

# Seminário - PHP

...

Linguagens de Programação - 2017/2

Lucas S. Alves, Guilherme S. Scopel e João Vitor A. Fazolo


# O que é PHP?

- PHP (PHP: Hypertext Preprocessor - originalmente Personal Home Page) é uma linguagem de script open source de uso geral amplamente utilizada para desenvolvimento web, principalmente pela sua capacidade de ser embutida dentro HTML e continuar se atualizando constantemente com tecnologias novas (como a integração com o Git).
- Trata-se de uma linguagem extremamente modularizada, o que a torna ideal para instalação e uso em servidores web.

# Por que usar PHP?

- Vasta documentação na internet
- Arquivos podem conter texto e códigos HTML, CSS, JavaScript e PHP
- É “rodável” em várias plataformas (Windows, Linux, Mac OS X, etc.)
- Compatível com a maioria dos servidores utilizados atualmente (Apache, IIS, etc.)
- Integrável com uma grande gama de bancos de dados
- Amigável para iniciantes
- É GRÁTIS

# História

- 1995 - Criado por Rasmus Lerdorf como um pacote de programas CGI, com o nome Personal Home Page Tools
  - 1997 - Foi lançado o novo pacote da linguagem com o nome de PHP/FI, trazendo a ferramenta Forms Interpreter, um interpretador de comandos SQL
    - Refeito pelos programadores que vieram a ser conhecidos como The PHP Group
  - 1998 - Lançado o PHP3, primeira versão com o uso de orientação à objetos
- 
- 2000 - Lançado o PHP4, com várias melhorias (e bugs) em relação à OO
  - 2004 - PHP 5 lançado. Versão que recebeu suporte por mais tempo e teve um número significativo de funcionalidades implementadas e atualizadas
  - PHP6?
  - 2012 - Lançamento do PHP 7, a versão mais atual.

# Características da linguagem

- Tipagem dinâmica
- Pode ser classificada, no que tange aos paradigmas, como procedural, reflexiva, orientada à objeto e funcional
- Variáveis são *case sensitive*, ao contrário de funções (tanto padrão quanto definidas por usuário), classes e palavras-chave
- Fracamente tipada
- *Server-side*

# Basically speaking

- Arquivos terminam em *.php*
- Scripts PHP podem estar em qualquer parte do documento e são iniciados com `<?php` e finalizados `?>`
- Comandos terminam em ;

```
<?php
```

```
# Tipos de comentário - Linha única
```

```
// Também linha única
```

```
/*
```

```
    Comentário em  
    múltiplas linhas
```

```
*/
```

```
/* Este tipo também pode ser utilizado no meio de uma linha de  
código */
```

```
?>
```

# Mostrando resultados

- As funções principais de PHP para imprimir resultados são o **echo** e o **print**
- A principal diferença entre as duas funções é o valor de retorno (**print** retorna 1, **echo** não possui valor de retorno)
- Utiliza-se o `.` para concatenação de *outputs* (no caso, strings)

// Linhas vazias acima da tag `<?php ?>` são interpretadas como `\n` na execução do programa

`<?php`

```
$z = "Isso eh um teste";
```

```
$w = print $z . "\n"; // Imprime -- Isso eh um teste -- e retorna 1 para a variável $w
```

```
echo $w . "\n"; // Imprime 1
```

`?>`

# Amarrações



# Variáveis

- As variáveis em PHP começam com \$
- Em PHP não há comando para se declarar uma variável
- Variáveis só podem começar com uma letra ou *underscores*
- Só podem conter caracteres alfanuméricos e *underscores* (A-z, 0-9, e \_ )
- A linguagem é extremamente permissiva quanto ao tipo das variáveis
- Escopo das variáveis pode ser **global**, **local** ou **estático**

# Escopos de variáveis

```
<?php
```

```
    $x = 1; # Global
```

```
function exemplo() {
```

```
    $y = 2; # Local
```

```
    static $z = 3; # Estática
```

```
}
```

```
?>
```

# Variáveis globais

- Variáveis globais podem ser diretamente acessadas fora de funções ou dentro com o uso de **global**
- A linguagem também guarda todas as variáveis globais dentro de uma array (que também permite o acesso de dentro de funções)

```
<?php
```

```
$x = 1;
```

```
$y = 2;
```

```
function exemplo() {
```

```
    global $x; # Acesso via global
```

```
    $x += ($GLOBALS['y'] /* acesso via array */ += 1);
```

```
}
```

```
echo $x . " e " . $y . "\n"; // Imprime "1 e 2"
```

```
exemplo();
```

```
echo $x . " e " . $y . "\n"; // Imprime "4 e 3"
```

```
?>
```

# Variáveis static

- Declarar uma variável local com **static** faz com que ela permaneça na memória após a execução da função

```
<?php
function teste() {
    static $x = 0;
    echo $x;
    $x++;
}

teste(); // Imprime 0
teste(); // Imprime 1
teste(); // Imprime 2
?>
```

# Váriaveis SuperGlobals

- São variáveis possíveis de serem acessadas independentemente do escopo, função, classe ou arquivo sem nenhum tipo de chamada especial.
  - `$GLOBALS`
  - `$_SERVER`
  - `$_REQUEST`
  - `$_POST`
  - `$_GET`
  - `$_FILES`
  - `$_ENV`
  - `$_COOKIE`
  - `$_SESSION`

# Palavras reservadas e palavras-chave

<code>abstract</code>	<code>and</code>	<code>array()</code>	<code>as</code>	<code>break</code>	<code>callable</code>	<code>case</code>	<code>catch</code>
<code>class</code>	<code>clone</code>	<code>const</code>	<code>continue</code>	<code>declare</code>	<code>default</code>	<code>die()</code>	<code>do</code>
<code>echo</code>	<code>else</code>	<code>elseif</code>	<code>empty()</code>	<code>enddeclare</code>	<code>endfor</code>	<code>endforeach</code>	<code>endif</code>
<code>endswitch</code>	<code>endwhile</code>	<code>eval()</code>	<code>exit()</code>	<code>extends</code>	<code>final</code>	<code>finally</code>	<code>for</code>
<code>if</code>	<code>implements</code>	<code>include</code>	<code>include_on ce</code>	<code>instanceof</code>	<code>insteadof</code>	<code>interface</code>	<code>isset</code>
<code>list()</code>	<code>namespace</code>	<code>new</code>	<code>or</code>	<code>print</code>	<code>private</code>	<code>protected</code>	<code>public</code>
<code>require</code>	<code>require_onc e</code>	<code>return</code>	<code>static</code>	<code>switch</code>	<code>throw</code>	<code>trait</code>	<code>try</code>
<code>unset()</code>	<code>use</code>	<code>var</code>	<code>while</code>	<code>xor</code>	<code>yield</code>	<code>__halt_compiler()</code>	

# E aí: cópia ou referência?

- PHP suporta ambos os tipos de passagem de parâmetros
- Os seguintes itens podem ser acessados por referência:
  - Variáveis;
  - Instruções *new*;
  - Referências retornadas de funções.

```
<?php
$a = 1;
function j(&$b){
    $b++;
}
echo $a; #Imprime 1
j($a);
echo $a; #Imprime 2
```

?>

# Valores e tipos



# Tipos de dados

- PHP tem tipagem dinâmica, é fracamente tipada e suporta os seguintes tipos de dados:
  - String;
    - PHP não possui suporte nativo para Unicode
  - Integer - Pode ser representado em decimal, octal e hexadecimal ;
  - Float;
  - Boolean;
    - True ou false (não é *case sensitive*)
    - Valores considerados *false*: 0, 0.0, string vazia ou “0”, NULL ou array sem elementos
  - Array;
  - Object;
  - NULL;
    - Variáveis criadas sem valor recebem NULL
  - Resource.

# Tipo string

- Em PHP, strings são uma sequência de caracteres onde cada caractere é representado por um byte
  - Por causa disso, só é possível representar 256 caracteres, sendo esse o motivo da falta suporte nativo a Unicode
  - String segue a codificação do arquivo de script

```
<?php
    echo 'Hello, World!\n'; # Imprime Hello, World!\n
    echo "Hello, World!\n"; # Imprime Hello, World!
?>
```

# Funções - Tipo string

- `strlen` Retorna o tamanho da string.
- `str_replace` Substitui parte de uma string por outra string
- `strpos` Encontra a posição da primeira ocorrência de uma substring
- `substr` Retorna uma parte de uma string
- `strtoupper` / `strtolower` Converte a string para maiúscula / minúscula
- `explode` Divide uma string em uma array de strings

# Tipo array

- Um array no PHP é na verdade um mapa ordenado
- Pode ser tratado como um array, uma lista (vetor), *hashtable* (que é uma implementação de mapa), dicionário, coleção, pilha, fila e provavelmente mais
- Existe a possibilidade dos valores do array serem outros arrays, árvores e arrays multidimensionais
- A chave é opcional, podendo ser um valor inteiro ou uma string
- Exemplo de código mais rebuscado: 44

```
<?php
```

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

```
$b = array(1 => "Um", 2 => "Dois", 3 => "Tres");
```

```
echo "$a[0]\n"; # Imprime 1
```

```
echo "$b[2]\n"; # Imprime Dois
```

```
?>
```

# Funções interessantes de tipo

- **var\_dump** e **gettype**
  - Retornam o tipo da variável
  - **var\_dump** imprime o tipo além do conteúdo, **gettype** retorna uma string com o tipo
- **is\_\***
  - Retorna true ou false, dependendo da equivalência

```
<?php
```

```
$z = "Isso eh um teste";  
$w = print $z . "\n"; // Imprime 1  
var_dump($z); // Imprime string(16) "Isso eh um teste"  
echo gettype($w) . "\n"; // Imprime integer  
echo gettype(is_string($z)) . "\n"; // Imprime boolean
```

```
?>
```

# Conversão de tipos

- (int), (integer) - molde para inteiro
- (bool), (boolean) - converte para booleano
- (float), (double), (real) - converte para número de ponto flutuante
- (string) - converte para string
- (array) - converte para array
- (object) - converte para objeto
- (unset) - converte para NULL (PHP 5)

# Constantes

- Diferentemente das variáveis, as constantes são globais por todo o script.
- Para criar uma constante usamos a função: `define(name, value, case-insensitive)`, o último parâmetro define se o nome da constante irá ser case-sensitive ou não.

```
<?php
define("helloworld", "Hello, World!\n",true);

function Teste() {
    echo HELLOWORLD; // Imprime "Hello, World!"
}
Teste();
?>
```

# Constantes mágicas

- Resolvidas em tempo de compilação
- Variam de acordo com a linha/diretório/o que referencia

<code>__LINE__</code>	O número da linha corrente do arquivo.
<code>__FILE__</code>	O caminho completo e nome do arquivo com links simbólicos resolvidos. Se utilizado dentro de um <code>include</code> , o nome do arquivo incluído será retornado.
<code>__DIR__</code>	O diretório do arquivo. Se usado dentro de um <code>include</code> , o diretório do arquivo incluído é retornado.
<code>__FUNCTION__</code>	O nome da função.
<code>__CLASS__</code>	O nome da classe. O nome da classe inclui o namespace em que foi declarado (por exemplo, <code>Foo\Bar</code> ).
<code>__TRAIT__</code>	O nome do trait. O nome do trait inclui o namespace em que foi declarado (por exemplo, <code>Foo\Bar</code> ).
<code>__METHOD__</code>	O nome do método da classe.
<code>__NAMESPACE__</code>	O nome do namespace corrente.



# Armazenamento

# Coletor de lixo

- O coletor de lixo do PHP utiliza o método de contagem de referências
- Esse método é ineficaz para referências cíclicas (como no tipo array)
  - Para solucionar esse problema, PHP tem um segundo coletor de lixo que opera nesses casos específicos
- O coletor de lixo pode ser desabilitado através do **gc\_disable()**

# Serialização

- A função `serialize()` retorna uma string contendo uma representação byte-stream de qualquer valor que pode ser armazenado pelo PHP
- função `unserialize()` pode utilizar essa string para recriar os valores originais da variável.
- Para desserializar um objeto com a função `unserialize()`, a classe deste objeto precisar estar definida no arquivo.
- Pode-se também fazer com que a classe do objeto a ser serializada implemente a interface `serialize` para que o programador defina como será feita.

```
<?php
$myvar = array(
    'oi',
    42,
    array(1,'dois'),
    'cacau'
);
// converte para uma string
$string = serialize($myvar);
echo $string;
/* prints
a:4:{i:0;s:2:"oi";i:1;i:42;i:2;a:2:{i:0;i:1;i:1;s:4:"dois";}i:3;s:5:"cacau";}
*/
// pegar de volta para memoria principal a variavel.
```

```
$newvar = unserialize($string);
print_r($newvar);
/* prints
Array(
    [0] => oi
    [1] => 42
    [2] => Array(
        [0] => 1
        [1] => dois
    )
    [3] => cacau
)
*/
?>
Array ( [0] => oi [1] => 42 [2] => Array ( [0] =>
1 [1] => dois ) [3] => cacau )
```

# Expressões e comandos

# Operadores

- Realizam operações sobre variáveis e valores
- PHP divide os operadores nos seguintes grupos:
  - Aritméticos;
  - De atribuição;
  - Comparação;
  - Incremento/Decremento;
  - Lógicos;
  - De string;
  - De array.

# 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  $\$a$  e  $\$b$ .
- $\$a \% \$b$               Resto                      Resto de  $\$a$  dividido por  $\$b$ .
- $\$a ** \$b$               Exponencial              Resultado de  $\$a$  elevado a  $\$b$

# Operadores de atribuição

- Aritméticos:

- $\$a += \$b$      $\$a = \$a + \$b$     Adição
- $\$a -= \$b$      $\$a = \$a - \$b$     Subtração
- $\$a *= \$b$      $\$a = \$a * \$b$     Multiplicação
- $\$a /= \$b$      $\$a = \$a / \$b$     Divisão
- $\$a \% = \$b$      $\$a = \$a \% \$b$     Resto

- De string:

- $\$a .= \$b$      $\$a = \$a . \$b$     Concatenação

# Operadores de comparação - Comuns

- $a == b$  Igual Verdadeiro (TRUE) se  $a$  é igual a  $b$ .
- $a != b$  Diferente Verdadeiro se  $a$  não é igual a  $b$ .
- $a <> b$  Diferente Verdadeiro se  $a$  não é igual a  $b$ .
- $a < b$  Menor que Verdadeiro se  $a$  é estritamente menor que  $b$ .
- $a > b$  Maior que Verdadeiro se  $a$  é estritamente maior que  $b$ .
- $a <= b$  Menor ou igual Verdadeiro se  $a$  é menor ou igual a  $b$ .
- $a >= b$  Maior ou igual Verdadeiro se  $a$  é maior ou igual a  $b$ .



# Outros operadores de comparação

- `$a === $b`      Idêntico      Verdadeiro (TRUE) se `$a` é igual a `$b`, e eles são do mesmo tipo.
- `$a !== $b`      Não idêntico      Verdadeiro de `$a` não é igual a `$b`, ou eles não são do mesmo tipo.
- `$a <=> $b`      Spaceship      Retorna 1 se `$a > $b`, 0 se `$a = $b` e -1 se `$a < $b`

# Operadores de incremento e decremento

- `++$a` Pré-incremento      Incrementa `$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.

# Operadores lógicos

- `$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.

# Curto circuito

- Implementado em && , and , || , or

```
<?php
    $a = 1;
    $b = 2;
    ++$a || ++$b;
    echo "$a e $b\n"; #Imprime 2 e 2
?>
```

# Operadores de array

- `$a + $b`      União      União de `$a` e `$b`. (prioridade do declarado antes)
- `$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 !== $b`      Não identidade      TRUE se `$a` não é idêntico a `$b`.

# Operadores bitwise

- $a \& b$  E (AND) Os bits que estão ativos tanto em  $a$  quanto em  $b$  são ativados.
- $a | b$  OU (OR inclusivo) Os bits que estão ativos em  $a$  ou em  $b$  são ativados.
- $a \wedge b$  XOR (OR exclusivo) Os bits que estão ativos em  $a$  ou em  $b$ , mas não em ambos, são ativados.
- $\sim a$  NÃO (NOT) Os bits que estão ativos em  $a$  não são ativados, e vice-versa.
- $a << b$  Deslocamento à esquerda Desloca os bits de  $a$   $b$  passos para a esquerda (cada passo significa "multiplica por dois")
- $a >> b$  Deslocamento à direita Desloca os bits de  $a$   $b$  passos para a direita (cada passo significa "divide por dois")

# Precedência de operadores

Associação	Operadores
não associativo	<i>clone new</i>
esquerda	<i>[</i>
direita	<i>**</i>
direita	<i>++ -- ~ (int) (float) (string) (array) (object) (bool) @</i>
não associativo	<i>instanceof</i>
direita	<i>!</i>
esquerda	<i>* / %</i>
esquerda	<i>+ - .</i>
esquerda	<i>&lt;&lt; &gt;&gt;</i>
não associativo	<i>&lt; &lt;= &gt; &gt;=</i>
não associativo	<i>== != === !== &lt;&gt; &lt;=&gt;</i>
esquerda	<i>&amp;</i>
esquerda	<i>^</i>
esquerda	<i> </i>
esquerda	<i>&amp;&amp;</i>
esquerda	<i>  </i>
direita	<i>??</i>
esquerda	<i>? :</i>
direita	<i>= += -= *= **= /= .= %= &amp;=  = ^= &lt;&lt;= &gt;&gt;=</i>
esquerda	<i>and</i>
esquerda	<i>xor</i>
esquerda	<i>or</i>

# Controle de fluxo

- PHP suporta a maioria das estruturas clássicas de controle de fluxo
  - if...else...elseif;
  - while (e do-while);
  - for (e foreach);
  - break e continue;
  - switch;
  - declare;
  - return;
  - require (require\_once);
  - include (include\_once);
  - Além do famigerado *goto*.



# Se x, então...

- Estrutura similar à C
- O PHP realiza avaliações pelo valor booleano
- PHP possui operador ternário

```
<?php
```

```
$a = 10;
```

```
$b = 20;
```

```
if(is_int($a)) echo "a eh um inteiro\n";
```

```
if ($a > $b):
```

```
    echo "a é maior que b\n";
```

```
else:
```

```
    echo "a é menor que b\n";
```

```
endif;
```

```
$a = $b;
```

```
if ($a > $b) {
```

```
    echo "a é maior que b\n";
```

```
} elseif($a == $b) {
```

```
    echo "a é igual a b\n";
```

```
} else {
```

```
    echo "a é menor que b\n";
```

```
}
```

```
$a += $b;
```

```
echo ($a > $b) ? "MAIOR\n":"MENOR\n";
```

```
$a = 0;
```

```
echo ($a > $b) ?:"MENOR\n";
```

```
?>
```

# while e do-while

```
<?php
$a = 10;
while($a > 0){
    $a--;
}
echo $a."\n";
?>
```

```
<?php
$a = 10;
while($a > 0):
    $a--;
endwhile;
echo $a."\n";
?>
```

```
<?php
$a = 10;
do{
    $a--;
}while($a > 0);
echo $a."\n";
?>
```

# For

```
<?php
```

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

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

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

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

# Foreach

- Fornece uma maneira simples de iterar sobre arrays e objetos
  - Tentar utilizar sobre outros tipos resulta em erro
- Pode iterar sobre uma array utilizando tanto o mecanismo de cópia quanto o de referência

```
<?php
```

```
$a = array(  
    "um" => 1,  
    "dois" => 2,  
    "3" => 3,  
    "vinte" => 20  
);
```

```
foreach ($a as $v) {  
    echo "$v\n";  
}
```

```
$i = 0;
```

```
foreach ($a as $k => $v):  
    echo "\$a[$k] => $v\n";  
endforeach;
```

```
$a = array();  
$a = array();  
$a[0][0] = "a";  
$a[0][1] = "b";  
$a[1][0] = "y";  
$a[1][1] = "z";
```

```
foreach ($a as $v1) {  
    foreach ($v1 as $v2) {  
        echo "$v2\n";  
    }  
}  
?>
```

# Break

- *break* finaliza a execução da estrutura *for*, *foreach*, *while*, *do-while* ou *switch* atual
- Aceita um argumento adicional, que diz quantas estruturas aninhadas o *break* deve interromper
  - Não aceita variáveis nem 0 como argumento

```
<?php
$a = array(
    "um" => 1,
    "dois" => 2,
    "3" => 3,
    "vinte" => 20
);
foreach ($a as $n) {
    if($n == 3)
        break;
    echo "$n\n";
}
```

```
foreach ($a as $n):
    switch ($n) {
        case 1:
            echo "Volta para o loop\n";
            break 1; #Utilizando o argumento
        case 3:
            echo "sai do switch e do loop\n";
            break 2;
        default:
            break;
    }
endforeach;

?>
```

# Continue

- “Pula” para a próxima iteração no loop
- Comportamento similar ao do *break*

```
<?php
$a = array(
    "um" => 1,
    "dois" => 2,
    "3" => 3,
    "vinte" => 20
);
foreach ($a as $n) {
    if($n == 3)
        continue;
    echo "$n\n";
}
?>
```

# Detalhes sobre o switch

- Um exemplo de switch é demonstrado no slide 45
- Somente tipos simples podem ser usados como chave
  - Números (int ou float)
  - strings
- O *continue* tem o mesmo efeito do *break*;
- Pode-se usar o ; no lugar do : ao final de um case

```
case 'valor';  
    # alguma coisa  
    continue;
```

# Goto

- Utilizado para pular para outros trechos do programa
- Não é completamente irrestrito
  - Pode ser usado para pular de switchs e laços
  - Não pode ser usado para pular para fora nem para dentro de funções e métodos

```
<?php
    goto a;
    echo 'Foo';

a:
    echo 'Bar';

?>
```



# Modularização

# Funções

- Nomes de funções seguem as mesmas regras que outros rótulo no PHP (slide 9)
- As funções não precisam ser criadas antes de serem referenciadas
  - Exceto quando é definida condicionalmente
- Funções são declaradas com **function**, seguido (ou não) do nome da função e os parâmetros entre parênteses
- Funções podem ser anônimas
  - Mais sobre o assunto no slide 61

```
<?php
$a = 0;
$nome_func = 'a';
$nome_func($a);
function a($b = 2){
    if($b == 1){
        function c(){
            echo "Igual\n";
        }
    }elseif($b < 1){
        function c(){
            echo "Menor\n";
        }
    }else{
        function d(){
            echo "Maior\n";
        }
    }
}

c();
a();
d();
?>
```

# Detalhes...

- 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, feita por referência.
- Admite valores padrões de argumentos e lista de argumentos de tamanho variável.
  - Os argumentos são passados na forma de array
- Argumentos-padrão deverão ser os últimos argumentos de uma função

# Include e require

- A declaração *include* inclui e avalia o arquivo informado
- Os arquivos são incluídos baseando-se no caminho do arquivo informado ou, se não informado, o **include\_path** especificado
- Se o arquivo não for encontrado no `include_path` (ou no caminho definido), a declaração *include* checará no diretório do script que o executa e no diretório de trabalho corrente
  - Caso não encontre o *include* emitirá um aviso, já o *require* emitirá um erro fatal
- *include\_once* e *require\_once* se diferenciam por incluírem o arquivo só uma vez, retornando **TRUE** para tentativas posteriores

# Incluído e escopos

- Quando um arquivo é incluído, o código herda o escopo de variáveis da linha que a inclusão ocorrer
- Todas as funções e classes definidas no arquivo incluído estarão no escopo global
- Se a inclusão ocorrer dentro de uma função, todo o código contido no arquivo incluído irá se comportar como se tivesse sido definido dentro da função
- Quando um arquivo é incluído, o interpretador sai do modo PHP e entra no modo HTML no começo do arquivo incluído, e volta novamente no final
  - Por esta razão, qualquer código dentro do arquivo incluído que necessite ser executado como PHP deve estar entre tags válidas de início e fim do PHP

# Exemplo

## teste.php

```
<?php
$a = 2;
function imprime(){
    echo "Teste\n";
}
?>
```

## outro.php

```
<?php
$b = 3;
function imprime(){
    echo "Outro\n";
}
?>
```

## include.php

```
<?php

function outro(){
    include_once 'outro.php';
    echo "$b\n"; # 3
    imprime();
}

$a = 1;
echo "$a\n"; # 1
include_once 'teste.php'; # Incluido
echo "$a\n"; # 2
imprime(); #Teste
outro();
echo "$b\n"; # Imprime nada
?>
```

# Classes

- Segue as regras de nomenclatura de variáveis (slide 9)
- Classes PHP tem a pseudo-variável **`$this`** disponível quando chamada a partir de um contexto de objeto
  - `$this` é uma referência ao objeto chamado
- Para criar uma instância de uma classe, a instrução **`new`** deve ser utilizada
  - Classes devem ser definidas antes de instanciadas
- Se uma string contendo o nome da classe é utilizada com `new`, uma nova instância da classe será criada
  - Se a classe estiver dentro de um namespace, o nome completo e qualificado deve ser utilizado para fazer isso (slide 64)
- PHP permite a criação de classes anônimas e permite a criação de classes com *final*



# Construção da classe

```
<?php
class Teste{
    private $a;

    public function Teste($b){
        $this->a = $b;
        echo "$this->a\n";
    }
    public function __construct($b){
        $this->a = $b;
        echo "$this->a entos\n";
    }
}

$nome_da_classe = "Teste";
$c = new $nome_da_classe(3); #Imprime 3 entos
?>
```

# Classe com herança (e outras coisas)

```
<?php
class Pessoa{
    private $nome;
}
class Aluno extends Pessoa{
    private $matricula;
    public $cr;

    public function __construct($m, $n){
        $this->matricula = $m;
        $this->nome = $n;
        $this->cr = function(){
            echo "$this->nome ($this->matricula): 4.0\n";
        };
    }
}

$aluno = new Aluno(2014100477, "Lucas Scopes Fazola");
($aluno->cr()); #Imprime Lucas Scopes Fazola (2014100477): 4.0
?>
```

# Visibilidade

- A visibilidade de uma propriedade ou método pode ser definida prefixando a declaração com as palavras-chave: `public`, `protected` or `private`.
- Itens declarados como públicos podem ser acessados de qualquer lugar.
- Membros declarados como protegidos só podem ser acessados na classe declarante e suas classes herdeiras.
- Membros declarados como privados só podem ser acessados na classe que define o membro privado.
- Caso a privacidade não seja declarada, o membro é considerado público

```

<?php
/**
 * Define MinhaClasse
 */
class MinhaClasse
{
    public $publica = 'Public'."\n";
    protected $protegida = 'Protected'."\n";
    private $privada = 'Private'."\n";

    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

```

```

/**
 * Define MinhaClasse2
 */
class MinhaClasse2 extends MinhaClasse
{
    // Nós podemos redeclarar as propriedades públicas e
    protegidas mas não as privadas
    protected $protegida = 'Protected2'."\n";

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

$obj2 = new MinhaClasse2();
echo $obj2->publica; // Funciona
# echo $obj2->privada; // Indefinida
# echo $obj2->protegida; // Fatal Error
$obj2->imprimeAlo(); // Mostra Public, Protected2, Indefinida

?>

```

# Closures

- PHP oferece suporte à *closures* através da classe pré-definida Closure
- Também é possível criar funções Lambda

```
<?php
```

```
$var = function ( $a ) {  
    echo "$a\n";  
    $b = function ( $x ) use ( $a ){  
        echo $x + $a . "\n";  
    };  
    $b(2);  
};
```

```
$var(1)
```

```
?>
```

# Classes abstratas

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

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

class ClasseConcreta1 extends ClasseAbstrata
{
    protected function pegarValor() {
        return "ClasseConcreta1";
    }

    public function valorComPrefixo( $prefixo ) {
        return "{$prefixo}ClasseConcreta1";
    }
}
```

```
class ClasseConcreta2 extends ClasseAbstrata
{
    protected function pegarValor() {
        return "ClasseConcreta2";
    }

    public function valorComPrefixo( $prefixo ) {
        return "{$prefixo}ClasseConcreta2";
    }
}

$classe1 = new ClasseConcreta1;
$classe1->imprimir();
echo $classe1->valorComPrefixo('FOO_') ."\n";

$classe2 = new ClasseConcreta2;
$classe2->imprimir();
echo $classe2->valorComPrefixo('FOO_') ."\n";
?>
```

# Traits

- Maneira encontrada para reutilizar código, tentando simular heranças múltiplas

```
<?php
class Hello {
    public function printHello() {
        echo 'Hello ';
    }
}

trait World {
    public function printHello() {
        parent::printHello();
        echo 'World!.'.\n";
    }
}

class HelloWorld extends Hello {
    use World;
}

$o = new HelloWorld();
$o->printHello();
?>
```

# Namespaces

- São declarados usando a *keyword namespace*
- Só afeta classes, interfaces, funções e constantes
- *Namespaces* PHP e php são reservados

```
<?php
namespace Pessoa\Academico;
class Aluno{
    public function hello()
    {
        echo "Sou aluno!\n";
        echo __NAMESPACE__."\n";
    }
}
```

```
<?php
namespace Pessoa\Academico;
class Professor{
    public function hello()
    {
        echo "Sou professor!\n";
        echo __NAMESPACE__."\n";
    }
}
```

```
<?php
use Pessoa\Academico\Aluno as A;
use Pessoa\Academico\Professor as P;
require_once 'aluno.php';
require_once 'professor.php';
$a = new A();
$a->hello(); #Sou aluno
Pessoa\Academico
$a = new P();
$a->hello(); #Sou professor
Pessoa\Academico
?>
```



# Polimorfismos

# Coerção

```
<?php
class Operacao{
    var $valor1;
    var $valor2;
    public function setValores($valor1,$valor2){
        $this->valor1 = $valor1;
        $this->valor2 = $valor2;
    }
    public function somaValores(){
        $resultado = $this->valor1 + $this->valor2;
        return $resultado;
    }
}
```

```
public function verificaValores(){
    if(is_int($this->valor1))
        echo 'o valor de $valor1 é inteiro.'."\n";
    else
        echo 'o valor de $valor1 não é inteiro.'."\n";

    if(is_int($this->valor2))
        echo 'O valor de $valor2 é inteiro.'."\n";
    else
        echo 'O valor de $valor2 não é inteiro.'."\n";
    }
}

$operacao = new Operacao();//instância da classe operacao
$operacao->setValores(5,4);
$operacao->verificaValores();//os dois valores serão retornados como inteiros
echo "Resultado da operação: ".$operacao->somaValores()."\n";//imprime 9

$operacao->setValores("5",4);//aqui $valor1 está recebendo uma string
$operacao->verificaValores();//será impresso 'o valor de $valor1 não é inteiro.'
//será impresso 9... A string foi convertida para inteiro automaticamente
echo "Resultado da operação: ".$operacao->somaValores()."\n";
?>
```

# Paramétrico

```
<?php
class Operacoes
{
    function soma($number1, $number2)
    {
        return $number1 + $number2;
    }
}

class OperacoesPorInclusao extends Operacoes
{
    function polimorfismoParametrico(Operacoes $op)
    {
        echo $op->soma(1, 2) . "<br />";
    }
}
```

```
$obj = new Operacoes();
$obj2 = new OperacoesPorInclusao();
$obj3 = new OperacoesPorInclusao();

echo $obj2->polimorfismoParametrico($obj3);
?>
```

# Inclusão

```
<?php
class Usuario
{
    var $nome;
    var $cpf;

    public function __Construct($nome,$cpf)
    {
        $this->nome = $nome;
        $this->cpf = $cpf;
    }

    public function getUsuario()
    {
        return "nome: ".$this->nome."cpf : ".$this->cpf;
    }

    public function imprime()
    {
        echo $this->nome."--".$this->cpf."\n";
    }
}
```

```
class Aluno extends Usuario
{
    var $codigo;

    public function
__Construct($nome,$cpf,$codigo)
    {
        parent::__Construct($nome,$cpf);
        $this->codigo = $codigo;
    }

    public function getAluno()
    {
        return
parent::getUsuario()."codigo:
".$this->codigo;
    }
}
```

```
public function imprime()
{
    echo "funcao imprime pai: ";
    parent::imprime();
    echo "<br>funcao imprime filho:" . $this->nome
. "--" . $this->cpf . "--" . $this->codigo . "\n";
}
}
/*
O polimorfismo por inclusao funciona, pois consigo
chamar o método imprime da classe pai dentro do
método imprime da classe filho.
*/
$aluno = new Aluno("Tiago",123456,40356788);
$aluno->imprime();

/*
resultado impresso:

funcao imprime pai: Tiago--123456
funcao imprime filho: Tiago--123456--40356788
*/
?>
```

# Sobrecarga

- Não existe o polimorfismo de sobrecarga em PHP assim como podemos ver em C++ e Java
- Pode ser simulado através dos chamados métodos mágicos, para “sobrecarregar” métodos e a implementação da interface *ArrayAccess* para o operador “[ ]”.
- A linguagem utiliza a sobrecarga de operadores, mas para o programador sobrecarregar um operador existe uma extensão, mas está desatualizada.

# Sobrecarga

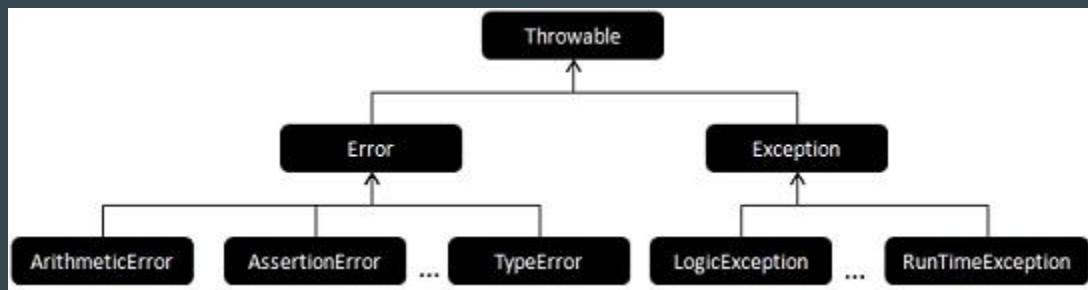
```
<?php
class test{
public function __construct(){
}
public function __call($nome , $parametro){
if($nome == "metodosobrecarregado"){
$count = count($parametro);
switch($count){
case "1":
echo "Voce passou 1 parametro";
break;
case "2":
echo "Voce passou 2 parametros";
break;
default:
throw new exception("Argumento Invalido");
}
}
}
```

```
else{
throw new exception("Funcao $nome nao existe");
}
}
}
$a = new test();
$a->metodosobrecarregado("argumento1"); // "Voce passou 1 parametro"
$a->metodosobrecarregado("argumento1" , "argumento2"); // "Voce passou 2 parametros"
?>
```

# Exceções

# Exceções

- Até o PHP5 só existia a classe *Exception*.
- Com o PHP7 foi criada a classe *Error* que se refere aos erros fatais, esta classe não herda *Exception*, que ficou com as exceções de usuário.
- Para conseguir capturar os dois tipos de exceções é necessário capturar *Throwable*, mas só funciona no PHP7 aonde *Exception* e *Error* herdam *Throwable*.





# Exceções

- Para manipular exceções são utilizados os blocos Try, catch e finally, sendo obrigatório o uso do finally, mas ele pode ser utilizado para substituir um bloco catch.
- Finally sempre será executado independente se houve o lançamento de uma exceção, e antes que a execução normal continue.
- Para tratar exceções da classe *Error* não pegadas pelo bloco try/catch de maneira diferente da definida pelo PHP é necessário definir a função *set\_exception\_handler()*.
- Alguns Errors simplesmente não podem ser tratados por se tratarem de problemas como falta de memória etc.

# Concorrência

- Não é nativo da linguagem
- Utiliza a extensão *Pthreads* e não pode ser utilizada no contexto de um servidor web, além do que só está disponível na versão PHP 7.2+

# Expandindo

# Tipo recurso

- Um recurso é uma variável especial, que mantém uma referência a um recurso externo.
- Recursos são criados e usados por funções especiais.

ftp      ftp\_connect(), ftp\_mkdir()

mysql    mysql\_connect(), mysql\_create\_db()

ldap     ldap\_connect(), ldap\_search()

pdf      pdf\_open\_image()

# Conexão - MySQL - 00

```
<?php
```

```
$servername = "localhost";
```

```
$username = "dovah";
```

```
$password = "atam_edadisoiruc";
```

```
// Cria conexão
```

```
$conn = new mysqli($servername, $username, $password);
```

```
// Checa conexão
```

```
if ($conn->connect_error) {
```

```
    die("Conexao falhou: " . $conn->connect_error . "\n");
```

```
}
```

```
echo "Conectado com sucesso!\n";
```

```
?>
```

# Avaliação de Linguagens

<b>CrITÉrios gerais</b>	<b>C</b>	<b>C++</b>	<b>Java</b>	<b>PHP</b>
<b>Aplicabilidade</b>	Sim	Sim	Parcial	<b>Parcial</b>
<b>Confiabilidade</b>	Não	Não	Sim	<b>Parcial</b>
<b>Aprendizado</b>	Não	Não	Não	<b>Sim</b>
<b>Eficiência</b>	Sim	Sim	Parcial	<b>Sim</b>
<b>Portabilidade</b>	Não	Não	Sim	<b>Sim</b>
<b>Método de projeto</b>	Estruturado	Estruturado e OO	OO	<b>Estruturado, OO, funcional</b>
<b>Evolutibilidade</b>	Não	Parcial	Sim	<b>Sim</b>
<b>Reusabilidade</b>	Parcial	Sim	Sim	<b>Sim</b>
<b>Integração</b>	Sim	Sim	Parcial	<b>Sim</b>

# Avaliação de Linguagens

<b>Crítérios Específicos</b>	<b>C</b>	<b>C++</b>	<b>Java</b>	<b>PHP</b>
<b>Escopo</b>	Sim	Sim	Sim	<b>Sim</b>
<b>Expressões e comandos</b>	Sim	Sim	Sim	<b>Sim</b>
<b>Tipos primitivos e compostos</b>	Sim	Sim	Sim	<b>Sim</b>
<b>Gerenciamento de memória</b>	Programador	Programador	Sistema	<b>Programador/Sistema</b>
<b>Persistência dos dados</b>	Biblioteca de funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização	<b>Biblioteca de classe e serialização</b>
<b>Passagem de parâmetros</b>	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável, por valor e por cópia de referência	<b>Por valor, referência, valores padrão de argumentos e lista variável</b>

# Avaliação de Linguagens

<b>Critérios Especificos</b>	<b>C</b>	<b>C++</b>	<b>Java</b>	<b>PHP</b>
<b>Encapsulamento e proteção</b>	Parcial	Sim	Sim	Sim
<b>Sistema de tipos</b>	Não	Parcial	Sim	<b>Não</b>
<b>Verificação de tipos</b>	Estática	Estática / Dinâmica	Estática / Dinâmica	<b>Dinâmica</b>
<b>Polimorfismo</b>	Coerção e sobrecarga	Todos	Todos	<b>Não possui sobrecarga de função</b>
<b>Exceções</b>	Não	Parcial	Sim	<b>Sim</b>
<b>Concorrência</b>	Não (biblioteca de funções)	Não (biblioteca de funções)	Sim	<b>Parcial(biblioteca de funções para multiThread)</b>



# Referências

- [https://secure.php.net/manual/pt\\_BR/index.php](https://secure.php.net/manual/pt_BR/index.php)
- <https://www.w3schools.com/php/>
- <https://pt.wikipedia.org/wiki/PHP>