

PHP Básico

- Função

Função

Sintaxe básica:

```
<?php
```

```
function nomeDaFuncao() {
```

```
    //código
```

```
}
```

Uma função é basicamente um bloco de instruções que podem ser utilizados repetidamente num programa.

Função

```
<?php
```

```
function escreverMensagem() {  
    echo "Olá Mundo!";  
}
```

```
echo escreverMensagem();  
// executar/chamar a função
```

Diferente das outras instruções básicas, uma função nunca será executada imediatamente quando uma página ou script é carregado.

Uma função só será executada apenas quando é feito uma chamada pela mesma.

Função (argumentos/parâmetros)

```
<?php
```

```
function imprimeNome($nome) {  
    echo "Meu nome é $nome !";  
}
```

```
imprimeNome("João");  
imprimeNome("Maria");
```

Uma informação pode ser passada para as funções por meio de argumentos. Um argumento é como uma variável (inclusive utilizamos o \$ antes do nome).

Argumentos são especificados após o nome da função, dentro dos parênteses.

Função (argumentos)

```
<?php
```

```
function imprimeNome($nome, $sobrenome) {  
    echo "Meu nome é $nome $sobrenome !";  
}
```

```
imprimeNome("João","Oliveira");  
imprimeNome("Maria","das Lurdes");
```

Podemos adicionar quantos argumentos quisermos, basta separá-los com uma vírgula.

Função (argumentos com valor padrão “opcional”)

```
<?php  
function imprimeNome($nome = "Fulano") {  
    echo "Meu nome é $nome !";  
}
```

```
imprimeNome();  
imprimeNome("João");  
imprimeNome("Maria");
```

Muitas vezes é necessário que alguns parâmetros de nossa função sejam opcionais.

Para isso, podemos definir valores padrões para os parâmetros que desejamos que se tornem opcionais.

Esta atribuição é feita através de forma muito similar a atribuição de um valor para uma variável usando o sinal de igual seguido do valor desejado.

Função (argumentos nomeados) (PHP 8.0 >=)

```
<?php
```

```
function imprimeNome($nome, $sobrenome, $pronomeTratamento = "") {  
    echo "Meu nome é $pronomeTratamento $nome $sobrenome  
    !".PHP_EOL;  
}
```

```
imprimeNome(nome:"João",sobrenome:"Oliveira");  
imprimeNome(sobrenome:"das Lurdes", nome:"Maria", pronomeTratamento:  
"Sra.");  
imprimeNome("Lucas",sobrenome:"Silva");  
//imprimeNome(nome:"Thiago"); //Erro !
```

- Com o advento do PHP 8.0 tornou-se possível passar parâmetros em ordem arbitrária através da indicação explícita do nome e valor desejado para cada argumento pré-definidos na assinatura de funções ou métodos;
- É possível combinar argumentos posicionais com argumentos nomeados, contanto que estes sejam o(s) últimos;

Lista de argumentos variável (Splat Operator "...")

```
<?php
```

```
function soma(...$numeros) {  
    echo array_sum($numeros);  
}
```

```
soma(1, 2, 3, 4);
```

A partir do PHP 5.6 tornou-se possível incluir o token reticências para indicar uma lista de argumentos e assim informar que a função aceita um número variável de argumentos.

Os argumentos serão passados na forma de um array e por isso, neste exemplo, utilizamos uma função do php chamada `array_sum` para somar os valores dentro deste array.

Argument Unpacking (Splat Operator "...")

```
<?php
```

```
function soma($a, $b, $c) {  
    echo $a + $b + $c;  
}
```

```
$args = array(2, 3);  
soma(1, ...$args);
```

O splat operator também pode ser utilizado para “desempacotar” um coleção em argumentos separados.

Função (return)

```
<?php
```

```
function soma($x, $y) {
```

```
    $z = $x + $y;
```

```
    return $z;
```

```
}
```

```
echo "5 + 10 = " . soma(5, 10)
```

```
;
```

O *return* permite que uma função possa retornar algum valor.

O *return* retorna o controle do programa para o trecho que executou a chamada.

Função Recursiva

- É basicamente uma função que chama a si mesma.
- Neste exemplo utilizaremos o resultado de cada iteração para multiplicar este resultado vezes ele mesmo menos 1 em uma cadeia de recursividade que será controlada por um “if” que irá garantir que a recursividade não seja eterna.
- **Atenção:** Chamadas de função / método recursivas com mais de 100 a 200 níveis de recursão podem estourar a pilha e causar o encerramento do script atual.
- A recursão infinita é considerada um erro de programação.

Função Recursiva

```
<?php
```

```
function fatorial($numero) {
```

```
    echo 'Numero atual'. $numero. ' ';
```

```
    if ($numero < 2) {
```

```
        return 1;
```

```
    } else {
```

```
        return ($numero * fatorial($numero-1));
```

```
    }
```

```
}
```

```
echo 'Resultado:'. fatorial(4);
```

$$= 4 * 3$$

$$= 4 * (3 * 2)$$

$$= 4 * (3 * (2 * 1))$$

$$= 4 * (3 * 2)$$

$$= 4 * 6$$

$$= 24$$

Função Recursiva

```
<?php
function fibonacci($n) {
    if ($n <= 0) {
        return 0; }
    elseif ($n == 1) {
        return 1; }
    else {
        return fibonacci($n - 1) + fibonacci($n - 2)
    }
}
for ($i = 0; $i < 10; $i++) {
    echo fibonacci($i) . " ";
}
```

$$F_n = F_{n-1} + F_{n-2}$$

$$1 + 1 = 2$$

$$2 + 1 = 3$$

$$3 + 2 = 5$$

$$5 + 3 = 8$$

$$8 + 5 = 13$$

$$13 + 8 = 21$$

$$21 + 13 = 34$$

$$34 + 21 = 55$$

$$55 + 34 = 89$$

Type Hint

```
<?php
```

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

```
echo soma(1,2);
```

Tipos usados no
TypeHint:

- bool
- float (php >=7)
- int (php >=7)
- string (php >=7)
- Classe/Interface
- callable
- self
- array
- mixed (php >= 8)

Type Hint + Splat Operator (...)

```
<?php
```

```
function soma(int ...$numeros) {  
    echo array_sum($numeros);  
}
```

```
soma(1, 2, 3, 4);
```

```
soma(1, 2, 3, 4, 5, 6, 7);
```

O *type hint* ainda pode ser usado em conjunto com o Splat Operator para especificar qual será o tipo dos valores de um determinado conjunto.

Função (declaração de tipo de retorno “:”)

```
<?php
```

```
function soma($a, $b): int {
```

```
    return $a + $b;
```

```
}
```

```
echo soma(1, 2.5);
```

A partir do PHP 7 também se tornou possível definir o tipo do valor do retorno de uma função.

Para utilizar este recurso, basta definir tipo da saída (retorno) antecedido pelo símbolo de dois pontos.

Valor vs Referência

Valor. Quando realizamos uma passagem de parâmetros baseada no valor, significa que uma função irá realizar suas operações baseado no valor do parâmetro informado para ela. Em outras palavras, o valor informado será **alterado apenas no escopo da função**. Essa é a maneira padrão de o PHP informar parâmetros.

Referência. Quando realizamos uma passagem de parâmetros baseada na referência, significa que uma função irá realizar suas operações baseado na referência original do parâmetro. Em outras palavras, se informarmos uma variável para uma função, o valor dessa variável será **alterada no escopo global**. Para realizar essa passagem, basta adicionar um & (“E” comercial) antes do parâmetro informado.

Passagem por referência “&”

```
<?php
```

```
function somar(&$var) {
```

```
    $var++;
```

```
}
```

```
$x=5;
```

```
somar($x);
```

```
echo $x;
```

Com o uso do e comercial, é possível determinar que um parâmetro possa modificar uma variável “mantendo a referência” para a mesma dentro do escopo da variável.

Valor vs Referência

```
<html>
<head>
  <title>Por Valor</title>
</head>
<body>
<?php
  $a=1;
  dobra($a);
  echo $a;

  function dobra($a){
    $a *= 2;
  }
?>
</body>
</html>
```

```
<html>
<head>
  <title>Por Referência</title>
</head>
<body>
<?php
  $a=1;
  dobra($a);
  echo $a;

  function dobra(&$a){
    $a *= 2;
  }
?>
</body>
</html>
```

Função Anônima (Closure)

```
<?php
```

```
$x = function ($txt) {  
    echo("Hello $txt !");  
};
```

```
$x('World');  
$x('PHP');
```

Funções anônimas, também conhecidas como *closures*, permitem a criação de funções que não tem o nome especificado.

Elas são mais úteis como o valor de parâmetros *callback*, mas podem tem vários outros usos.


Vale apenas salientar que toda função anônima atribuída a uma variável termina com ponto e vírgula.

Função Anônima (“use”)

```
<?php
```

```
$total = 40;
```

```
$soma = function ($x) use ($total) {  
    echo $x+$total;  
};
```

A diagram consisting of a horizontal line with an arrow pointing from the right towards the variable '\$total' in the line '\$total = 40;'. This indicates that the 'use' clause in the function definition refers to the variable defined in the scope above.

```
$soma(50);
```

O termo “use” permite a utilização de variáveis externas dentro do escopo de uma *closure*.

Função Anônima como parâmetro

```
<?php
```

```
function add($x,$y){  
    return $x+$y();  
}
```

```
echo add (3, function(){return 5;} );
```



Praticar...

- 1) Escreva uma função que recebe dois números como parâmetros e retorna a soma deles.
- 2) Crie uma função que receba um array de números como parâmetro e retorne o maior valor presente no array.
- 3) Implemente uma função que receba um string como parâmetro e retorne o número de caracteres presentes nessa string.
- 4) Crie uma função que receba uma string como parâmetro e retorne a mesma string, mas com todas as letras em maiúsculas.
- 5) Escreva uma função que receba um número inteiro como parâmetro e verifique se ele é par. A função deve retornar par se for true e ímpar se for false caso contrário.
- 6) Implemente uma função que receba um array de strings como parâmetro e retorne um novo array contendo apenas as strings com mais de 5 caracteres.
- 7) Crie uma função que receba uma data no formato "dd/mm/aaaa" como parâmetro e retorne a data no formato "aaaa-mm-dd".
- 8) Escreva uma função que receba um array de números como parâmetro e calcule a média aritmética dos valores. A função deve retornar o resultado.
- 9) Implemente uma função que receba um array de strings como parâmetro e retorne um novo array contendo as strings invertidas. Por exemplo, se o array for ["amor", "casa", "sol"], o resultado deve ser ["roma", "asac", "los"].
- 10) Crie uma função que receba uma frase como parâmetro e retorne a mesma frase, mas com todas as palavras em ordem inversa.