

Primeiros Passos em PHP

Parte II

Thiago H. P. Silva

Execução

- LAMP: Linux + Apache + Mysql + PHP
- WAMP: Windows + Apache + Mysql + PHP
 - https://www.apachefriends.org/pt_br/download.html
- Rodar os programas online
 - <https://paiza.io/>

```
<?php  
    echo "oi";  
?>
```

Functions

```
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
    echo "Example function.\n";
    return $retval;
}
?>
```

Functions

```
<?php
```

```
testando();
```

```
function testando() {  
    echo "Oie :)";  
}
```

```
?>
```

Functions

```
<?php
```

```
testando();
```

```
function testando() {  
    echo "Oie :)";  
}
```

```
?>
```

```
<?php
```

```
testando();
```

```
if (true) {  
    function testando() {  
        echo "Oie :)";  
    }  
}
```

```
?>
```

Functions

```
<?php  
testando();  
  
function testando() {  
    echo "Oie :)";  
}  
?>
```



Existe assim que o programa se inicia

```
<?php  
testando();  
  
if (true) {  
    function testando() {  
        echo "Oie :)";  
    }  
}  
?>
```



Existe apenas quando o programa o atinge

Functions

- Cria uma função recursiva que recebe um número inteiro maior que um e calcule $n!$
 - $4! = 4 * 3 * 2 * 1 = 24$

Functions

- Cria uma função recursiva que recebe um número inteiro maior que um e calcule $n!$
 - $4! = 4*3*2*1 = 24$

```
<?php

function factorial($n) {
    if ($n < 2) {
        return 1;
    } else {
        return ($n * factorial($n-1));
    }
}

print(factorial(4));

?>
```


Functions

- Cria uma função recursiva que recebe um número inteiro maior que um e calcule $n!$
 - $4! = 4 * 3 * 2 * 1 = 24$

Functions

```
1 <?php
2
3 function func($arg1, $arg2) {
4     echo $arg1 . " - " . $arg2 . "<br>";
5 }
6
7 func(5,6);
8
9 ?>
```

Functions



The image shows a web browser window with the address bar displaying 'localhost/helloWorld.php'. The main content area shows the output '5 - 6'. Below the content area is a dark toolbar with an 'Open' button, a dropdown arrow, and a file icon. To the right of the toolbar, the filename 'helloWorld.php' and its path '/var/www/html' are displayed. The bottom section of the browser shows the source code of the PHP file, which defines a function 'func' and calls it with arguments 5 and 6.

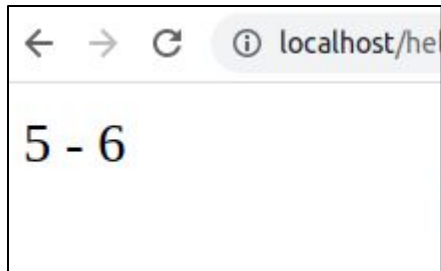
```
1 <?php
2
3 function func($arg1, $arg2) {
4     echo $arg1 . " - " . $arg2 . "<br>";
5 }
6
7 func(5,6);
8
9 ?>
```

Functions

```
<?php  
  
function func($arg1, $arg2) {  
    echo $arg1 . " - " . $arg2 . "<br>";  
}  
  
func(5,6);  
  
func(5);  
  
?>
```

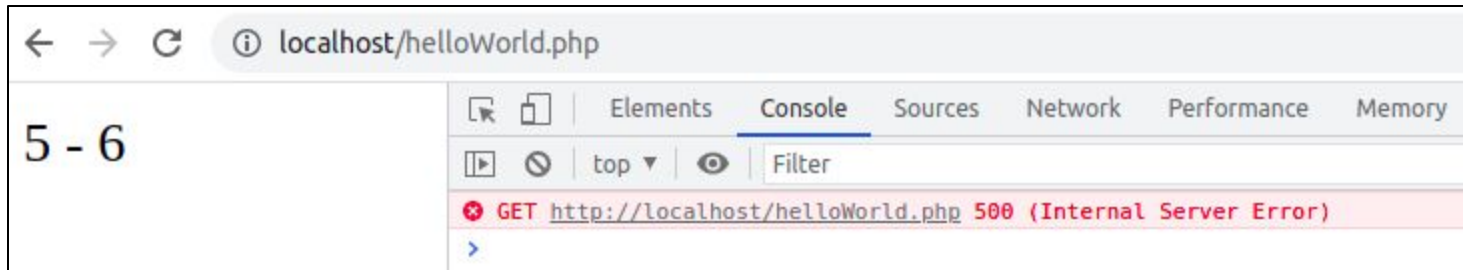
Functions

```
<?php  
  
function func($arg1, $arg2) {  
    echo $arg1 . " - " . $arg2 . "<br>";  
}  
  
func(5,6);  
  
func(5);  
  
?>
```



Functions

```
<?php  
  
function func($arg1, $arg2) {  
    echo $arg1 . " - " . $arg2 . "<br>";  
}  
  
func(5,6);  
  
func(5);  
  
?>
```



Functions

```
ini_set('display_errors', 1);  
ini_set('display_startup_errors', 1);  
error_reporting(E_ALL);
```

← → ↻ ⓘ localhost/helloWorld.php

5 - 6

Fatal error: Uncaught ArgumentCountError: Too few arguments to function func(), 1 passed in /var/www/html/helloWorld.php on line 13 and exactly 2 expected in /var/www/html/helloWorld.php:7 Stack trace: #0 /var/www/html/helloWorld.php(13): func() #1 {main} thrown in /var/www/html/helloWorld.php on line 7

Functions



The screenshot shows a web browser window with the address bar displaying 'localhost/h'. The page content shows the output of a PHP script: '5 - 6'. The PHP code is displayed in a code editor on the right side of the browser window. The code defines a function 'func' that takes two arguments, '\$arg1' and '\$arg2', and echoes them separated by a hyphen and a space, followed by a line break. The function is called twice: first with arguments 5 and 6, and then with argument 5. The output of the first call is '5 - 6'.

```
<?php
function func($arg1, $arg2) {
    echo $arg1 . " - " . $arg2 . "<br>";
}

func(5,6);

func(5);

?>
```

5 - 6

Performance Memory

al Server Error)

Functions

5 - 6

5 - teste

```
1 <?php
2
3 function func($arg1, $arg2 = "teste") {
4     echo $arg1 . " - " . $arg2 . "<br>";
5 }
6
7 func(5,6);
8
9 func(5);
10
11 ?>
```

Functions



The image shows a code editor window with a dark theme. The title bar at the top right says "helloWorld.php" and the path below it is "/var/www/html". The editor contains PHP code with line numbers 1 through 12. The code defines a function "func_style" with three parameters: \$A, \$B, and \$C. The function returns a string formatted as "\$A, \$B e \$C.". Below the function definition, there are four "echo" statements that call the function with different arguments: no arguments, 1 argument, 2 arguments, and 3 arguments. The code is syntax-highlighted, with keywords in red, variables in blue, and strings in purple.

```
1 <?php
2
3 function func_style($A = "A", $B = "B", $C = "C")
4 {
5     return "$A, $B e $C.";
6 }
7
8 echo func_style() . "<br>";
9 echo func_style(1) . "<br>";
10 echo func_style(1, 2) . "<br>";
11 echo func_style(1, 2, "BUH") . "<br>";
12 ?>
```

Functions

← → ↻ ⓘ localhost/helloWorld.php

A, B e C.

1, B e C.

1, 2 e C.

1, 2 e BUH.

Open



helloWorld.php
/var/www/html

```
1 <?php
2
3 function func_style($A = "A", $B = "B", $C = "C")
4 {
5     return "$A, $B e $C.";
6 }
7
8 echo func_style() . "<br>";
9 echo func_style(1) . "<br>";
10 echo func_style(1, 2) . "<br>";
11 echo func_style(1, 2, "BUH") . "<br>";
12 ?>
```

Functions

← → ↻ ⓘ localhost/helloWorld.php

A, B e C.

1, B e C.

1, 2 e C.

1, 2 e BUH.

Open



helloWorld.php
/var/www/html

```
1 <?php
2
3 function func_style($A = "A", $B = "B", $C = "C")
4 {
5     return "$A, $B e $C.";
6 }
7
8 echo func_style() . "<br>";
9 echo func_style(1) . "<br>";
10 echo func_style(1, 2) . "<br>";
11 echo func_style(1, 2, "BUH") . "<br>";
12 ?>
```


Functions



The image shows a code editor window with a dark theme. The title bar at the top right says 'helloWorld.php' and the path below it is '/var/www/html'. The editor contains PHP code with line numbers 1 through 12. The code defines a function 'func_style' that takes three arguments: \$A, \$B, and \$C. The function returns a string formatted as '\$A, \$B e \$C.'. Below the function definition, there are four 'echo' statements that call the function with different arguments, each followed by a line break '
'. The code is syntax-highlighted: keywords like 'function', 'return', 'echo', and '?>' are in red; variables and strings are in blue or purple; and the opening PHP tag '<?php' is in orange.

```
1 <?php
2
3 function func_style($A = "A", $B = "B", $C = "C")
4 {
5     return "$A, $B e $C.";
6 }
7
8 echo func_style(C: "oie") . "<br>";
9 echo func_style(B: "2222") . "<br>";
10 echo func_style(A: "AAA", C: "CCC") . "<br>";
11 echo func_style(C: "CCC", A: "AAA") . "<br>";
12 ?>
```

Functions



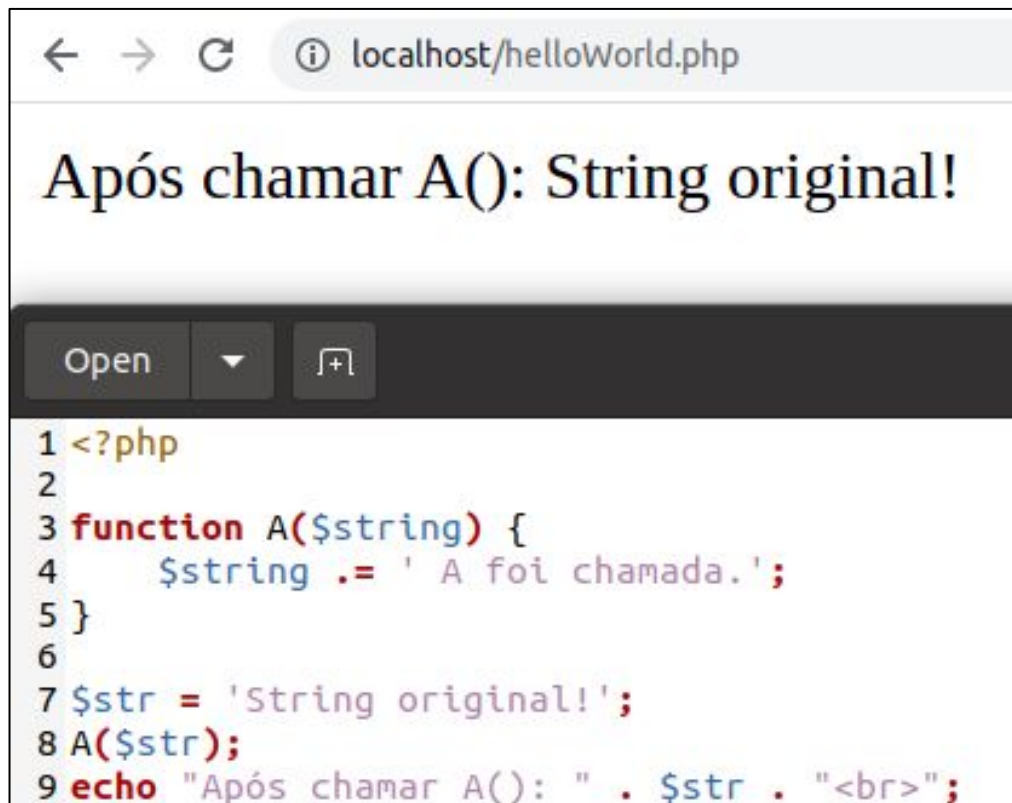
The image shows a web browser window with the address bar displaying 'localhost/helloWorld.php'. The browser content area shows the output of a PHP script, which consists of four lines of text: 'A, B e oie.', 'A, 2222 e C.', 'AAA, B e CCC.', and 'AAA, B e CCC.'. Below the browser window is a dark-themed code editor showing the source code of the file 'helloWorld.php' located at '/var/www/html'. The code defines a function 'func_style' that takes three arguments (\$A, \$B, \$C) and returns a string formatted as '\$A, \$B e \$C.'. The function is then called four times using 'echo' statements, each time with different arguments and a trailing '
' to create new lines in the browser output.

```
1 <?php
2
3 function func_style($A = "A", $B = "B", $C = "C")
4 {
5     return "$A, $B e $C.";
6 }
7
8 echo func_style(C: "oie") . "<br>";
9 echo func_style(B: "2222") . "<br>";
10 echo func_style(A: "AAA", C: "CCC") . "<br>";
11 echo func_style(C: "CCC", A: "AAA") . "<br>";
12 ?>
```

Functions

```
1 <?php
2
3 function A($string) {
4     $string .= ' A foi chamada.';
5 }
6
7 $str = 'String original!';
8 A($str);
9 echo "Após chamar A(): " . $str . "<br>;
```

Functions



The image shows a web browser window with the address bar displaying 'localhost/helloWorld.php'. The main content area shows the text 'Após chamar A(): String original!'. Below the browser window, the PHP source code is displayed with line numbers 1 through 9. The code defines a function 'A' that appends a string to a parameter and then echoes the result.

```
1 <?php
2
3 function A($string) {
4     $string .= ' A foi chamada.';
5 }
6
7 $str = 'String original!';
8 A($str);
9 echo "Após chamar A(): " . $str . "<br>;
```


Functions

```
1 <?php
2
3 function A($string) {
4     $string .= ' A foi chamada.';
5 }
6
7 $str = 'String original!';
8 A($str);
9 echo "Após chamar A(): " . $str . "<br>";
10
11 function B(&$string) {
12     $string .= ' B foi chamada.';
13 }
14
15 B($str);
16 echo "Após chamar B(): " . $str . "<br>";
17
18 ?>
```

Functions



The screenshot shows a web browser window with the address bar displaying 'localhost/helloWorld.php'. The browser content area shows the output of the PHP script: 'Após chamar A(): String original!' followed by 'Após chamar B(): String original! B foi chamada.' on a new line. Below the browser window is a code editor showing the source code of 'helloWorld.php' located at '/var/www/html'. The code defines two functions, A and B, and calls them in sequence. Function A appends ' A foi chamada.' to the string, and function B appends ' B foi chamada.' to the string. The output is displayed using 'echo' statements with line breaks.

```
1 <?php
2
3 function A($string) {
4     $string .= ' A foi chamada.';
5 }
6
7 $str = 'String original!';
8 A($str);
9 echo "Após chamar A(): " . $str . "<br>";
10
11 function B(&$string) {
12     $string .= ' B foi chamada.';
13 }
14
15 B($str);
16 echo "Após chamar B(): " . $str . "<br>";
17
18 ?>
```

Functions

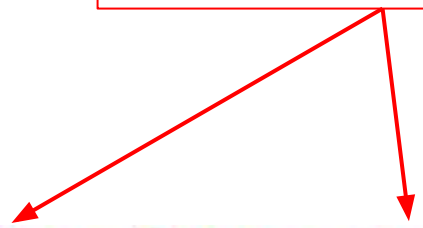
```
<?php
function add($a, $b) {
    return $a + $b;
}

echo add(...[1, 2])."\n";

$a = [1, 2];
echo add(...$a);
?>
```

Functions

Definindo a tipagem dos dados



```
5 function func(string $nome, int $idade) {  
6     echo "Nome: " . $nome . "<br>";  
7     echo "Idade: " . $idade . "<br>";}  
8  
9 echo func("Allan", 5) . "<br>";  
10 echo func("Allan", "5") . "<br>";  
11 echo func("Allan", 1.2) . "<br>";  
12 echo func("Allan", array(1,2)) . "<br>";
```

Nome: Allan
Idade: 5

Nome: Allan
Idade: 5

Nome: Allan
Idade: 1

Fatal error: Uncaught TypeError: func(): Argument #2 (\$idade) must be of type int, array given, called in /var/www/html/helloWorld.php on line 12 and defined in /var/www/html/helloWorld.php:5 Stack trace: #0 /var/www/html/helloWorld.php(12): func() #1 {main} thrown in **/var/www/html/helloWorld.php** on line 5

Functions

```
1 <?php
2 function contar_args(...$args) {
3     echo "Contagem: " . count($args) . "<br>";
4     var_dump($args);
5     echo "<br>-----<br>";
6 }
7
8 echo contar_args(1) . "<br>";
9 echo contar_args() . "<br>";
10 echo contar_args(NULL) . "<br>";
11 echo contar_args(1, 2, "3", 4, "aaa") . "<br>";
12 ?>
```

Functions

The image shows a web browser window on the left and a code editor on the right. The browser displays the output of a PHP script at `localhost/helloWorld.php`. The output consists of four blocks, each starting with "Contagem: " followed by a number, then an array representation, and a separator line of five dashes.

The code editor on the right shows the PHP code for the `contar_args` function. The function uses `count($args)` to count the number of arguments and `var_dump($args)` to display their types and values. The function is called with various arguments, including `1`, `NULL`, and a mix of integers and strings.

Browser Output:

```
Contagem: 1
array(1) { [0]=> int(1) }
-----

Contagem: 0
array(0) { }
-----

Contagem: 1
array(1) { [0]=> NULL }
-----

Contagem: 5
array(5) { [0]=> int(1) [1]=> int(2) [2]=> string(1) "3"
[3]=> int(4) [4]=> string(3) "aaa" }
-----
```

Code Editor Content:

```
1 <?php
2 function contar_args(...$args) {
3     echo "Contagem: " . count($args) . "<br>";
4     var_dump($args);
5     echo "<br>-----<br>";
6 }
7
8 echo contar_args(1) . "<br>";
9 echo contar_args() . "<br>";
10 echo contar_args(NULL) . "<br>";
11 echo contar_args(1, 2, "3", 4, "aaa") . "<br>";
12 ?>
13 |
```

Functions

- Return

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


Functions

- `return $arg1, $arg2;` → **ERROR**
- Outras linguagens como **Python** permitem
- O que fazer quando precisamos retornar vários dados?

```
function small_numbers()  
{  
    return [0, 1, 2];  
}
```

```
// Array destructuring will collect each member of the array individually  
[$zero, $one, $two] = small_numbers();
```

Functions

- O que fazer quando precisamos retornar valores gigantescos?

Functions

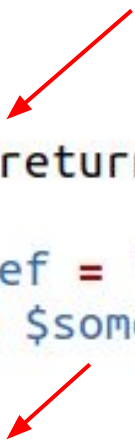
- O que fazer quando precisamos retornar valores gigantescos?

```
1 <?php
2
3 function &returns_reference()
4 {
5     $someref = "Muitas dados processados aqui!";
6     return $someref;
7 }
8
9 $newref =& returns_reference();
10
11 echo $newref;
12
13 ?>
```

Functions

- O que fazer quando precisamos retornar valores gigantescos?

```
1 <?php
2
3 function &returns_reference()
4 {
5     $someref = "Muitas dados processados aqui!";
6     return $someref;
7 }
8
9 $newref =& returns_reference();
10
11 echo $newref;
12
13 ?>
```

Two red arrows are present. The first arrow points from the top right towards the function definition line 3, specifically to the ampersand in 'function &returns_reference()'. The second arrow points from the top right towards the usage line 9, specifically to the ampersand in '\$newref =& returns_reference();'.

Functions

```
1 <?php
2 |
3 function A($argA) {
4     echo "A: $argA<br>";
5 }
6
7 function B($argB) {
8     echo "B: $argB<br>";
9 }
10
11 $c = 'A';
12 $c("Oie!");
13 $c = 'B';
14 $c("Olá!");
15
16 ?>
```

Functions



localhost/helloWorld.php

A: Oie

B: Olá

```
1 <?php
2
3 function A($argA) {
4     echo "A: $argA<br>";
5 }
6
7 function B($argB) {
8     echo "B: $argB<br>";
9 }
10
11 $c = 'A';
12 $c("Oie!");
13 $c = 'B';
14 $c("Olá!");
15
16 ?>
```

Functions

```
<?php
$greet = function($name)
{
    printf("Hello %s\r\n", $name);
};

$greet('World');
$greet('PHP');
?>
```

Functions

- Arrow Functions: **fn** (argument_list) => expr

Functions

- Arrow Functions: **fn** (argument_list) **=>** expr

```
<?php
$fn1 = fn($x, $y) => $x + $y;
echo $fn1(3,5);
?>
```

Functions

- Arrow Functions: **fn** (argument_list) => expr

```
<?php
$z = 1;
$fn = fn($x) => fn($y) => $x * $y + $z;
echo($fn(5)(10));
?>
```

Functions

1. Crie uma função que receba um argumento e imprima “Fazendo <valor do argumento>!”. Por padrão, imprima “Fazendo café!”.
 - Teste os seguintes cenários:
 - `echo makecoffee();`
 - `echo makecoffee(null);`
 - `echo makecoffee("espresso");`
2. Crie uma função para calcular o fatorial (não use recursão).
3. Crie uma função que receba vários argumentos e retorne a soma e a média deles.