11. Funções definidas pelo usuário

Uma função pode ser definida usando a seguinte sintaxe demonstrada no quadro 48.

Quadro 48 – Exemplo de sintaxe de função

```
<?php
function fn1 ($arg_1, $arg_2, /* ..., */ $arg_n)
{
   echo "Exemplo de função.\n";
   return $valor_retornado;
}
?>
```

Qualquer código PHP válido pode aparecer dentro de uma função. Nomes de funções seguem as mesmas regras que outros rótulo no PHP. Um nome de função válido começa com uma letra ou um sublinhado, seguido, seguido por qualquer número de letras, números ou sublinhado. Nomes de funções são **case-insensitive** para caracteres ASCII A até Z, mas é recomendado chamar as funções da mesma forma que elas aparecem nas declarações.

As funções não precisam ser criadas antes de serem referenciadas, exceto quando uma função é condicionalmente definida como mostrado nos dois exemplos do quadro 49, neste caso, sua definição precisa ser processada antes de ser chamada.

Quadro 49 – Funções condicionalmente definidas

```
<?php

$makefoo = true;

/* Nós não podemos chamar foo() daqui
porque ela ainda não existe, mas nós podemos chamar bar() */

bar();

if ($makefoo) {
  function foo()
  {
```

```
    echo "Eu não existo até que o programa passe por aqui.\n";
  }
}

/* Agora nós podemos chamar foo()
   porque $makefoo foi avaliado como true */

if ($makefoo) foo();

function bar()
{
  echo "Eu existo imediatamente desde o programa começar.\n";
}
?>
```

Fonte PHP.net

Todas as funções e classes no PHP tem escopo global - elas podem ser chamadas fora de uma função mesmo que tenham sido definidas dentro e vice-versa.

O uso da palavra **return** somente é obrigatória caso a função precise retornar um valor, caso contrário pode ser omitida. No exemplo do quadro 49, as funções não retornam nenhum valor, apenas escrevem na tela.

O PHP não suporta sobrecarga de funções[1], e também não é possível cancelar ou alterar a definição de funções previamente declaradas.

O PHP suporta número variável de argumentos e argumentos padrões nas funções.

[1] Permite que sejam definidas duas ou mais funções com o mesmo nome, desde que suas listas de argumentos sejam suficientemente diferentes para as chamadas a serem resolvidas.

11.1. Número variável de argumentos

O PHP tem suporte para um número variável de argumentos nas funções definidas pelo usuário. Isso é implementado usando o token ... a partir do PHP 5.6, e usando as funções func_num_args(), func_get_arg(), e func_get_args() a partir do PHP 5.5.

O quadro 50 mostra um exemplo de uso variável de argumentos na função.

Quadro 50

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

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

Fonte PHP.net

O exemplo do quadro 50 escreve na tela o valor 10. Para mais detalhes e exemplos, consulte o manual do PHP no site php.net/manual.

11.2. Argumentos de funções

Informações podem ser passadas para funções através da lista de argumentos, que é uma lista de expressões delimitados por vírgulas. Os argumentos são avaliados da esquerda para a direita.

O PHP suporta a passagem de argumentos por valor (o padrão), passagem por referência, e valores padrões de argumentos. Lista de argumentos de tamanho variável também são suportadas (ver tópico 11.1). O quadro 51 mostra um exemplo de passagem de argumento por valor.

Quadro 51 – Passagem por valor

```
<?php declare(strict_types=1); // strict requirement ?>
<!DOCTYPE html>
<html>
<body>
<?php
function familyName(string $fname, int $year) {
  echo "$fname Silva. Nasceu em $year <br>";
}
familyName("Maria", 1975);
familyName("Marcos", 1978);
familyName("Matheus", 2003);
?>
</body>
</html>
```

Lembra da declaração declare(strict_types=1) (Tópico 4 deste material)? Com ela a **checagem de tipo será feita.**

11.3. Argumentos default

O exemplo do quadro 52 mostra como usar um parâmetro padrão. Se chamarmos a função familyName () sem o argumento ano, ela tomará o valor padrão como argumento, nesse caso o ano 1900.

Quadro 52

```
<?php declare(strict_types=1); // strict requirement ?>
<!DOCTYPE html>
<html>
<body>
<?php
function familyName(string $fname, int $year=1900) {
```

```
  echo "$fname Silva. Nasceu em $year <br>";
}
familyName("Maria", 1975);
familyName("Marcos", 1978);
familyName("Matheus", 2003);
familyName("Michel");
?>
</body>
</html>
```

Nesse exemplo do quadro 52 será impresso:

Maria Silva. Nasceu em 1975

Marcos Silva. Nasceu em 1978

Matheus Silva. Nasceu em 2003

Michel Silva. Nasceu em 1900

11.4. Retornando um tipo declarado

PHP 7 também suporta declarações de tipo para a instrução de retorno. Como acontece com a declaração de tipo para argumentos de função, ao habilitar o requisito estrito, ele lançará um "Erro fatal" em uma incompatibilidade de tipo. Para declarar um tipo para o retorno da função, adicione dois pontos (:) e o tipo logo antes do colchete ({) de abertura ao declarar a função. O exemplo do quadro 53 especifica o tipo de retorno para a função.

Quadro 53

```
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : float {
    return $a + $b;
}
echo addNumbers(1.2, 5.2);
?>
```

Você pode especificar um tipo de retorno diferente dos tipos de argumento, mas certifique-se de que o retorno seja do tipo correto, no exemplo do quadro 54, a função irá retornar um inteiro, apesar dos argumentos serem do tipo float.

Quadro 54 – Cast no retorno

```
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : int {
  return (int)($a + $b);
}
echo addNumbers(1.2, 5.2);
?>
```

Nesse exemplo, apesar de permitida a conversão de float para int, a parte decimal foi perdida, **pois o retorno da função será 6**.

11.5. Passagem de argumentos por referência

No PHP, os argumentos geralmente são passados por valor, o que significa que uma cópia do valor é usada na função e a variável que foi passada para a função não pode ser alterada. Quando um argumento de função é passado por referência, as alterações no argumento também mudam a variável que foi passada. Para transformar um argumento de função em uma referência, o operador **&** é usado. O exemplo do quadro 55 mostra uma função chamada contadora que adiciona 1 ao ser chamada.

Quadro 55

```
<?php
function contador(&$value) {
  $value += 1;
}
$num = 0;
contador($num);
echo $num;
?>
```

11.6. Função Callback

Uma função de retorno de chamada (frequentemente chamada apenas de “callback”) é uma função que é passada como um argumento para outra função. Qualquer função existente pode ser usada como função de retorno de chamada. Para usar uma função como função de retorno de chamada, passe uma string contendo o nome da função como o argumento de outra função, o quadro 56 mostra um exemplo de uso de uma função call-back.

Quadro 56 – Exemplo de função callback

```
<!DOCTYPE html>
<html>
<body>
<?php
function myfunction($num)
{
  return($num*$num);
}
$a=array(1,2,3,4,5);
print_r(array_map("myfunction",$a));
?>
</body>
</html>
```

Fonte w3schools