop;**

```
<?php
$arr_1 = array('nome' => 'especialista PHP', 'curso', 'dia de semana');
array_pop($arr_1); // a função retira e retorna o valor do último elemento do array
var_dump($arr_1);
// SAÍDA
array(2) {
  ["nome"]=>
        string(16) "especialista PHP"
  [0]=>
        string(5) "curso"
}
```

Inserindo elementos no final de um array com a função **array_push**, além de inserir elementos no final de um array ele retorna a quantidade de elementos após a inserção.

```
<?php
  echo '<pre>';
  $arr_1 = array('nome' => 'especialista PHP', 'curso', 'dia de semana');
  echo array_push($arr_1, array('tenho que', 'estudar mais'));
  echo '<br />';
  var_dump($arr_1);
  // SAÍDA
4
array(4) {
  ["nome"]=>
        string(16) "especialista PHP"
    [0]=>
        string(5) "curso"
  [1]=>
        string(13) "dia de semana"
  [2]=>
         array(2) {
               [0]=>
                  string(9) "tenho que"
            [1]=>
                  string(12) "estudar mais"
        }
}
```

Retirando elementos do início do array com a função **array_shift**, além se inserir elementos no início do array essa função retorna o valor do elemento removido; O detalhe dessa função é que os elementos de índices numéricos serão reordenados

```
<?php
  echo '<pre>';
  $arr_1 = array('nome' => 'especialista PHP', 'curso', 'dia de semana');
  var_dump(array_shift($arr_1));
  echo '<br />';
  var_dump($arr_1);
  // SAÍDA
  string(16) "especialista PHP"
  array(2) {
         [0]=>
                string(5) "curso"
         [1]=>
                string(13) "dia de semana"
  }
```

Inserindo elementos no início de um array com a função **array_unshift**, essa função insere elementos no início e retorna o número de elementos após a inserção.

```
<?php
  echo '<pre>';
  $arr_1 = array('nome' => 'especialista PHP', 'curso', 'dia de semana');
  var_dump(array_unshift($arr_1,  'tenho que', 'estudar mais'));
  echo '<br />';
  var_dump($arr_1);
  // SAÍDA
  int(5)
array(5) {
  [0]=>
        string(9) "tenho que"
  [1]=>
        string(12) "estudar mais"
  ["nome"]=>
        string(16) "especialista PHP"
  [2]=>
        string(5) "curso"
  [3]=>
        string(13) "dia de semana"
}
```

Ordenando um array, as funções abaixo também são passadas por referência;

Ao todo são 6 funções, das quais todas que tem a letra **"r"** são para colocar o array em ordem decrescente;

FORMA ASCENDENTE:

Ordenando pelo valor e desprezando os índices;
sort($arr_1);
Ordenando pelo valor e mantendo os índices;
asort($arr_1);
Ordenando pela chave;
ksort($qrr_1);

Ordenando decrescente:
rsort($arr_1);
a**r**sort($arr_1);
k**r**sort($arr_1);

Pesquisando um valor e retornando o índice com a função **array_search**, caso o valor pesquisado esteja presente várias vezes num array ele retorna apenas o índice do primeiro valor encontrado, caso o valor pesquisado não esteja no array a função retorna

o booleano **false** (assim como na maioria das funções);

```
<?php
  echo '<pre>';
  $arr_1 = array('nome' => 'especialista PHP', 'curso', 'dia de semana',
'especialista PHP');
  var_dump(array_search('especialista PHP', $arr_1));
  // SAÍDA
  string(4) "nome"
```

Invertendo índices de um array com a função **array_reverse**, essa função retorna um array em ordem inversa, ou seja, o primeiro elemento será o primeiro e o primeiro será o último, quando a função inverte o array ele reordena os índices numéricos fincando assim:

```
<?php
  echo '<pre>';
  $arr_1 = array('nome' => 'especialista PHP', 'curso', 'dia de semana',
'especialista');
  var_dump(array_reverse($arr_1));
  // SAÍDA
  array(4) {
        [0]=>  string(12) "especialista"
        [1]=>  string(13) "dia de semana"
        [2]=>  string(5) "curso"
        ["nome"]=> string(16) "especialista PHP"
  }
```

Mas caso você queira que os elementos mantenham os índices coloque o booleano true no segundo argumento da função, ficando assim:

```
<?php
  echo '<pre>';
  $arr_1 = array('nome' => 'especialista PHP', 'curso', 'dia de semana',
'especialista');
  var_dump(array_reverse($arr_1, true));
  // SAÍDA
  array(4) {
        [2]=>  string(12) "especialista"
        [1]=>  string(13) "dia de semana"
        [0]=>  string(5) "curso"
        ["nome"]=>    string(16) "especialista PHP"
  }
```

Coisas legais em PHP
Manipulação de arquivos

Por: Hábner Nascimento
habner@treinamentoinfoco.com.br

O que sempre digo é que o PHP tem sempre formas e facilidades de se fazer alguma coisa, além dos nomes das funções serem bem sugestivas...

Começo falando que mais ou menos 95% das funções no PHP retorna um booleano e que a maioria das funções são “perguntas” que esperam duas respostas sim ou não, A função **is_dir** verifica se o parâmetro passado é um diretório, ela retorna **true** caso seja e **false** caso não seja.

Criando e deletando diretórios

a função mkdir contém os seguintes parâmetros:

```
// nome do diretório, nível de permissão no linux
mkdir('dir_1', 0777);
```

caso queira criar 3 diretórios um dentro do outro como faríamos**?**

```
mkdir('dir_1', 0777);
mkdir('dir_1/dir_2', 0777);
mkdir('dir_1/dir_2/dir_3', 0777);
```

```
mkdir('dir_1/dir_2/dir_3', 0777, true);
```

* o terceiro parâmetro da função sinaliza que os diretórios serão criados recursivamente ou seja você não precisa

Vamos falar sobre os mais utilizados na maioria das aplicações.

- basename — Retorna apenas a parte que corresponde ao nome do arquivo de um caminho/path
- chmod — Modifica as permissões do arquivo
- clearstatcache — Limpa as informações em cache sobre arquivos
- copy — Copia arquivo
- delete — Veja unlink ou unset
- dirname — Retorna o caminho/path do diretório pai
- fclose — Fecha um ponteiro de arquivo aberto
- feof — Testa pelo fim-de-arquivo (eof) em um ponteiro de arquivo
- fgets — Lê uma linha de um ponteiro de arquivo

- fgetss — Ler uma linha de um ponteiro de arquivo e retira as tags HTML
- file_exists — Verifica se um arquivo ou diretório existe
- file_get_contents — Lê todo o conteúdo de um arquivo para uma string
- file_put_contents — Escreve uma string para um arquivo
- file — Lê todo o arquivo para um array
- filesize — Obtém o tamanho do arquivo
- fopen — Abre um arquivo ou URL
- fputs — Sinônimo de fwrite
- fread — Leitura binary-safe de arquivo
- fwrite — Escrita binary-safe em arquivos
- glob — Acha caminhos que combinam com um padrão
- is_dir — Diz se o caminho é um diretório
- is_file — Informa se o arquivo é um arquivo comum
- is_readable — Diz se o arquivo existe e se ele pode ser lido
- is_uploaded_file — Diz se o arquivo foi enviado por POST HTTP
- is_writable — Diz se o arquivo pode ser modificado
- is_writeable — Sinônimo de is_writable
- mkdir — Cria um diretório
- move_uploaded_file — Move um arquivo enviado para uma nova localização
- readfile — Lê e exibe o conteúdo de um arquivo
- readlink — Retornar o alvo de um link simbólico
- realpath — Retorna o path absoluto canonicalizado
- rename — Renomeia um arquivo ou diretório
- rewind — Reinicializa a posição do ponteiro de arquivos para o início
- rmdir — Remove um diretório
- tempnam — Cria um nome de arquivo único
- tmpfile — Cria um arquivo temporário
- unlink — Apaga um arquivo

Coisas legais em PHP
Falando de orientação a objetos – básico

Por: Hábner Nascimento
habner@treinamentoinfoco.com.br

Esqueça um pouco computador e volte ao mundo real... O que seria um objeto? Algo existente? Algo paupável? Você poderia me dar um exemplo de objeto?

Objeto: mamadeira

Ok, precisamos definir apenas duas coisas que "todo" objeto tem, o que a mamadeira tem e o que ela faz.

// o que a mamadeira tem (características) chamamos de propriedade
Propriedades de uma mamadeira
- Bico
- ML

// o que a mamadeira faz chamamos de métodos
Métodos de uma mamadeira
- Gira bico
- Puxa bico
- Empurra bico

Simples não? O que acabamos de fazer foi definir uma classe, classe é um agrupado de propriedades e métodos pertencentes a um objeto. Cada arquivo PHP deve ter apenas uma classe (boa prática, por questão do autoload e manutenção).

Um objeto pode ser qualquer coisa que tem características e/ou ações.

Até agora trabalhamos com funções dentro de arquivos e tal, agora acompanha meu raciocínio: - você tem funções com o mesmo propósito, vejamos:

cadastrarCliente().... editarCliente().... excluirCliente().... e pesquisarCliente().....

Em termos de organização era bom que todas essas funções já que tem um mesmo propósito (manipular dados de cliente) ficassem em um único arquivo, ou seja, colocar isoladamente em um arquivo apenas funções referentes a cliente.

Orientação a objetos é organização, pense em orientação a objeto como agrupamento de funções para um mesmo propósito, como a manipulação da informação do cliente. O que outrora estaria assim:

```
<?php
	function cadastrarCliente() {
		[...]
	}

	function editarCliente() {
		[...]
	}

	function excluirCliente() {
		[...]
	}

	function pesquisarCliente() {
		[...]
	}
```

E agora está assim:

```
<?php
class Cliente
{
	function cadastrarCliente()
	{
		[...]
	}

	function editarCliente()
	{
		[...]
	}

	function excluirCliente()
	{
		[...]
	}

	function pesquisarCliente()
	{
		[...]
	}
}
	// *A partir de agora chamaremos as funções de métodos(caso este esteja dentro de uma classe).
```

Vamos mas a fundo,

/** note que não fechamos a tag php no final do arquivo (**?>**), pois é proposto que nunca feche a tag php para arquivos que contenham apenas código php, para prevenir espaços em branco na saida para o browser isso pode ser perigoso caso tenhamos um **header('location : ….php');** mais a frente, por isso sempre colocamos um **die;** logo após. */

Coisas legais em PHP
Falando de PDO (PHP Data Object)

Por: Hábner Nascimento
habner@treinamentoinfoco.com.br

PDO – PHP Data Object

PDO é uma abstração de acesso a banco de dados orientado a objetos(o que o torna mais flexível e documentado) e o melhor de tudo é que essa biblioteca já vem integrada no php, ou, no caso do linux basta adicionar essa biblioteca(pode ser através do Synaptic), para habilita-la basta descomentar a linha da biblioteca no php.ini.

Atualmente você pode utilizar dos mais variados bancos de dados da seguinte forma:

```
mysql_connect
        *_query
[…],

mssql_connect
        *_query
[…],

pgsql_connect
        *_query
[…],
```

Observe que sempre antes de qualquer comando você tem que definir o sgbd que será utilizado em cada comando, enfim, temos uma infinidade de comandos para um sgbd específico, o "problema" que nos deparamos em muitos sistemas é que nunca pensamos que o mesmo pode crescer, e dependendo desse crescimento será necessário mudar de banco de dados, por questão de segurança, suporte, comodidade e integridade dos dados. Imagine agora que todo o seu sistema esteja usando o sgbd mysql, e de repente tenhamos que utilizar outro sgbd, agora você entende que temos um problema? Pois todos os cantos do nosso sistema que esteja executando algo para um banco de dados específico temos que modificar para o novo banco de dados:

o que era:

```
mysql_connect
mysql_close
mysql_fetch_array
mysql_escape_string
```

mysql_fetch_assoc
mysql_fetch_field
mysql_fetch_row
mysql_query
mysql_select_db …. dentre muitas outras(http://br.php.net/manual/en/book.mysql.php)

agora terá que ser:
pg_connect
pg_close
pg_escape_string
pg_fetch_all
pg_fetch_array
pg_fetch_assoc
pg_fetch_row
pg_num_rows

* Notamos que o que muda de um sgbd para outro é apenas o prefixo, vamos resolver esse grande problema, da seguinte forma.

Vamos garantir a flexibilidade do nosso sistema usando o PDO:

```
<?php
	$conexao = new PDO('mysql:dbname=testdb;host=127.0.0.1', 'root', 'root');
```

A parte mais difícil vem agora..., no código acima estamos conectados com que banco de dados? Isso mesmo o mysql, só que agora todo o sistema vai usar postgreSql e agora...

```
<?php
	$conexao = new PDO("pgsql:dbname=testdb;host=127.0.0.1", 'root', 'root');
```

Note que o que mudou apenas foi o driver de conexão, que elegante não..., isso garante que seu sistema use sem problemas as funções do pdo.
E se você quiser utilizar qualquer comando basta colocar:

```
$sql = 'SELECT name, color, calories FROM fruit ORDER BY name';
/**
*Esquecendo aqui do prefixo, pois o drive foi definido na conexão.
*/
foreach ($conexao->query($sql) as $row) {
	print $row['name'] . "\t";
	print $row['color'] . "\t";
	print $row['calories'] . "\n";
}
```

Perceba que agora codamos assim:

[objeto pdo] → comando

que é o mesmo de:

$conexao->query

e não mais

mysql_query

pgsql_query

Pois o driver já foi definido no construtor..., esse arquivo aqui só é uma base, para você aprender o resto basta olhar no php.net, mas atenção:

Em http://br.php.net/pdo encontramos duas partes:

PDO — The PDO class [comandos]

e

PDOStatement — The PDOStatement class

A opção de PDOStatement só pode ser chamada por um resultado de um método que está alocado na parte do PDO.

Exemplo:

```
<?php
	$sth = $dbh->prepare("SELECT name, colour FROM fruit");
	$sth->execute();

	/* Fetch all of the remaining rows in the result set */
	print("Fetch all of the remaining rows in the result set:\n");
	$result = $sth->fetchAll();
	print_r($result);
```

Referências:

http://br.php.net/pdo

http://php.net/manual/en/ref.pdo-pgsql.php

Dúvidas basta acessar: http://habnernascimento.blogspot.com e comentar sobre o assunto.