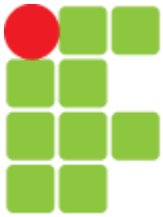

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Programação Web

Aula 07 – Funções e classes

Prof. Pedro Baesse
pedro.baesse@ifrn.edu.br

Roteiro

- Funções
  - Definição
  - Criação
  - Utilização
  - Passagem de parâmetros: valor e refência
  - Recursividade

- Classe
  - Criação
  - Programação Orientada a Objetos em PHP5

Definição de função

- Trechos de código que podem executar qualquer tipo de tarefa
  - Somar dois números, verificar se é um CPF está correto ou se um valor de uma variável é válido, transformar letras maiúsculas em minúsculas...
- Podem ser usadas em qualquer momento da execução
- Deixam o código dos programas mais organizados e modulares
- Evitam a repetição de código quando é preciso usar a mesma tarefa

Função – Sintaxe

Funtion *nome_função* (arg1, arg2 ..., argn)
{
    comandos
    [**return** <expressão>]
}

- *nome_função* deve ser um nome único, um identificador
- Não se pode iniciar o identificador com números e nem usar caracteres especais como ponto, espaço etc. Entretanto o "_" (sublinhado) pode e é muito usado para separar palavras
- Quando chamada uma função pode receber vários valores chamados de argumentos (arg1, arg2 ..., argn). O uso de argumentos é **opcional**, escrevendo *nome_função()*

Definição de função

- Quando o PHP encontra uma chamada de função, o fluxo de execução é direcionado para o início do código da função e depois do seu termino o fluxo retorna para aonde foi parado anteriormente e segue o próximo comando

- Na sintaxe o comando *return* também pode ser usado. Sendo usado, faz com que a função retorne um valor no local em que foi chamada. O *return* pode ser usado para:
  - Atribuir um valor retornado a uma variável
  - Testar (comandos condicionais) o valor retornado de uma função

- Um função também pode não retornar nenhum valor. Como uma função que calcula a multiplicação de dois valores e os imprime na tela, sendo assim não é necessário nenhum retorno

Criação de uma função

- Criando uma função e incluindo uma chamada no programa principal

```
function soma_valores($valor1, $valor2, $valor3)
        {
                $soma = $valor1 + $valor2 + $valor3;

                echo "A soma dos valores $valor1, $valor2 e $valor3 ";
                echo "é $soma";
        }

        $n1 = 10;
        $n2 = 20;
        $n3 = 30;

        soma_valores ($n1,$n2,$n3);
```

Criação de uma função

- As variáveis passadas como parâmetros não precisam ter o mesmo nome dos argumentos definidos na função. Cada um deles assumirá os valores respectivamente dos parâmetros ($valor1 receberá $n1, $valor2 $n2...)

Comando *return* nas funções

- Retorna para ponto de chamada da função o valor que aparece logo depois desse comando. Assim podemos atribuir uma variável (com o operador =) com o valor retornado

- O PHP possui a função *strtoupper* para transformar todas letras de uma string em maiúsculas. Porém, de acordo a versão, esse comando não funciona com acentos. Pode ser criado uma função para resolver isso.

Comando *return* nas funções

```php
function maisculo($string){
        $string = strtoupper ($string);
        $string = str_replace ("á","Á",$string);
        $string = str_replace ("é","É",$string);
        $string = str_replace ("í","Í",$string);
        $string = str_replace ("ó","Ó",$string);
        $string = str_replace ("ú","Ú",$string);
        $string = str_replace ("â","Â",$string);
        $string = str_replace ("ê","Ê",$string);
        $string = str_replace ("ô","Ô",$string);
        $string = str_replace ("î","Î",$string);
        $string = str_replace ("Û","U",$string);
        $string = str_replace ("ã","Ã",$string);
        $string = str_replace ("õ","Õ",$string);
        $string = str_replace ("ç","Ç",$string);
        $string = str_replace ("á","Á",$string);
        $string = str_replace ("à","À",$string);

        return $string;
}

$nome = "José Antônio";
$nome_m = maisculo ($nome);

echo "O nome do rapaz é $nome_m";
```

- Usamos o função *strtoupper* para deixar os caracteres suportados em maiúsculo. Depois o *str_replace*, que substitui todos os caracteres de um tipo de por outro em uma string, para os acentuados
- Se não fosse usado o comando *return* todas as alterações dentro da função seriam perdidas
- Existem diversas funções que trabalham com string, basta consultar o manual do PHP

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Comando *return* nas funções

- O retorno de uma função pode ser impresso diretamente, sem ser preciso armazená-lo em uma variável

```
function triplo($numero){

        $x = $numero * 3;
        return $x;

}

$valor = 5;
echo "O triplo de $valor é " . triplo($valor);
```

Comando *return* nas funções

```
function capitais(){
        $capitais [] = "Natal";
        $capitais [] = "Belo Horizonte";
        $capitais [] = "Rio de Janeiro";
        $capitais [] = "Goiânia";
        $capitais [] = "Brasília";
        return $capitais;
}
//início do programa principal
$nomes = capitais();
for($i=0; $i<sizeof($nomes); $i++){
        echo "\$nome[$i] vale $nome[$i] <br>";
}
```

- A função capitais não recebe nenhum argumento. Apenas retorna um array aonde foi chamada

Comando *return* nas funções

- O próximo exemplo usará as funções *time* e *getdate* do PHP. *Time* retorna o tempo passado desde 1° de janeiro de 1970 (padrão unix) em segundos e *getdate* transforma esse valor em uma data

Comando *return* nas funções

```php
function retorna_data(){
                $agora = time();
                $data = getdate($agora);
                if($data["wday"]==0){echo "Domingo, "; }
                elseif($data["wday"]==1){echo "Segunda-feira, "; }
                elseif($data["wday"]==2){echo "Terça-feira, "; }
                elseif($data["wday"]==3){echo "Quarta-feira, "; }
                elseif($data["wday"]==4){echo "Quinta-feira, "; }
                elseif($data["wday"]==5){echo "Sexta-feira, "; }
                elseif($data["wday"]==5){echo "Sábado, "; }
                if($data["mon"]==1){$mes = "Janeiro"; }
                elseif($data["mon"]==2){$mes = "Fevereiro"; }
                elseif($data["mon"]==3){$mes = "Março"; }
                elseif($data["mon"]==4){$mes = "Abril"; }
                elseif($data["mon"]==5){$mes = "Maio"; }
                elseif($data["mon"]==6){$mes = "Junho"; }
                elseif($data["mon"]==7){$mes = "Julho"; }
                elseif($data["mon"]==8){$mes = "Agosto"; }
                elseif($data["mon"]==9){$mes = "Setembro"; }
                elseif($data["mon"]==10){$mes = "Outubro"; }
                elseif($data["mon"]==11){$mes = "Novembro"; }
                elseif($data["mon"]==12){$mes = "Dezembro"; }
                $data_atual = $data["mday"] . " de " . $mes . " de " .$data["year"];
                return $data_atual;
}
$hoje = retorna_data();
echo $hoje;
```

- Esse exemplo usa as funções *time* e *getdate* do PHP. *Time* retorna o tempo passado desde 1° de janeiro de 1970 (padrão unix) em segundos e *getdate* transforma esse valor em uma data com diversas informações

Passagem de parâmetros: valor e referência

- Quando passamos uma variável como parâmetro em uma função, todas as alterações nelas que ocorrerem dentro da função não serão refletidas fora da função. Isso é chamado de processo de passagem de parâmetros por valor, já que apenas o valor é passado para uso na função

- Quando queremos que variável usada no argumento sofra as alterações internas da função usamos a passagem de parâmetros por referência

Passagem de parâmetros: valor e referência

```php
function dobro($valor)
{
        $valor = 2 * $valor;
}
function duplica(&$valor)
{
        $valor = 2 * $valor;
}

$valor = 5;
dobro($valor);
echo $valor . "<br>";
duplica ($valor);
echo $valor;
```

- Para usar o parâmetro por referência, bastar usar o símbolo & antes da variável
- Enquanto em *dobro*, foi passado somente o valor da variável, em *duplica* o PHP simplesmente criou uma referência para a variável

Passagem de parâmetros: valor e referência

```
function capital($capital, $estado = " Um estado brasileiro")
{
        echo "$estado, capital:  $capital<br>";
}
capital("Natal", "RN");
capital("Belo Horizonte", "MG");
capital("Goiânia");
```

- Também é possível usar valores-padrão nos parâmetros. Caso não seja colocado nenhum argumento, o valor-padrão será assumido, como na última linha do exemplo

- Os valores-padrão devem ser **sempre** os últimos para que o PHP interprete primeiro os parâmetros sem valor-padrão e assim evitar erros

Funções Recursivas

- São funções que fazem chamadas a elas mesmas

```
function recursiva($valor)
{
        if($valor!=0)
        {
                echo "Foi chamada a função recursiva passando o valor $valor <br>";
                recursiva($valor-1);
        }
}
recursiva(8);
```

Funções Recursivas

- Outro exemplo clássico de função recursiva é o cálculo do fatorial. Fatorial de um número n é multiplicação dele mesmo por todos os seus antecessores. Por convenção, o fatorial de 0 é 1

```
function fatorial($numero)
{
        if($numero<0)
                { return -1;}
        if($numero<=1)
                { return 1;}
        return $numero*fatorial($numero-1);
}

echo "O fatorial de 3 é " . fatorial(3);
echo "<br>O fatorial de 4 é " . fatorial(4);
echo "<br>O fatorial de 5 é " . fatorial(5);
```

Reutilização de Funções

- Imagine que um site possua 50 páginas com a mesma função em cada uma delas. Não é necessário repetir a função em cada uma delas. Caso isso ocorresse, seriam necessárias 50 funções diferentes e uma alteração teria que ser feita em cada uma delas

- Um exemplo simples é o formato de exibição de uma data. O ideal é mudar somente um arquivo e refletir em todas locais que se usa a função. Isso é possível com o uso de *includes*, que será visto mais a frente

Definição de classe

- Classes são conjuntos de variáveis e funções que descrevem um objeto com suas características e ações. Um objeto pode usar todas variáveis e funções existentes em sua classe

- **Classe apartamento (propriedades)**
  - numero_quartos
  - numero_banheiros
  - preco_aluguel
  - preco_venda
  - tamanho
  - localizacao

Como criar uma classe

```
class nome_classe
{
        <variáveis>
        <procedimentos>
}
```

- *var* é usado para criar variáveis dentro de uma classe. Também pode ser usado a variável interna do PHP, *$this* que serve para referenciar o próprio objeto
- Para criar um novo objeto, deve ser atribuído a uma variável que representará o objeto a instrução *new* seguida do nome da classe
- Já para acessar valores de uma array dentro de uma classe, devemos usar -> seguido do nome do array e chave associada

Como criar um classe

```
class Loja
{
        var $itens;
        function adiciona ($codigo,$quantidade){
                if(isset($this->itens[$codigo]))
                        $this->itens[$codigo] += $quantidade;
                else
                        $this->itens[$codigo] = $quantidade;
        }
        function remove($codigo,$quantidade){
                if(itens[$codigo] > $quantidade;){
                        itens[$codigo] -= $quantidade;
                }
                else{
                        return false;
                }
        }
}

$estoque = new Loja;
```

Como criar um classe

- Classe Loja
  - Possui como variável o array *itens* que foi declarada usando a instrução var
  - Também possui duas funções para acrescentar ou remover itens do estoque
  - O objeto é criado quando fazemos a atribuição de *new* Loja à $*estoque*
  - Depois de criado o objeto, $*estoque* já acessar as variáveis e funções disponíveis na classe

Como criar um classe

- Note que o uso do símbolo -> para acessar funções e variáveis. Sempre deve ser usado quando trabalhando com objetos

```
...

$estoque->adiciona("bermuda",2);
$estoque->adiciona("camiseta",3);

echo "A loja já possui " . $estoque->itens["bermuda"] . "bermudas. <br>";
echo "A loja já possui " . $estoque->itens["camisetas"] . "camisetas. <br>";

...
```

Palavra-chave *private* e *protected*

- Utilizadas para criar métodos e variáveis privadas e protegidas dentro de uma classe (PHP 5)
- Uma variável ou método privados só podem ser acessados de dentro da classe que foram declarados
- Uma variável ou método protegidos poderão ser acessados também pelas subclasses da classe onde foram declarados

Palavra-chave *private* e *protected*

```php
class Classe1{
        private $var1 = "Olá, var1! \n";
        protected $var2 = "Olá, var2! \n";
        protected $var3 = "Olá, var3! \n";

        function bomDia(){
                print "Classe1 ". $this->var1. "<br>";
                print "Classe1 ". $this->var2. "<br>";
                print "Classe1 ". $this->var3. "<br><br>";
        }
}

class Classe2 extends Classe1{
        function bomDia(){
                Classe1::bomDia(); //Exibe
                print "Classe2 ". $this->var1. "<br>"; //Não exibe nada (erro)
                print "Classe2 ". $this->var2. "<br>"; //Exibe
                print "Classe2 ". $this->var3. "<br><br>"; //Exibe
        }
}
$obj = new Classe1();
$obj->bomdia();
$obj= new Classe2();
$obj->bomDia();
```

Palavra-chave *private* e *protected*

```
class Classe1{
        private function MetodoPrivado(){
                echo "Classe1::MetodoPrivado() chamado. <br>";
        }
        protected function MetodoProtegido(){
                echo "Classe1::MetodoProtegido() chamado. <br>";
                $this->MetodoPrivado();
        }
}

class Classe2 extends Classe1{
        public function MetodoPublico(){
                echo "Classe2::MetodoPublico() chamado.<br>";
                $this->MetodoProtegido();
        }
}
$obj = new Classe2;
$obj->MetodoPublico();
```

Métodos abstratos e interface

- O PHP 5 permite a criação de métodos abstratos, ou seja, o método é apenas declarado, sem uma implementação. Isso é feito por alguma classe que a herda

- A classe com o método abstrato também deve ser declarada abstrata e não pode criar objetos diretamente

Métodos abstratos e interface

```
abstract class ClasseAbstrata{
        abstract public function teste();
}

class ClasseImplementacao extends ClasseAbstrata{
    public function teste(){
        echo "Método teste() chamado!<br>";
    }
}

//Erro $obi = new ClasseAbstrata;
$obi = new ClasseImplementacao;
$obj->teste();
```

Métodos abstratos e interface

- O PHP 5 também possibilita a criação de interfaces. Recurso parecido com classes abstratas porém uma classe pode implementar diversas interfaces
- São adicionadas apenas as declarações dos métodos que irão fazer parte da classe que irá implementá-la
- A palavra reservada *implements* indica uma classe implementa uma determinada interface

Métodos abstratos e interface

```
interface MinhaInterface{
        public function Teste();
}

Class MinhaClasse implements MinhaInterface{
        public function Teste(){
                echo "Minha implementação de uma interface funciona!";
        }
}

$obi = new MinhaClasse;
$obj->Teste();
```

Palavra-chave *final*

- Os métodos declarados com a palavra *final* não poderão ser sobrescritos pelas subclasses, causando erro causo ocorra

```
Class MinhaClasse implements MinhaInterface{
        final function Teste(){
                        //..
        }
}
```

- Uma classe também pode ser *final*, não permitindo subclasse, causando erro na tentativa

```
final class MinhaClasse {
        //..
}
```

Construtores e destrutores

- O método *__construct* é executado toda a vez que for criado um novo objeto da classe aonde foi declarado. Isso permite que objetos internos sejam iniciados antes do seu uso

- Por questões de compatibilidade, se o *__construct* não for achado, o PHP irá procurar por um método que possua o mesmo nome da classe

Construtores e destrutores

```
class Classe{
        function __construct(){
                print "Este é construtor da Classe<br>";
        }
}

class SubClasse extends Classe{
        function __construct(){
                parent::__construct();
                print "Este é construtor da SubClasse<br>";
        }
}

$obj = new Classe;
$obj = new SubClasse;
```

Construtores e destrutores

- Uma classe também pode ter um destrutor, usando __destruct(). Será chamado toda a vez depois da última referência a objeto, antes da liberação de sua memória

- Útil para fins de depuração, fechamento de conexão com banco de dado entre outras tarefas

Construtores e destrutores

```
class Classe{
        function __construct(){
                $this->nome= "Classe com construtor e destrutor";
                print "Este é construtor da ". $this->nome ."<br>";
        }
        function __destruct(){
                print "Este é destrutor da ". $this->nome ."<br>";
        }
}

$obj = new Classe;
```

Variáveis e métodos estáticos

- Variáveis e métodos estáticos (antecedidos por *static*) podem ser usados mesmo sem que a classe seja iniciada

```
class Classe{
        static $variavel_estatica = 10;

        public static function Estatico(){
                echo "<br>Isso é um método estático!";
        }
}
print Classe::$variavel_estatica;
Classe::Estatico();
```

Dúvidas

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Referências

- PHP Manual:
  - http://www.php.net/manual/pt_BR/index.php
- Desenvolvendo Websites com PHP
  - De Juliano Niederauer