8.0 Manipulação de funções

Objetivos

Apresentar as estrutura de funções em PHP; Mostrar qual a definição entre variável global e estática e sua relação com o uso de funções; Mostrar também o uso de passagem de parâmetros, recursão e qual a sua principal finalidade.

Quando queremos um código funcional para determinado fim, com por exemplo fazer um cálculo ou alguma interação dentro do PHP, usamos o que chamamos de função. As funções são um pedaço de código com o objetivo específico, encapsulado sob um estrutura única que recebe um conjunto de parâmetros e retorna ou não um determinado dado. Uma função é declarada uma única vez, mas pode ser utilizada diversas vezes. É uma das estruturas mais básicas para prover reusabilidade ou reaproveitamento de código, deixando as funcionalidades mais legíveis.

8.1 Declarando uma Função.

Declaramos uma função, com o uso do operador function seguido do nome que devemos obrigatoriamente atribuir, sem espaços em branco e iniciando sempre com uma letra. Temos na mesma linha de código a declaração ou não dos argumentos pelo par de parênteses "()". Caso exista mais de um parâmetro, usamos virgula(,) para fazer as separações. Logo após encapsulamos o código pertencente a função por meio das chaves ({}). No final, temos o retorno com o uso da cláusula **return** para retornar o resultado da função que pode ser um tipo inteiro, array, string, ponto flutuante etc. A declaração de um retorno não é obrigatório. Observe a sintaxe:

```
function nome_da_função( $argumento_1, $argumento_2, $argumento_n )
{
comandos; return $valor;
}
```

Observe um exemplo onde criamos uma função para calcular o índice de massa corporal de uma pessoa (IMC), onde recebe como parâmetro dois argumentos. Um é a altura representada pela variável $altura e o outro é o peso representada pela variável $peso. Passamos como parâmetros para essa função o peso = 62 e a altura = 1.75. Observe:

Exemplo:

```php
<?php
// declaração da função:
function calculo_IMC($peso, $altura){

    return $peso/($altura*$altura);

}
// fazendo a chamada da função:
echo calculo_IMC( 62 , 1.75 );

?>
```


Resultado: 20.244897959184

Nesse exemplo temos a declaração e logo após a chamada da função, onde é nesse momento que passamos os dois parâmetros na ordem que foi declarada na função.

Lembrando que essa ordem é obrigatória.

Observe mais um exemplo, onde a função declarada porém não possui a cláusula **return.**

```php
<?php
// declaração da função:
function imprime($texto){

    echo "<h1>$texto</h1>";

}
// fazendo a chamada da função:
imprime("Testando a função!!");

?>
```

Resultado:

Testando a função

8.2 Escopo de Variáveis em Funções

Um conceito importante em programação são os tipos de declarações de variáveis, onde sua visibilidade vai depender de onde ela é declarada. O acesso a essas variáveis podem ser definidas da seguinte forma:

Variáveis locais < São aquelas declaradas dentro de uma função e não tem visibilidade

fora dela. Veja um exemplo:

O valor da variável $a não é impresso na tela, pois ela só existe dentro da função, qualquer outra variável declarada com o mesmo nome fora da função é uma nova variável.

```php
<?php
function Teste()  {
    $a = 25; // variável local
}
Teste();     /* ativa a função */
echo $a;     /* não imprime */
?>
```

Variáveis Globais < São variáveis declaradas fora do escopo de uma função, porém tem visibilidade (pode ser acessada) ao contexto de uma função sem passá-la como parâmetro. Para isso declaramos a variável e fazemos a sua chamada logo após com o uso do termo **global**.

Exemplo:

```php
<?php
$a = 0;          /* escopo global */
function Teste()  {
    global $a;  /* variável global */
    $a = 25;      }
Teste();         /* ativa a função */
echo $a;       /*imprime 25 na tela*/
?>
```

Resultado: 25

Variáveis estáticas < Podemos armazenar variáveis de forma estática dentro de uma função. Significa que ao fazermos isso, temos o valor preservado independente da ultima execução. Usamos o operador static para declaramos a variável.

Exemplo:

```php
<?php
function Teste()  {
    static $a; // variável statica
    $a += 10;  // atribuição +10
    echo $a.",";   // imprime na tela
}
Teste();    // 1° chamada da função
Teste();    // 2° chamada da função
Teste();    // 3° chamada da função
?>
```

Resultado: 10,20,30,

Observe que o valor é mantido e a

cada chamada é acrescentado +10, caso não exista o static o resultado seria: 10,10,10, .

8.3 Passagem de Parâmetro.

Como vimos anteriormente, podemos passar ou não parâmetros em uma função, porém existem dois tipos de passagem de parâmetros: Por valor (by value) e por referência (by reference).

Por Valor < Normalmente, a passagem de parâmetros em PHP é feita por valor, ou seja, se o conteúdo da variável for alterado, essa alteração não afeta a variável original. Exemplo:

```php
<?php
function valores($variavel , $valor)  {
    $variavel += $valor;
}
$a = 23;
valores($a,26);
print $a;  #Resultado: 23
?>
```


O exemplo acima mostra que passamos um valor de $a para a função, porém o valor temos a garantia que o valor continua integro, ou seja, não foi modificado ao longo do código.

Por Parâmetro < Para passarmos um valor por parâmetro, simplesmente colocamos o operador "&" na frente do parâmetro que queremos que o valor seja alterado, observe o exemplo abaixo:

```php
<?php
function valores( &$variavel , $valor)  {
    $variavel += $valor;
}
$a = 23;
valores($a,26);
print $a;  #Resultado: 49
?>
```

Observe agora nesse último exemplo que apenas acrescentamos o operador "&" no parâmetro que queríamos que alterasse a variável passada como parâmetro, fazendo com que o resultado fosse a soma de 23 + 26 = 49.

Por argumentos variáveis < O PHP permite outras formas avançadas de passagem de parâmetros, onde o valor e a quantidade são definidas de forma automáticas por meio das funções **func_get_args()** e **func_num_args().**

func_get_args() < diz os valores(argumentos) passado para a função.

func_num_args() < diz a quantidade de valores passados para a função.

Observe um exemplo mais complexo abaixo:

```php
<?php
function Nomes()  {
  $argumentos = func_get_args();
  $quantidade = func_num_args();

    for($i=0;$i<$quantidade;$i++){
        echo "nome = ".$argumentos[$i]." ,";
    }
}

Nomes("Alex","Sara","Maria","Bruna");
?>
```

Resultado: nome = Alex ,nome = Sara ,nome = Maria ,nome = Bruna ,

8.4 Valor de Retorno

Toda função pode opcionalmente retornar um valor, ou simplesmente executar os comandos e não retornar valor algum. Não é possível que uma função retorne mais de um valor, mas é permitido fazer com que uma função retorne um valor composto, como listas ou array's. As operações aritméticas podem ser feita de forma direta no retorno. Observe um exemplo onde

temos uma operação direta:

```php
<?php

function Soma($n1, $n2)  {
  return $n1+$n2;
}
function Texto(){
  return "O resultado é: ";
}

echo Texto().Soma(115,24.5);

?>
```

Resultado:

O resultado é: 139.5

Também podemos determinar mais de um retorno desde que eles não sejam acessado ao mesmo tempo, observe o exemplo abaixo:

```php
<?php
function Teste($caso)  {
  switch($caso){
      case 1:
          return "opção 1"; break;
      case 2:
          return "opção 2"; break;
      case 3:
          return "opção 3"; break;
      default:
          return null;
  }
}
echo Teste(2);
?>
```

Esse código mostra de forma clara que não existe a possibilidade de retornarmos mais de um **return**, caso isso ocorresse, teríamos um erro, ou não funcionamento da função.

8.5 Recursão.

Função recursiva é uma definição usada tanto na programação quanto na matemática, onde, significa que uma função "faz a chamada" de si mesma na sua execução. Um exemplo é o calculo do fatorial de um número. Observe:

Fatorial de 5: *5! = 5*4!, 4! = 4*3!, 3! = 3*2!, 2! = 2*1! ou 5*4*3*2*1 = 120.*

Exemplo:

```php
<?php
function Fatorial($numero)  {
if($numero == 1) // Fatorial encerra quando o numero é igual a 1.
      return $numero;
else
// Nesse retorno fazemos a chamda da função, passando o numero - 1.
      return $numero* Fatorial($numero-1);
}
echo Fatorial(5);
?>
```

Resultado: 120

EXERCÍCIO RESOLVIDO COM FUNÇÕES.

Realize o passo a passo de todas as operações matemáticas realizadas nas chamadas da função "calculo" junto com o resultado final.

```php
<?php

function calculo($n1 , $n2 , $n3){
    return $n1 + $n2 *$n3;
}

echo calculo(5, calculo(1, 2, 3), 3);
```

Entendendo o algorítmo:

- Uma função chamada "calculo" foi definida com 3 parâmetros, "$n1", "$n2", "$n3"
- A função "calculo" é responsável por retornar a multiplicação de "$n2" e "$n3" e com o resultado, realizar a soma com "$n1" de acordo com a precedência matemática.
- Na sua chamada, temos 2 funções, a linguagem PHP começa realizando os cálculos das funções mais internas, sendo assim a primeira função que iremos analisar.
- A função calculo interna recebe 3 parâmetros.

1. Numeral 1
2. Numeral 2
3. Numeral 3

- baseado na operação matemática temos : 1 + 2 * 3 = 7.
- Agora podemos resolver a função calculo mais externa, podemos encontrar mais 3 parâmetros

1. Numeral 5
2. Função que retorna um Numeral 7
3. Numeral 3.

Baseado nos cálculos temos: 5 + 7 *3 = 26.

O resultado que será impresso é : **26** baseado na lógica construída.

EXERCÍCIOS PROPOSTOS

Diga com suas palavras uma definição para função, e como podemos declará-la em PHP.

Qual a diferença de variáveis globais para variáveis locais e como podemos defini-las em PHP?

O que é um parâmetro, e quais os tipos de parâmetros em PHP?

Quais as funções que podemos usar para criarmos uma função onde seus parâmetros são passados pro argumentos variáveis?

O que é um valor de retorno e qual o comando usado quando queremos retornar algo dentro de uma função?

O que é recursão?

7°) Crie uma função que determine se um numero é par ou ímpar. E faça uma chamada dessa função imprimindo o resultado.

8°) Crie uma função que calcule a fatorial de um número.

9°) Crie uma função para determina se um numero é primo ou não. Numero primo é aquele que possui dois divisores, 1 e ele mesmo. Criem um laço de repetição e use estrutura de controle.