Universidade do Estado de Santa Catarina
Departamento de Sistemas de Informação

6WEB103 - Desenvolvimento de Aplicações para a Web I

PHP – Funções e reutilização de código

Prof. Mário Ezequiel

Reutilização de código

Reutilização de código permite minimizar custo de implementação, aumenta confiabilidade e consistência com o restante do sistema.

PHP fornece duas instruções simples que permitem a reutilização de código:

- require
- include

Instrução **require**

Considerando os dois arquivos abaixo:

reusavel.php:

```
<?php  echo 'Código lido em vários arquivos PHP.<br />';   ?>
```

main.php:

```
<?php   echo 'Este é o arquivo principal.<br />';
        require 'reusavel.php';
        echo 'Finalizando script.<br />';   ?>
```

Quando carregamos main.php, a instrução *require 'reusable. php'* é substituída pelo conteúdo do arquivo, produzindo o seguinte código:

```
<?php
   echo 'Este é o arquivo principal.<br />';
   echo 'Código lido em vários arquivos PHP.<br />';
   echo 'Finalizando script.<br />';          ?>
```

Instrução **include**

A instrução **include** é semelhante à **require**.

A única diferença entre elas é que quando falham, a instrução **require** apresenta um erro fatal, enquanto a instrução **include** somente dá um aviso.

A extensão dos arquivos requeridos ou incluídos pode ser .inc (de include), .php ou qualquer outra.

Uma aplicação muito útil para estas funções é utilizá-las em sites com dezenas de páginas com o mesmo cabeçalho e rodapé.

Implementação de funções

As funções definidas pelo programador seguem a seguinte sintaxe:

```
function  nome_da_função (parametro1, parametro2)
{
  // comandos;
}
```

Para executar uma função, informamos o nome da função com os parâmetros entre parênteses. A função pode retornar ou não um valor. Exemplos:

```
exibir( );
abrir('arquivo.txt');
$soma = somar($a,$b);
```

Implementação de funções

As chamadas para funções não fazem distinção entre letras minúsculas e maiúsculas, assim soma( ), Soma( ) ou SOMA( ) terão o mesmo resultado.

Os nomes de variável distinguem letras maiúsculas e minúsculas, então $Nome e $nome são duas variáveis distintas, mas Nome( ) e nome( ) são a mesma função.

Podemos colocar as funções mais utilizadas num arquivo separado e chamar com **require**.

Implementação de funções

Os parâmetros passados para uma função podem ser opcionais.

No exemplo abaixo, o único parâmetro obrigatório é $data.

```
function create_table( $data, $border=1, $cellspacing =4 ) {
    echo ´<table border=”$border” cellspacing=”$cellspacing”>´;
    echo ´<tr><td>$data[0]</td></tr>\n>´;
    echo ´<tr><td>$data[1]</td></tr>\n>´;
    echo ‘</table>’;
}
```

Podemos chamar a função sem ou com os parâmetros opcionais:

create_table($my_array) ou *create_table($my_array, 3,8)*

Implementação de funções

A quantidade de parâmetros passados para uma função pode variar.

No exemplo abaixo, $x recebe um vetor com os valores passados.

```
function somaValores(...$x) {    # aceita quantidade variada de parâmetros
  $soma = 0;
  $compr = count($x);
  for($i = 0; $i < $compr; $i++) {
    $soma += $x[$i];
  }
  return $soma;
}

echo somaValores(1, 2, 3, 4);              #imprime 10
echo somaValores(1, 2, 3, 4, 5);           #imprime 15
echo somaValores(1, 2, 3, 4, 5, 6);        #imprime 21
```

Escopo de variáveis

PHP possui as seguintes regras de escopo:

- Variáveis declaradas dentro de uma função são chamadas **variáveis locais** e não podem ser acessadas fora dela;
- Variáveis declaradas fora de funções são visíveis em todo o arquivo, mas não dentro de funções. São chamadas **variáveis globais**;
- Utilizar instruções *require* e *include* não afetam o escopo;
- A palavra-chave ***global*** especifica que uma variável definida dentro de uma função terá escopo global;
- As variáveis podem ser manualmente excluídas chamando **unset**($nome_da_variavel).

Escopo de variáveis

Considerando o código abaixo:

```
function fn( ) {
    echo ’dentro da função, var = ’ . $var . ’<br />’;
    $var = ‘conteúdo2’;
    echo ‘dentro da função, var = ‘ . $var . ’<br />’;
    }
$var = ‘conteúdo 1’;
fn ( );
echo ‘fora da função, var =’ . $var . ’<br />’;
```

A saída do código será a seguinte:

dentro da função, var =

dentro da função, var = conteúdo 2

fora da função, var = conteúdo 1

Escopo de variáveis

Considerando o código abaixo:

```
function fn( )   {
    global $var;
    $var = ‘conteudo’;
    echo ‘dentro da função, var =’ . $var . ’<br /> ’;
}
fn( );
echo ‘fora da função, var =  ‘ . $var . ’<br /> ’;
```

A saída desse script será a seguinte:

dentro da função, $var = conteúdo

fora da função, $var = conteúdo

Passagem de parâmetro por referência

Normalmente a passagem é feita por valor, mas podemos passar por referência segundo o exemplo abaixo:

```
function incrementa (&$value, $amount=1)
{
   $value = $value + $amount;
}
$a = 10;
echo $a . ‘<br />’;
incrementa ($a);
echo $a . ‘<br />’;
```

O código acima irá imprimir o valor 10 antes de chamar a função *incrementa*( ) e o valor 11 depois.

Retorno de funções

A instrução *return* finaliza a execução de uma função e pode retornar um valor. Exemplo:

```
function maior ($x, $y)   {
    if (!isset($x) || !isset($y))   {
            echo 'Esta função requer dois números';
            return;
            }
    if ($x>=$y)
            echo $x;
    else
            echo $y;
    }
```

Exercício

Reimplementar o exercício da aula de Processamento de formulários, colocando os códigos de verificação e validação dos campos do formulário em funções

Depois: separar as funções em um arquivo de biblioteca e chamar no PHP usando **require** ou **include**

Referências

- Deitel, cap. 29
- Thomson, cap. 5
- https://www.w3schools.com/php/php_functions.asp
- https://www.w3schools.com/php/php_includes.asp