PHP

Luiz Felipe Ferreira Mai

SUMÁRIO

Por onde começar?

PARTE 1

Introdução

Introdução

- Criada em 1994 por Rasmus Lerdorf, que escreveu um conjunto de binários em C utilizados em sua página pessoal.
- Expandido para trabalhar com formulários web e comunicar-se com banco de dados.
- Inicialmente chamado PHP/FI (Personal Home Page/Forms Interpreter)
- Lançado ao público com o nome PHP Tools (Personal Home Page Tools) para agilizar a detecção de possíveis bugs.
- Criada “por necessidade”.

Introdução

“Eu não sei como parar isso. Eu nunca tive a intenção de escrever uma linguagem de programação...”

“Eu não faço ideia de como escrever uma linguagem de programação, eu apenas continuei adicionando o próximo passo lógico.”

LERDORF, Rasmus

Introdução

- Hypertext Preprocessor
- Open Source
- Desenvolvimento Web
- Embutida em HTML
- Delimitada por **<?php** e **?>**
- Processada no servidor

```
<!DOCTYPE HTML>
    <head>
        <title>Exemplo</title>
    </head>
    <body>

        <?php
        echo “Olá, eu sou um script PHP!”;
        ?>

    </body>
</html>
```

O que podemos fazer?

- Geração e manipulação de HTML
- Linguagem estruturada ou orientada a objetos
- Geração de PDF, XML e até animações Flash
- Suporte a diversos bancos de dados
- Permite trabalhar com expressões regulares
- Scripts de linhas de comando

VOCÊ SABIA?

Embora pouco difundidas, a linguagem possui uma grande variedade de bibliotecas, varrendo funções desde envio de e-mail a manipulação de imagens.

O que eu preciso?

Você só precisa de um servidor com suporte a PHP :)

(e de um editor de texto, claro...)

Algo um pouco mais útil...

```php
<?php
   if (strpos($_SERVER['HTTP_USER_AGENT'], 'MSIE') !== FALSE) {
?>
      <p>Tsc... Você está usando o Internet Explorer...</p>
<?php
   } else {
?>
      <p>Você não está usando o Internet Explorer :)</p>
<?php
   }
?>
```

PARTE 2

Aspectos básicos da linguagem

Separação de instruções

- Assim como em C, o PHP também requer o uso de **ponto-e-vírgula** ao final de cada instrução.
- A última linha do bloco PHP não requer o ponto-e-vírgula.
- A tag **?>** também pode ser omitida no final dos arquivos .php

```
<?php echo 'Isto é um teste';?>

<?php echo 'Isto é um teste'?>
```

Comentários

- Comentário de linha: **//** ou **#**
- Comentário de bloco: **/*** ... ***/**
- Os efeitos de um bloco de comentário só persistem até o fechamento de um bloco PHP.
- Bloco de documentação: **/**** ... ***/**

VOCÊ SABIA?

O PHP possui a biblioteca PHPDocumentor que, da mesma forma que Java, gera a documentação do código por meio dos blocos **/**** ... ***/**

Comentários

```
<?php

    echo 'Isto é um teste';// Comentário de uma linha

    /* Bloco de comentário
     echo 'Teste 1';
     echo 'Teste 2';    */

    echo 'Um teste final'; # Comentário de uma linha

?>
```

Tipos

- Tipos primitivos:
  - Boolean
  - Integer
  - Float
  - String
  - Array
  - Object
  - Resource
  - NULL

Tipos

- PHP utiliza tipagem dinâmica
  - O tipo da variável é definido em tempo de execução
- PHP é uma linguagem fracamente tipada
  - Conversões de tipo podem ser feitas livremente
- Duas funções importantes:
  - **gettype(var):** retorna o tipo da variável passada como parâmetro.
  - **is_int(var), is_string(var), ...:** retorna true ou false dependendo do tipo da variável.

Tipos - Booleano

- Pode ser **TRUE** ou **FALSE**
- Ambas são *case-insensitive*
- Essencial para estruturas de controle

- São considerados **false**:

- o próprio booleano false
- o inteiro 0
- o float 0.0
- uma string vazia
- a string “0”
- um array sem elementos
- o tipo NULL
- o objeto SimpleXML criado de tags vazias

Tipos - Inteiros

- Números do conjunto Z = {..., -2, -1, 0, 1, 2, ...}.
- Podem ser especificados em base decimal, octal ou hexadecimal.
  - Decimal: representação padrão
  - Octal: precedido por um 0
  - Hexadecimal: precedido por 0x
- O overflow de números inteiros acarreta na conversão para float.

VOCÊ SABIA?

A constante PHP_INT_MAX fornece o maior inteiro suportado no interpretador PHP que está sendo utilizado.

Tipos - Ponto Flutuante

- Podem ser representados de 2 maneiras:
  - $a = 1.234
  - $b = 7.1e5
- O PHP geralmente utiliza o sistema de precisão dupla do IEEE 754.
- Valores não representáveis por ponto flutuante são representados por NaN.

Tipos - String

- PHP não tem suporte nativo a Unicode. Para isso, utiliza as funções **utf8_encode()** e **utf8_decode()**.
- O tamanho máximo de uma string é limitado apenas pela memória do computador.
- Pode ser especificada:
  - **Entre apóstrofos:** permite apenas o escape de apóstrofos e barras invertidas.
  - **Entre aspas:** permite o escape de expressões como \n e \t e interpreta o valor de variáveis diretamente.

Tipos - String

- O operador . permite concatenar strings.
- **strlen:** retorna o tamanho da string.
- **strpos:** encontra a posição da primeira ocorrência de uma substring
- **strtoupper / strtolower:** converte a string para maiúscula / minúscula
- **str_replace:** substitui parte de uma string por outra string
- **substr:** retorna uma parte de uma string
- **explode:** divide uma string em uma array de strings

Tipos - Strings

```php
<?php

   $string = “Aula de LP”;

   $length = strlen($string);
   //$length = 10;

   $new_string = str_replace(“LP”, “EngSoft”, $string);
   //$string agora vale “Aula de EngSoft”

   $array = explode(“ “, $string);
   //$array[0] = “Aula”, $array[1] = “de”, $array[2] = “LP”;

?>
```

Tipos - Array

- Representa um mapeamento, relacionando *chaves* a *valores*.
- Criados pela função array(), que pode, ou não, receber parâmetros.
- As chaves podem ser omitidas no momento da criação do array.
    - Neste caso, as chaves são inteiros ordenados (0, 1, 2, ...)
- Valores podem ser alterados diretamete. Por exemplo, $array[0] = 'PHP';
- A função unset() permite remover um elemento de um array ou todo o array.
- A função foreach() permite percorrer todo o array.

Tipos - Array

```
<?php

   $array = array(0 => ‘banana’, 1 => ‘uva’, 2 => ‘melancia’);

   foreach($array as $fruta){
      echo $fruta;
   }

   unset($array[1]);

   foreach($array as $fruta){
      echo $fruta;
   }
?>
```

Tipos - Objeto

- Novos objetos podem ser criados utilizando **new**.

```
<?php
   class Teste
   {
      function testando()
      {
           echo “Este é um teste”;
       }
   }

   $meu_teste = new Teste;
   $meu_teste->testando();
?>
```

Tipos - Recurso

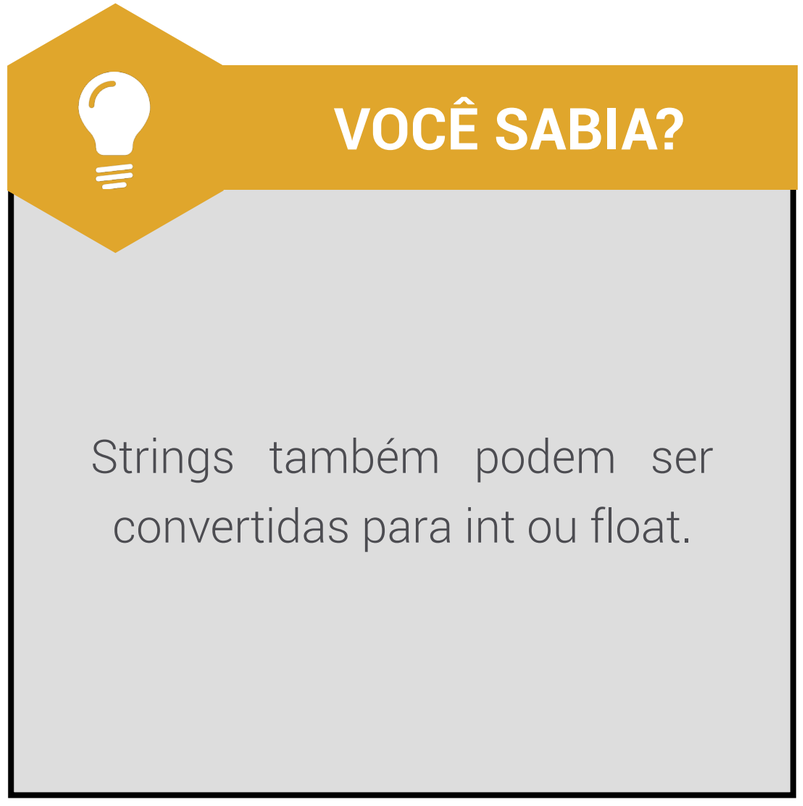
- Recursos são variáveis especiais, que referenciam um recurso externo.
- Quando um recurso deixa de ser utilizado, o coletor de lixo libera a memória automaticamente.
- Exemplos de recursos:
  - Conexões a bancos de dados
  - Conexão FTP
  - Conexão CURL

Tipos - NULL

- O único valor que pode ser associado a esse tipo é NULL.
- A variável é considerada NULL se:
  - a ela foi atribuído o valor NULL
  - se nenhum valor foi atribuído a ela
  - se ela foi apagada com unset()

Conversão de tipos

- O tipo de uma variável é determinado pelo contexto em que é utilizada.
- Conversão explícita pode ser feito por ***casting*** ou por funções como **intval()**
- Em diversos casos, não é necessário explicitar a conversão de tipo.



Conversão de tipos

- (int), (integer) - converte para inteiro
- (bool), (boolean) - converte para booleano
- (float), (double), (real) - converte para número de ponto flutuante
- (string) - converte para string
- (binary) - converte para string binária
- (array) - converte para array
- (object) - converte para objeto

```
<?php

   $foo = “0”;
   // $foo é string

   $foo += 2;
   // $foo é agora um inteiro (2)

   $foo = $foo + 1.3;
   // $foo é agora um float (3.3)

   $foo = 5 + “10 porcos”;
   // $foo é inteiro (15)

?>
```

Variáveis

- Representadas por um **$** seguido do nome da variável
- Formado por letras, algarismos ou sublinhados
- Não pode se iniciar com um algarismo
- Case-sensitive
- Atribuição por valor ou referência

Variáveis - Palavras-chave

__halt_compiler()	abstract	and	array()	as
break	callable (a partir do PHP 5.4)	case	catch	class
clone	const	continue	declare	default
die()	do	echo	else	elseif
empty()	enddeclare	endfor	endforeach	endif
endswitch	endwhile	eval()	exit()	extends
final	finally (a partir do PHP 5.5)	for	foreach	function
global	goto (a partir do PHP 5.3)	if	implements	include
include_once	instanceof	insteadof (a partir do PHP 5.4)	interface	isset()
list()	namespace (a partir do PHP 5.3)	new	or	print
private	protected	public	require	require_once
return	static	switch	throw	trait (a partir do PHP 5.4)
try	unset()	use	var	while
xor	yield (a partir do PHP 5.5)			

Variáveis - Palavras reservadas

int (desde o PHP 7)	float (desde o PHP 7)	bool (desde o PHP 7)	string (desde o PHP 7)
true (desde o PHP 7)	false (desde o PHP 7)	null (desde o PHP 7)	resource (desde o PHP 7)
object (desde o PHP 7)	mixed (desde o PHP 7)	numeric (desde o PHP 7)	

Variáveis - Atribuição por valor

```php
<?php
   $var1 = 10;
   $var2 = $var1;
   $var2 = $var2 + 5;


   //Ao final da execução:
   //var1 = 10;
   //var2 = 15;
?>
```

Variáveis - Atribuição por referência

```php
<?php
   $var1 = 10;
   $var2 = &$var1;
   $var2 = 1000;

   //Ao final da execução:
   //var1 = 1000;
   //var2 = 1000;

   $var3 = &(24 * 7);
   //NÃO PODE
?>
```

Variáveis - Escopo

- Escopo local - variáveis não são vistas fora da função
- Variáveis globais utilizam a keyword **global**
- Variáveis dentro de blocos de seleção e repetição são reconhecidas fora deles
- Permite o uso de variáveis estáticas
- Expressões não podem ser atribuídas a variáveis estáticas. Apenas valores.

```php
<?php
$a = 1;
$b = 2;

function Soma(){
   global $a, $b;
   $b = $b + a;
}

Soma();
echo $b;

?>
```

Variáveis - Escopo

```php
<?php
function Teste()
{
    static $count = 0;

    $count++;
    echo $count;
    if ($count < 10) {
        Teste();
    }
    $count--;
}
?>
```

Variáveis variáveis

- PHP aceita que uma variável tenha seu nome variando de acordo com o valor de outra variável.
- É possível encadear quantas variáveis forem necessárias.

```
<?php
  $Bar = “a”;
  $Foo = “Bar”;
  $World = “Foo”;
  $Hello = “World”;
  $a = “Hello”;

  $a; //Hello
  $$a; //World
  $$$a; //Foo
  $$$$a; //Bar
  $$$$$a; //a
?>
```

Variáveis externas

- PHP é geralmente utilizado na validação de formulários HTML
- Para isso, faz-se necessário obter os valores inseridos nestes formulários
- **GET:** retorna os dados da QUERY_STRING (http://www.example.com/index.php?id=6)
- **POST:** retorna os dados de forma oculta
- Variáveis $_POST e $_GET

Variáveis externas

```
<form action=”foo.php” method=”post”>
    Nome:  <input type=”text” name=”username” /><br />
    Email: <input type=”text” name=”email” /><br />
    <input type=”submit” name=”submit” value=”Enviar” />
</form>



<?php
   echo $_POST[‘username’];
?>
```

Variáveis externas

- Além do $_POST e $_GET, PHP também dá suporte a $_COOKIES e $_SESSION
- Ambos são formas de se armazenar dados de forma mais duradoura.
- Sessões só funcionam se o servidor tiver cookies ativados.
- O tempo de vida de um cookie pode ser especificado na chamada de função
- O tempo de vida padrão de uma sessão é de 24 minutos (ou o tempo em que a janela do navegador permanecer aberta, se inferior a isso).

Constantes

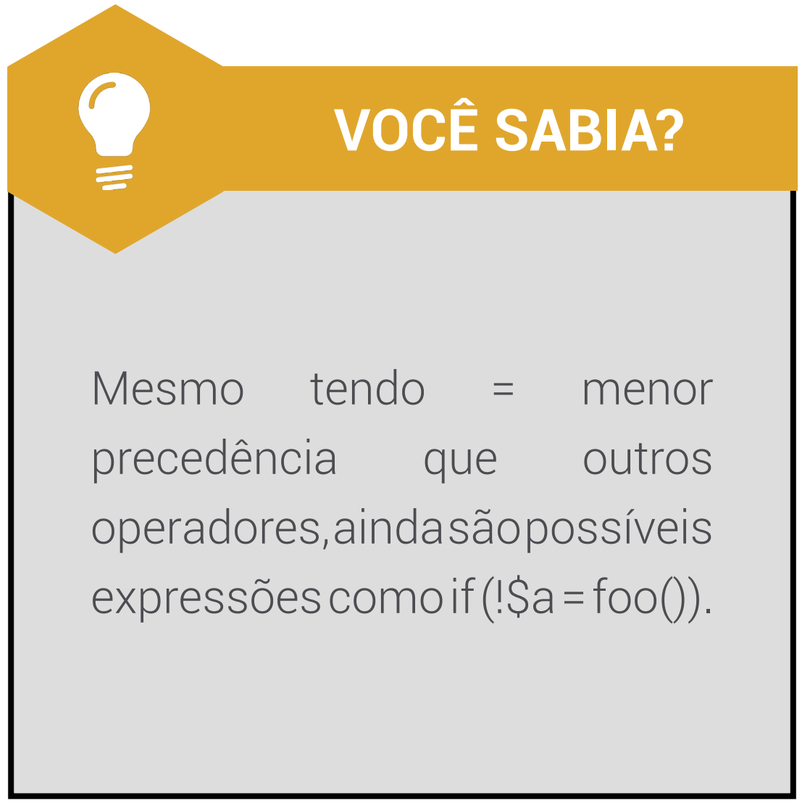
- Devem iniciar com letra ou sublinhado e serem seguidas por letras, números ou sublinhados
- São case sensitive
- Possuem escopo global
- Define-se uma constante utilizando a função define() ou a palavra-chave const
- Quando definidas por const, não suportam conteúdo que precise de processamento do servidor

Constantes

```
$color = “azul”;  //Funciona
const AZUL = “This is the color $color”;   //Não funciona
define(strtoupper($color), “Esta é a cor $color”)  //Funciona


echo AZUL;      //Imprime “Esta é a cor azul”
echo Azul;      //Imprime “Azul” e gera um alerta
```

Operadores - Precedência

- Alguns operadores possuem precedências sobre os demais
- Afeta diretamente o resultado de uma expressão
- Parênteses podem ser utilizados para “forçar" a precedência de uma expressão



Operadores - Precedência

Associação	Operador	Informação adicional
não associativo	clone new	clone e new
esquerda	[	array()
não associativo	++ --	incremento/decremento
não associativo	~ - (int) (float) (string) (array) (object) (bool) @	tipos
não associativo	instanceof	tipos
direita	!	lógico
esquerda	* / %	aritmético
esquerda	+ - .	aritmético e string
esquerda	<< >>	Bit-a-bit
não associativo	< <= > >= <>	comparação
não associativo	== != === !==	comparação

Operadores - Precedência

esquerda	&	Bit-a-bit e referências
esquerda	^	Bit-a-bit
esquerda	|	Bit-a-bit
esquerda	&&	lógico
esquerda	||	lógico
esquerda	? :	ternário
direita	= += -= *= /= .= %= &= |= ^= <<= >>=	atribuição
esquerda	and	lógico
esquerda	xor	lógico
esquerda	or	lógico
esquerda	,	muitos usos

Operadores Aritméticos

- -$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 por $b
- $a % $b (Módulo): Resto da divisão de $a por $b

Operadores de Atribuição

- $a = 5 (Atribuição Simples): O operando da esquerda recebe o valor da expressão da direita
- $a += $b: Equivalente a $a = $a + $b
- $a -= $b: Equivalente a $a = $a - $b
- $a *= $b: Equivalente a $a = $a * $b
- $a /= $b: Equivalente a $a = $a / $b
- $a .= $b: Equivalente a $a = $a . $b (Strings)

Operadores Bit-a-bit

- $a & $b (AND): Efetua um AND entre cada bit de $a e $b
- $a & $b (OR): Efetua um OR entre cada bit de $a e $b
- $a & $b (XOR): Efetua um XOR entre cada bit de $a e $b
- ~$a (NOT): Nega todos os bits de $a
- $a << X (SL): Desloca X vezes à esquerda os bits de $a
- $a >> Y (SR): Desloca Y vezes à direita os bits de $a

Operadores de Comparação

- $a == $b (Igual): TRUE se $a for igual a $b
- $a === $b (Idêntico): TRUE se $a for igual a $b e ambos forem do mesmo tipo
- $a != $b / $a <> $b (Diferente): TRUE se $a for diferente de $b
- $a !== $b (Não idêntico): TRUE se $a for diferente de $b OU não forem do mesmo tipo
- $a < $b / $a > $b (Maior/Menor): TRUE se $a for menor/maior que $b
- $a <= $b / $a >= $b (Maior ou Igual /Menor ou Igual): TRUE se $a for menor/maior ou igual a $b

Operadores de Execução

- Expressões entre acentos graves (` ... `) serão executados como comando da shell
- Semelhante ao comando shell_exec()

Operadores de Incremento/Decremento

- ++$a (Pré-incremento): Incrementa $a em um e retorna seu valor
- $a++ (Pós-incremento): Retorna o valor de $a e o incrementa em um
- --$a (Pré-decremento): Decrementa $a em um e retorna seu valor
- $a-- (Pós-decremento): Retorna o valor de $a e o decrementa em um

Operadores Lógicos

- $a && $b / $a and $b (AND): TRUE se ambos forem TRUE
- $a || $b / $a or $b (OR): TRUE se pelo menos um deles for TRUE
- $a xor $b (XOR): TRUE se ou um ou outro forem verdadeiros
- !$a (NOT): TRUE se $a for FALSE

Operadores de tipo

- $a instanceof MyClass: retorna TRUE se $a for uma instância da classe MyClass

```
class MyClass{}

class NotMyClass{}

$a = new MyClass;

var_dump($a instanceof MyClass);      //imprime bool(true)
var_dump($a instanceof NotMyClass);   //imprime bool(false)
```

Estruturas de Controle

- Permitem maior articulação do código, garantido uma certa “inteligência”
- Alteram o fluxo “linear” de um programa
- Dentre as estruturas de controle, destacam-se:
  - Estruturas Condicionais
  - Estruturas de Repetição
  - Estruturas de Parada

Estruturas de Controle - IF/ELSE

- Permite executar um trecho de código apenas se uma determinada expressão for avaliada como TRUE. Caso contrário, um outro trecho de código poderá ser executado.

```
if($a == $b){
   echo $a;
}
else{
   echo $a + $b;
}
```

```
if($a == $b):
   echo $a;
else:
   echo $a + $b;
endif;
```

```
$a == $b ? echo $a; : echo $a + $b;
```

Estruturas de Controle - SWITCH

- Permite comparar uma variável a diversos valores, tomando decisões diferentes para cada um deles.
- Permite a declaração de uma instrução “default”, para o caso em que nenhuma das alternativas sejam verdadeiras.
- Necessário utilizar o break.

```
switch($beer)
{
    case ‘skol’;
    case ‘brahma’;
    case ‘heineken’;
        echo ‘Good choice’;
    break;
    default;
        echo “Tente novamente: ”;
    break;
}
```

Estruturas de Controle - WHILE

- Permite executar um trecho de código repetidamente enquanto uma condição for verdadeira.

```
while ($a < $b){
   echo $a;
   $a++;
}
```

```
while ($a < $b):
   echo $a;
   $a++;
endwhile;
```

Estruturas de Controle - DO WHILE

- Permite executar um trecho de código repetidamente enquanto uma condição for verdadeira. Diferere do WHILE por realizar a verificação no final do loop.

```
do{
   echo $a;
   $a++;
} while ($a < $b);
```

Estruturas de Controle - FOR

- Difere do WHILE por inicializar uma variável (geralmente de contagem) no início do loop e por executar uma expressão ao final de cada iteração.

```
for($i=0; $i<10; $i++){
   echo $i;
}
```

```
for($i=0; $i<10; $i++):
   echo $i;
endfor;
```

Estruturas de Controle - FOREACH

- Permite percorrer arrays de forma mais fácil.
- Dado um array, o PHP o percorre retornando cada conteúdo na forma de uma nova variável.

```
$array = array(8,3,37,9);

foreach($array as $item){
   echo $item;
}
```

```
$array = array(8,3,37,9);

foreach($array as $key => $item){
   echo “Posição “ . $key . “: “ . $item;
}
```

Estruturas de Controle - FOREACH

- Para alterar diretamente os valores de um array, é possível passar a “variável-conteúdo” por referência.

```
$array = array(8,3,37,9);

foreach($array as &$item){
   $item++;
}
```

VOCÊ SABIA?

Mesmo após o fim do foreach(), a referência de $item para o último elemento permance. Por isso, sugere-se utilizar o comando unset($value).

Estruturas de Controle - BREAK

- O comando continue indica que o loop atual deverá ignorar a iteração atual, passando para a próxima.
- Também suporta a passagem de argumentos numéricos indicando quantas iterações deverão ser puladas.

```
for ($i = 0; $i < 5; ++$i) {
    if ($i == 2){
        continue;
    }
    print $i ;
}
```

Estruturas de Controle - BREAK

- O comando break indica que o loop atual deverá interromper a iteração atual, saindo do loop.
- Também suporta a passagem de argumentos numéricos indicando quantos loops serão saltados.

```
$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo “At 5”;
        break 1;
    case 10:
        echo “At 10”;
        break 2;
    default:
        break;
    }
}
```

Estruturas de Controle - REQUIRE / INCLUDE

- Permitem incluir e avaliar um arquivo específico.
- Utiliza o caminho relativo ao arquivo atual
- O arquivo incluído herda o escopo de variáveis da linha em que a inclusão ocorre.
- Enquanto o include() apenas alerta o usuário de algum erro, o require() emite um erro fatal e encerra a execução do programa.
- include_once() e require_once() permitem que o arquivo seja incluído apenas uma vez em toda a execução do programa.

Estruturas de Controle - GOTO

- Desvio incondicional
- Considerada uma má prática de programação
- Deve vir acompanhado de um parâmetro, que indica a que ponto o código deverá saltar.

```
for($i=0,$j=50; $i<100; $i++) {
  while($j--) {
    if($j==17) goto end;
  }
}
echo “i = $i”;
end:
echo ‘j hit 17’;
```

Funções

- É possível gerar funções dentro de funções e até mesmo definição de classes.
- Nomes de funções seguem a mesma regra para variáveis e constantes.
- Todas as funções possuem escopo global
- É possível utilizar funções recursivas
- As funções não precisam ser declaradas antes de serem referenciadas, exceto quanto ela é condicionalmente definida ou definida dentro de outra função.

Funções

```
function foo()
{
  function bar()
  {
    echo “Eu não existo até foo() ser chamada.\n”;
  }
}

/* Nós não podemos chamar bar() ainda */
foo();

/* Agora nós podemos chamar bar() */
bar();
```

Funções

- Argumentos de funções são avaliados da esquerda para a direita
- Utiliza a correspondência posicional
- Possui como padrão a passagem por valor, mas permite a passagem por referência.
- A passagem de objetos é, por padrão, feito por referência.
- Admite valores padrões de argumentos e lista de argumentos de tamanho variável.
- Argumentos-padrão deverão ser os últimos argumentos de uma função

Funções

```
function cafeteira ($tipo = “cappuccino”)
{
    return “Fazendo uma xícara de café $tipo.\n”;
}


echo cafeteira ();              //Fazendo uma xícara de cappuccino.
echo cafeteira (null);          //Fazendo uma xícara de .
echo cafeteira (“expresso”);    //Fazendo uma xícara de expresso.
```

Funções

- É possível restringir o tipo das variáveis que são passadas como parâmetro para uma função.
- É possível restringir que o argumento seja uma instância de uma determinada classe.
- O PHP realiza a coerção de argumentos automaticamente a menos que isto seja desabilitado pela expressão **declare(strict_types=1);**.

Funções

```
declare(strict_types=1);

function sum(int $a, int $b) {
    return $a + $b;
}

var_dump(sum(1, 2));                    //Imprime int(3)
var_dump(sum(1.5, 2.5));                //Erro de execução
```

Funções

```php
function sum(int $a, int $b) {
    return $a + $b;
}

var_dump(sum(1, 2));                //Imprime int(3)
var_dump(sum(1.5, 2.5));            //Imprime int(3)
```

Funções

- PHP aceita a passagem de um número variável de parâmetros por meio do indicador ...
- Os argumentos são passados na forma de array
- Também é possível utilizar o indicador ... para converter uma array em uma lista de parâmetros

Funções

```php
function sum(...$numbers) {
    $acc = 0;
    foreach ($numbers as $n) {
        $acc += $n;
    }
    return $acc;
}

echo sum(1, 2, 3, 4);
```

Funções Variáveis

- Nomes de variáveis com () ao final são interpretadas como funções
- Muito utilizada para implementar callbacks
- Também podem ser utilizadas para métodos de objetos

```
function imprimir($string)
{
    echo $string;
}

$func = 'imprimir';
$func('Testando');       //Chama imprimir('Testando');
```

PARTE 3

Orientação a Objetos

Orientação a Objetos

- Classes são definidas a partir da palavra **class** seguida do nome da classe e um par de chaves.
- Dentro das chaves, pode-se declarar atributos e métodos daquela classe.
- Todo método possui uma pseudo-variável $this que faz referência ao objeto que chamou aquele método.
- Novas instância de uma classe são criadas pelo comando **new**

Orientação a Objetos

```
class PrimeiraClasse
{
    public $var = ‘Maçã’;

    public function displayVar() {
        echo $this->var;
    }
}

$comida = new PrimeiraClasse();
$comida->displayVar();          //imprime “Maçã”
```

Orientação a Objetos - Herança

- Uma classe pode herdar os atributos e métodos de uma outra usando a palavra-chave **extends**.
- É possível sobrescrever os métodos de uma classe pai contanto que estes não tenham sido declarados com a palavra-chave **final**.
- A ligação tardia é feita por meio do operador **static::** antes do nome da função.
- É possível acessar os métodos da classe-pai referenciando-os com a palavra-chave **parent::**.
- PHP não suporta herança múltipla.

Orientação a Objetos - Herança

```
class SegundaClasse extends PrimeiraClasse
{
    function displayVar()
    {
        echo “Banana”;
        parent::displayVar();
    }
}

$segunda = new SegundaClasse();
$segunda->displayVar();
```

Orientação a Objetos - Herança

```
class A {
    public static function who() {
        echo __CLASS__;
    }
    public static function test() {
        static::who();     }
}

class B extends A {
    public static function who() {
         echo __CLASS__;
    }
}

B::test();
```

Orientação a Objetos - Atributos

- São declarados da mesma forma que variáveis, mas são preceddidos por **public**, **protected** ou **private**.
- Podem ser inicializados contanto que seu valor seja constante.
- Atributos de um objeto podem ser acessados pelo operador **->**
- É possível declarar constantes como atributos de uma classe.
- Constantes são acessadas pelo operador **::**.
- Constantes são sempre publicamente visíveis

Orientação a Objetos - Atributos

```
class SegundaClasse
{
    const constante = 'valor constante';

    function displayVar() {
        echo  self::constante;
    }
}

$classe = new SegundaClasse();
$classe->displayVar();

echo $classe::constante;
```

Orientação a Objetos - Construtores e Destrutores

- É possível declarar métodos construtores e destrutores para uma classe chamando-os por **__construct()** e **__destruct()**.
- Construtores e destrutores não garantem a construção/destruição da classe-pai.
- Métodos destrutores são chamados quando não existe mais nenhuma referência àquele objeto ou for explicitamente destruído.

Orientação a Objetos - Construtores e Destrutores

```
class PrimeiraClasse {
    function __construct() {
        print “Construtor de Primeira Classe”;
    }
}

class SegundaClasse extends PrimeiraClasse {
    function __construct() {
        parent::__construct();
        print “Construtor de Segunda Classe”;
    }
}

$obj = new PrimeiraClasse();
$obj = new SegundaClasse();
```

Orientação a Objetos - Visibilidade

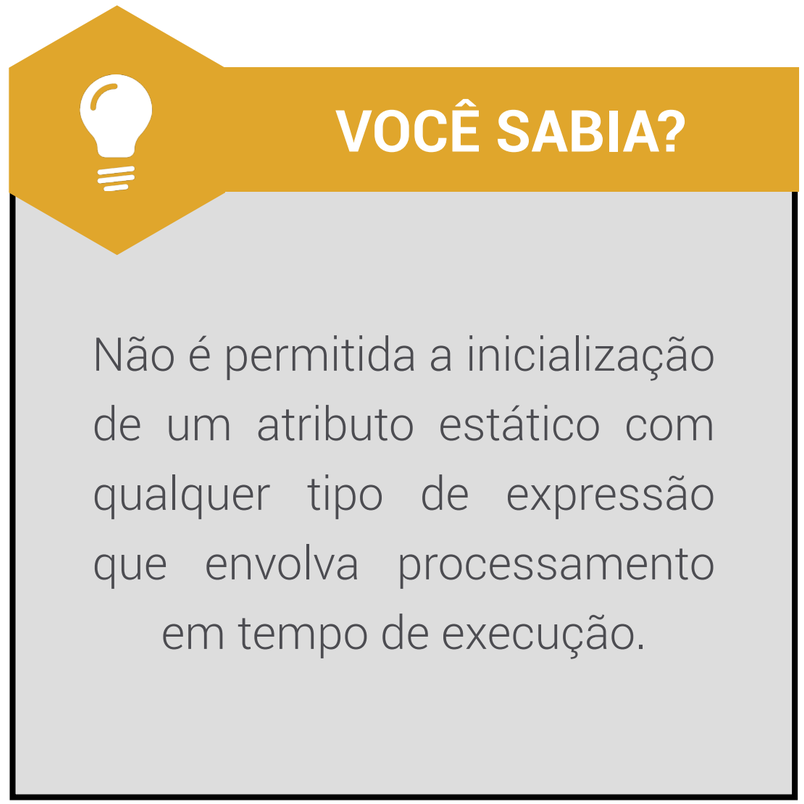
- Atributos e métodos podem ter sua visibilidade alterada por meio das palavras-chave **public**, **protected** ou **private**.
- Quando a visibilidade não for especificada, o PHP adota o **public** como padrão.
  - **Public:** permite que qualquer um possa acessar o elemento
  - **Protected:** permite que apenas classes herdadas tenham acesso ao elemento
  - **Private:** não permite que ninguém acesse o elemento.

Orientação a Objetos - Visibilidade

```
class MinhaClasse{
    public $publica = 'Public';
    protected $protegida = 'Protected';
    private $privada = 'Private';
    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
```

Orientação a Objetos - Static

- Atributos ou métodos podem ser declarados como static, fazendo com que estes possam ser acessíveis sem a instanciação da classe.
- Não devem ser acessados usando o operador **->**. Neste caso, o operador a ser utilizado é o **::**.



Orientação a Objetos - Static

```
class ClasseA {
  public static function umMetodoEstatico() {
    // ...
  }
}

ClasseA::umMetodoEstatico();
```

Orientação a Objetos - Classes Abstratas

- Definidas pela palavra-chave **abstract**
- Classes abstratas não podem ser instanciadas.
- Métodos abstratos não podem definir a sua implementação.
- Classes com pelo menos um método abstrato também devem ser abstratas.
- A implementação de um método abstrato não pode ter visibilidade maior que aquela usada em sua definição e sua assinatura deve coincidir à utilizada em sua definição.

Orientação a Objetos - Classes Abstratas

```
abstract class ClasseAbstrata
{
    abstract public function pegarValor();

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

class ClasseConcreta1 extends ClasseAbstrata
{
    public function pegarValor() {
        echo “ClasseConcreta”;
    }
}
```

Orientação a Objetos - Classes Abstratas

```
$classe1 = new ClasseAbstrata();       //NÃO PODE
$classe2 = new ClasseConcreta();       //PODE
$classe2->imprimir();                  //imprime “ClasseConcreta”
```

Orientação a Objetos - Interfaces

- Especifica quais atributos e métodos uma classe deverá implementar.
- Uma classe “implementa” uma interface por meio da palavra-chave **implements**.
- Classes podem implementar mais de uma interface
- Interfaces podem herdar propriedades de outras interfaces.

Orientação a Objetos - Interfaces

```
interface iTemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}

class BadTemplate implements iTemplate                  //NÃO PODE
{
    private $vars = array();

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }
}
```

Orientação a Objetos - Traits

- Permite a reutilização de códigos livremente em classes independentes.
- Reduz as limitações da linguagem não suportar herança múltipla.
- Não é possível instanciar Traits
- Classes “herdam” traits por meio da palavra-chave **use**.
- Classes podem sobrescrever métodos de uma trait.
- Classes podem “herdar” mais de uma trait

Orientação a Objetos - Traits

```
trait minhaTrait {
    function getReturn1() { /* ... */ }
    function getReturn2() { /* ... */ }
}

class Classe1 extends SuperClasse {
    use minhaTrait;
    /* ... */
}

class Classe2 extends SuperClasse {
    use minhaTrait;
    /* ... */
}
```

Namespaces

- Permitem “agrupar” itens (geralmente classes).
- Atuam como diretórios de computador:
  - Arquivos de mesmo nome podem existir em diretórios diferentes, mas não podem coexistir em um mesmo diretório.
- Evitam colisões entre códigos criados pelo usuário e as bibliotecas internas do PHP.
- São afetadas pelos namespaces classes, interfaces, funções e constantes.
- Deve ser a primeira declaração em um documento PHP.

Namespaces

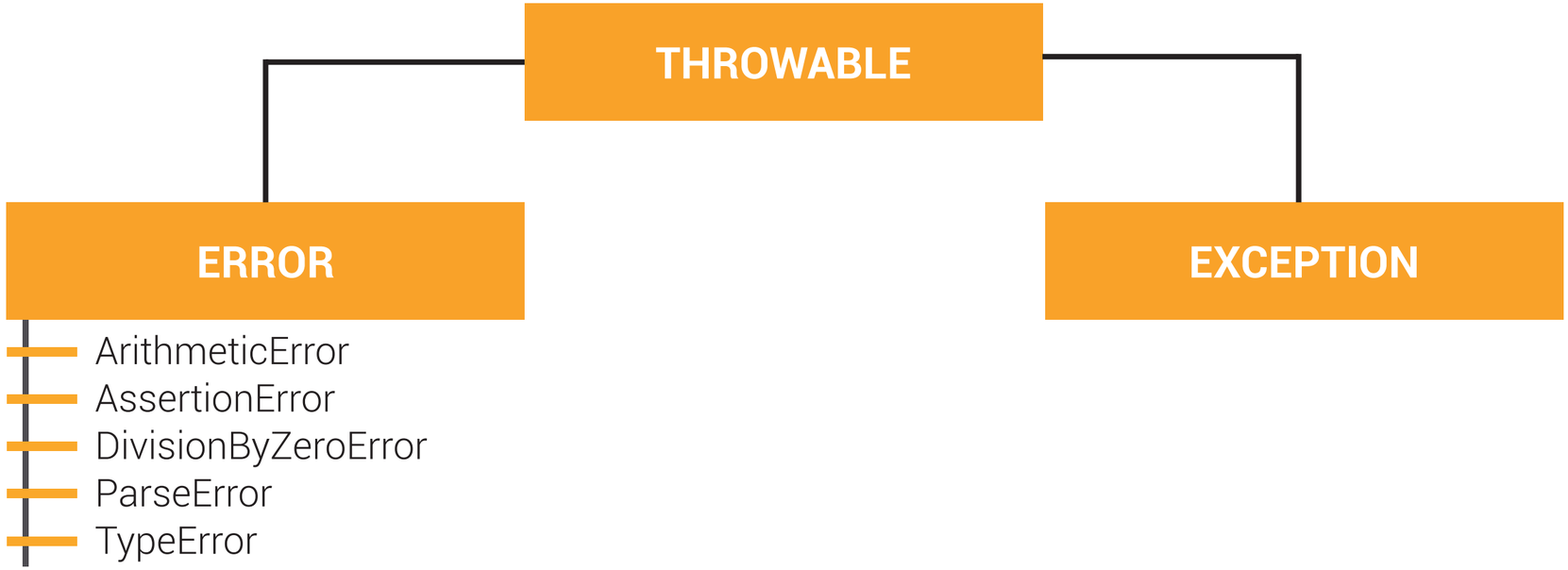
```
namespace MyProject;

const CONNECT_OK = 1;

class Connection {
   /* ... */
}

function connect() {
   /* ... */
}
```

PARTE 4

Exceções

Exceções

- PHP trata erros de execução de forma semelhante a exceções.
- Como Errors não herdam de Exception, ambos precisam de tratamentos separados.



Exceções

- Tratamento de exceções semelhante às demais linguagens.
- Uma exceção pode ser lançada (*throw*) e capturada (*catch*).
- Cada bloco *try* precisa ter pelo menos um *catch* ou *finally* correspondente
- A exceção lançada deve ser uma instância de **Exception** ou uma subclasse da mesma.
- Blocos *finally* sempre serão executados, mesmo que uma exceção tenha sido lançada.

Exceções

```
function inverso($x) {
    if (!$x) {
        throw new Exception('Divisão por zero.');
    }
    return 1/$x;
}

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

PARTE 5

Features

Cookies

- PHP suporta transparentemente Cookies HTTP.
- Permite armazenar dados do cliente de forma temporária.
- Cookies são criados pela função setcookie() e são armazenados na variável global $_COOKIES.

```
setcookie(“color”, “red”);

...

echo $_COOKIE[“color”];
```

Sessões

- Semelhante ao Cookie, armazena dados do cliente de forma temporária.
- Mais fáceis de manipular, pois todo o tratamento é feito em $_SESSION.
- Existem diversas funções pré-existentes para sessões, como session_destroy(), session_encode() e session_unset().

```
session_start();
if (!isset($_SESSION[‘count’])) {
  $_SESSION[‘count’] = 0;
} else {
  $_SESSION[‘count’]++;
}
```

Upload de Arquivos

- Muito utilizado em conjunto com formulários HTML

```
<form enctype=”multipart/form-data” action=”” method=”POST”>
   Enviar esse arquivo: <input name=”userfile” type=”file” />
   <input type=”submit” value=”Enviar arquivo” />
</form>

<?php
$uploaddir = ‘/var/www/uploads/’;
$uploadfile = $uploaddir . basename($_FILES[‘userfile’][‘name’]);

if (move_uploaded_file($_FILES[‘userfile’][‘tmp_name’], $uploadfile)) {
    echo “Arquivo válido e enviado com sucesso.\n”;
}
?>
```

Tratamento de Arquivos

- PHP tem funções específicas para o tratamento de arquivos de texto.
- Funções semalhantes às utilizadas em C.

```
$handle = fopen(“/tmp/arquivodeentrada.txt”, “r”);
if ($handle) {
    while (!feof($handle)) {
        $buffer = fgets($handle, 4096);
        echo $buffer;
    }
    fclose($handle);
}
```

Bancos de Dados

- Funções para conexão com bancos de dados
- Suporte a diversos tipos de bancos de dados, incluindo MySQL e PostgreSQL.

```
$link = mysqli_connect(“localhost”, “my_user”, “my_password”, “world”);

if ($result = mysqli_query($link, “SELECT Name FROM City LIMIT 10”)) {

   printf(“Select returned %d rows.\n”, mysqli_num_rows($result));

}
```

Concorrência

- Implementada pela biblioteca pthreads
- Para gerar concorrência, a classe em questão deve herdar da classe Thread e executar o método run().
- **Classe Worker:** a thread é executada até a classe saia de escopo ou o programa seja finalizado.
- **Classe Mutex:** implementa métodos de prevenção de acesso mútuo a uma região crítica.

Concorrência

```
class AguardaRand extends Thread {

     protected $id;

     public function __construct($id) {
         $this->id = $id;
     }

     public function run() {
         $tempo_rand = mt_rand(1, 4);
         sleep($tempo_rand);
         printf("Sou a thread %d e aguardei %d segundos\n",
             $this->id,
             $tempo_rand
         );
     }
}
```

Concorrência

```
$vetor = array();
for ($id = 0; $id < 10; $id++) {
    $vetor[] = new AguardaRand($id);
}

foreach ($vetor as $thread) {
    $thread->start();
}
```

Garbage Collector

- PHP possui um Garbage Collector nativo
- O sistema de “limpeza” é feito pela quantidade de referência para um determinado elemento.
- Quando o número de referência chega a 0, o PHP realiza automaticamente a remoção deste elemento da memória do computador.
- Referências podem ser removidas utilizando a função unset().

PARTE 6

Comparação de Linguagens

PHP vs. C & Java

Critérios Gerais	PHP	Java	C
Aplicabilidade	Parcial	Parcial	Sim
Confiabilidade	Parcial	Sim	Não
Aprendizado	Parcial	Não	Não
Eficiência	Sim	Parcial	Sim
Portabilidade	Parcial	Sim	Não
Paradigma	OO e Estruturado	Orientado a Objetos	Estruturado
Evolutibilidade	Parcial	Sim	Não
Reusabilidade	Sim	Sim	Parcial
Interação	Sim	Parcial	Sim
Custo	?	?	?

PHP vs. C & Java

Critérios Específicos	PHP	Java	C
Escopo	Sim	Sim	Sim
Expressões e Comandos	Sim	Sim	Sim
Tipos Primitivos e Compostos	Sim	Sim	Sim
Gerenciamento de Memória	Programador / Sistema	Sistema	Programador
Persistência de Dados	Biblioteca de classes, Serialização	JDBC, Biblioteca de classes, Serializaçao	Biblioteca de funções
Passagem de Parâmetros	Lista variável, por valor e cópia referência	Lista variável, por valor e cópia referência	Lista variável e por valor

PHP vs. C & Java

Critérios Específicos	PHP	Java	C
Encapsulamento e Proteção	Sim	Sim	Parcial
Sistemas de Tipos	Parcial	Sim	Não
Verificação de Tipos	Dinâmica	Estática / Dinâmica	Estática
Polimorfismo	Todos	Todos	Coerção e Sobrecarga
Exceções	Sim	Sim	Não
Concorrência	Biblioteca de Funçoes	Sim	Biblioteca de Funçoes

Referências Bibiográficas

- Diversos autores. **PHP Manual.** Disponível em <http://php.net/manual/en/index.php>.

- Autor desconhecido. **Scripting Languages I: Node.js, PHP, Python, Ruby (Sheet One)**. Disponível em: <http://hyperpolyglot.org/scripting>