Introdução à Funções

É um bloco de código que faz alguma coisa, x, y, e internamente se cria um processo e retorna um resultado. Toda função tem entrada de dados, que processa esses dados e retorna um resultado como saída de dados.

Exemplo função que vai somar...

- numero 1

- numero 2

Internamente:

- pegar o numero 1 e adicionar o número 2

Resultado:

- a soma do número 1 e número 2

somar 1 3 = 4

somar 10 15 = 25

No PHP temos vários tipos de funções. A função print_r() é um exemplo de **função nativa**, por que já vem junta ao PHP. Quem criou o PHP já criou junto essa função, já vem internamente, pronta para uso. Mas podemos criar funções.

Definindo Função

Como criar uma função:

sintaxe:

```
function nomeFunção (parâmetros) { códigos }
```

Para exemplificar uma função, vamos criar uma que mostre um número x e a seguir mostre os 9 números seguintes a esse número.

Arquivo index.php

```
<?php
```

```
	//Criei aqui a função subsequente
	function subsequente(){
		for($i=0;$i<10;$i++){
			echo $i."</br>";
		}
		echo "</hr>";
	}
	//Na linha a seguir, executamos a função subsequente.
	subsequente();
?>
```

Outro Exemplo:

Arquivo programa1.php

```
<?php
	function stop(){
		echo "Stop COVID_19";
	}
	stop();
?>
```

Por exemplo, queremos que execute 20 vezes essa função stop(), para isso podemos usar o comando FOR.

```
<?php
	function stop(){
		echo "Stop COVID_19 </br>";
	}
	//stop();
	for($i=0;$i<20;$i++){
```

```
        stop();
    }
?>
```

Parâmetros: Definição normal

Arquivo programa2.php

```
<?php
    //func. somar com parâmetros, as duas variáveis $n1 e $n2
    function somar($n1, $n2){
        $total = $n1 + $n2;
        return $total; //Saída dessa função
    }
    //O valor que retornar da função somar fica armazenado em $soma.
    $soma = somar(10, 5); //Aqui executamos a função somar e mandando
    //os valores 10 e 5 respectivamente para $n1 e $n2
    echo "Total: ".$soma;
?>
```

Tudo que é executado dentro da função fica dentro da função, inclusive definição de variáveis. Por exemplo, se mudarmos o código conforme exemplo anterior para:

```
function somar($n1, $n2){
    $total = $n1 + $n2;
    }
    $soma = somar(10, 5);
    echo "Total: ".$total;
```

Observem no exemplo anterior, que foi tirado o return da função somar. E fora da função fiz a referência a variável $total. O que faz ocorrer um erro no código especificando como variável não definida. Isso mostra que tudo que está dentro da função permanece dentro da função.

Pra torná-la útil, usamos o return:

```
<?php
function somar($n1, $n2){
   $total = $n1 + $n2;
   return $total;
}
   $soma = somar(10, 5);
   echo "Total: ".$soma;
   //echo somar(10, 5);
?>
```

Uma função pode ser rodada diversas vezes, observem o código do programa3.php

Arquivo programa3.php

```
<?php
function somar($n1, $n2){
   $total = $n1 + $n2;
   return $total;
}
//a seguir, exemplo de execução de varias vezes da função somar().
$x = somar(10, 20);
$y = somar(3, 6);
$z = somar($x, $y);
echo $x."</br>";
echo $y."</br>";
echo $z;
?>
```

Parâmetros: Type e Valor padrão

Vamos ver a seguir, como trabalhar com parâmetros de funções, como podemos criar um parâmetro opcional.

Por exemplo:

```
<?php

//Adicionei mais um parâmetro na função somar() a variável $n3

function somar($n1, $n2, $n3){

   $total = $n1 + $n2 + $n3;

   return $total;

}

//Observem que na execução só  mandamos dois valores (10, 20)

$x = somar(10, 20);

$y = somar(3, 6);

$z = somar($x, $y);

echo $z;

?>
```

Se executarmos essecó digo, vamos ver que vai dar um erro fatal, dizendo que muitos poucos argumentos foram mandados para a função somar. Era esperado três parâmentros na função em vez de dois.

Agora vamos tornar esse terceiro parâmetro " $n3 " opcional. Como fizemos para torná-lo opcional ? Fizemos indiretamente, através de um valor padrão, pra qualquer um dos três parâmetros da função ($n1, $n2, $n3). Quando definimos um valor padrão para um parâmetro, automaticamente na execução dessa função caso o parâmetro seja omitido, ele não seja enviado por exemplo, automaticamente aquela variável vai assumir o valor padrão. Vamos definir para $n3 o valor padrão 0 (zero). Mas isso não significa que $n3 sempre vai ser 0 (zero). Se mandar um terceiro parâmetro para função ele vai ser subtituido, caso contrário fica com valor padrão igual a 0 (zero).

Exemplo:

```
<?php

function somar($n1, $n2, $n3 = 0){        //Aqui $n3 = 0 virou opcional

   $total = $n1 + $n2 + $n3;

   return $total;
```

```
}

$x = somar(10, 20);

$y = somar(3, 6);

$z = somar($x, $y);

echo $z;

?>
```

Se executarmos o código vamos ver que não dá mais o erro. Vejam que não foi mandado um terceiro parâmetro (**somar(10, 20); somar(3, 6);**) em nenhuma dessas opções, para a função somar($n1, $n2, $n3 = 0), mas não deu erro, por que definimos um valor padrão para o terceiro parâmetro nela que nesse caso assume o valor zero (0).

Isso significa se mandarmos um terceiro parâmetro para a função somar(), esse valor padrão opcional vai ser subtituido. Veja exemplo a seguir:

```
<?php

function somar($n1, $n2, $n3 = 0){        //Aqui $n3 = 0 virou opcional

   $total = $n1 + $n2 + $n3;

   return $total;

}

$x = somar(10, 20);

$y = somar(3, 6, 2); //Mandamos valor 2 como terceiro parâmetro para função.

$z = somar($x, $y);

echo $z;

?>
```

Executamos o código e o resultado vai ser o valor 41 porque mandou o valor 2 como terceiro parâmetro e subtituiu o valor padrão de $n3 = 0, pelo valor 2.

Veja exemplo abaixo:

```
function somar($n1, $n2 = 0, $n3 = 0){   //$n2 = 0,  $n3 = 0 viraram opcionais

   $total = $n1 + $n2 + $n3;
```

```
    return $total;
}
```

O que significa function **somar($n1, $n2 = 0, $n3 = 0)** ?

Significa que a função somar() necessita receber um único parâmetro.

```
$x = somar(10); //Aqui exemplo de envio de um único valor para função.
$y = somar(3, 6, 2);
$z = somar($x, $y);
echo $z;
```

Isso significa que ao executar vai funcionar sem dar nenhum erro.

Mas por exemplo, se mandarmos sem nenhum parâmetro:

```
$x = somar(); //Aqui exemplo sem nenhum valor, parâmetro.
$y = somar(3, 6, 2);
$z = somar($x, $y);
echo $z;
```

Nesse caso dá erro, apontando que esperava pelo menos um parâmetro.

Aqui vocês aprenderam como tornar um parâmetro opcional e como definir um valor padrão para um parâmetro, que pode ser um número, uma string.

Tem um outro detalhe muito importante que é o tipo de parâmetro que a função vai receber. Esse recurso de definir o tipo específico de dados para as variáveis é novo no PHP.

Na função somar() quero definir para receber apenas números, ou seja, na função somar($n1, $n2 = 0, $n3 = 0), vai receber um tipo de dado específico.

Vamos definir que as variáveis na função somar, receba somente tipo numerico inteiro. Para isso antes da variável colocamos o termo int. Observem a aplicação no exemplo a seguir:

```
<?php
```

```
function somar(int $n1, int $n2 = 0, int $n3 = 0){
   $total = $n1 + $n2 + $n3;
   return $total;
}
$x = somar(10, 4);
echo $x;
?>
```

Parâmetros: Referência ou Valor

Vamos ver como passar parâmetros como referência e passar parâmetros como valor.

```
<?php
	function somar($n1, $n2){
		$total = $n1 + $n2;
		return $total;
	}
	//Aqui passa variáveis como parâmetros:
	$x = 10;
	$y = 20;
	//Aqui passamos variáveis como parâmetros.
	$soma = somar($x, $y);
	echo "Total: ".$soma;
?>
```

Significa que não passamos as variáveis $x e $y para a função somar, mas sim os valores contidos nelas. Então foi passado um parâmetro por valor, ou seja, 10 e 20 respectivamente. Essa é a forma que vamos utilizar em 90% dos casos.

Mas existe um processo de passar variáveis por referência. Que significa que não passamos o valor mas sim a própria variável. Então se alteramos essa variável dentro da função, como foi passada a própria variável, ela vai ser alterada também fora da função. Um exemplo prático:

```php
<?php
	function somar($n1, $n2, &$total){ //O & representa var. por referência
		$total = $n1 + $n2;
		return $total;
	}

	$x = 2;
	$y = 4;
	$soma = 0;
	somar($x, $y, $soma); //foi passada a variável $soma e não seu valor
	echo "Total: ".$soma;
?>
```

Observação: Para função receber a **variável por referência** temos que colocar o **&** (e comercial) antes da variável. Então quando definimos uma variável por referência obrigatóriamente temos que passar uma variável para ela, não pode ser um número. Conforme exemplo anterior esse terceiro parâmetro da função somar, tem que ser uma variável passada como referência.

Foi mandada a $soma com parâmetro, a variável por referência faz tipo um espelhamento, ela altera internamente e manda para a variável fora da função, ou seja, altera quem mandou, no caso a variável $soma.

O próprio php, tem funções nativas que funcionam por referência, seja exemplo:

```php
<?php
	$lista = [5, 9, 3];
	//print_r($lista);
	sort($lista);
	print_r($lista);
?>
```

A função sort(), ela ordena os valores do menor para o maior de um array que usar como referência. Ela suga o array, ordenandos seus valores em ordem crescente. É um exemplo de função nativa do php que recebe o parâmetro por referência.

Funções Anônimas

É uma funcionalidade nova no php.

```
<php

        function somar(){

        }

?>
```

Quando criamos uma função conforme exemplo anterior, ele tem um nome, que é somar().

Em uma função anônima ela não tem um nome. Ela é igual ao exemplo a seguir.

```
<php

        function(){

        }

?>
```

Lógico que não vamos criar uma função e não ligar a nada. Como usamos uma função anônima? Duas formas principais:

- Primeiro armazenando ela dentro de uma variável
- Segundo passando como parâmetro para uma outra função que criamos ou função nativa do php.

Exemplo de função anônima armazenada numa variável:

```
<?php

        // Função calcula 10% do valor que mandar como parâmetro

        $dizimo = function($valor){ //armazenei a fun. anônima em $dizimo

                return $valor * 0.1;

         }; // finalizar com ponto e vírgula

        //Aqui estou usando ela

        echo $dizimo(90);

?>
```

Além disso podemos passar ela para uma outra variável, por exemplo:

```
$funcao = $dizimo;
echo $funcao(82);
```

Ficando assim o exemplo anterior com as alteração:

```
<?php
	// Função calcula 10% do valor que mandar como parâmetro
	$dizimo = function($valor){
		return $valor * 0.1;
   };
	$funcao = $dizimo;
	echo $funcao(82);
?>
```

Agora passar a função como parâmetro para qualquer outra função:

```
<?php
	algumaFuncao(10, function(){
		criaAquiAFunção
	});
?>
```

Ainda pode ser feito armazenando em uma variável e passar ela assim:

```
<?php
	$funcao = function(){
		. . .
```

```
    }
    algumaFuncao(10, $funcao);
?>
```

Funções Flecha (Arrow) (7.4)

Mais conhecida como Arrow function. Esse tipo funciona só a partir da versão php 7.4.

Em exercício anterior tem um exemplo feito assim:

```
<?php
    // Função calcula 10% do valor que mandar como parâmetro
    $dizimo = function($valor){
       return $valor * 0.1;
  };
    echo $dizimo(82);
?>
```

Podemos diminuir essa função acima, em uma única linha com a função Arrow function, por exemplo:

```
<?php
    $dizimo = fn($valor) => $valor * 0.1;
    echo $dizimo(82);
?>
```

Observação: Colocá-se fn() e dentro os parâmetros e logo depois a => (sinal de igual e maior), que significa o arrow, que é o retorno.

Mas atenção, a Arrow function permite fazer só uma expressão dentro dela, um só comando para executar. O exemplo a seguir, já não poderia ser passada para arrow function, pois tem dois comando dentro dela.

```
$dizimo = function($valor){
        $res = $valor * 0.1;
        return $res;
  };
```

Funções Recursivas

Se usa muito pouco na prática essas funções recursivas em sistemas. Mas é bom ter conhecimento a respeito. Função recursiva é uma função que executa ela mesma internamente. Como exemplo prático para entender, uma função que vai receber um número e dividir esse número por 2, várias vezes ela mesma, até que o número chegue a 0 (zero).

```
<?php

	function dividir($numero){ //recebe um número

		$metade = $numero / 2;

		echo $metade."</br>";

		//Depois de dividir, verifico se ainda está maior que zero

		if(round($metade) > 0){ //Aqui usei a função round() para
			//arredondar o valor até chegar 0.0...

				dividir($metade);

		}

	}

	dividir(100);

?>
```

Um exemplo de uso na prática, quando fizer um sistema que vai navegando em pasta. Entra na pasta vê tudo que tem dentro, arquivos, pastas e então ele vai usar a mesma função que escaneou a pasta original, a primeira pasta e vai usar essa mesma função para escanear as pastas internas. E se tiver outras dentro delas faz o mesmo procedimento, assim sucessivamente até não achar mais pastas dentro de outras. Nesses casos específicos vai ser usado funções recursivas.

Funções Nativas de Matemática

São funções matemáticas que já estão dentro do php. São funções uteis que usaremos eventualmente no dia a dia criando sistemas. Vamos ver as principais:

Função **abs(),** pega o valor absoluto de um número.

```
<?php

	$numero = -8.4;

	echo abs($numero);
```

```
?>
```

Função **pi()** mostra o valor do pi.

```
<?php
        echo pi();
?>
```

Função **floor()** arredonda para menos.

```
<?php
        $numero = 2.7;
        echo floor($numero); //mostra o valor 2 nesse exemplo.
?>
```

Função **ceil()** arredonda para mais.

```
<?php
        $numero = 2.7;
        echo ceil($numero); //mostra o valor 3 nesse exemplo
        $numero = 3.1;
        echo "</br>".ceil($numero); //mostra 4
?>
```

Função **round()** dependendo do número vai arredondar para menos ou para mais.

```
<?php
        $numero = 3.1;
        echo round($numero); //mostra o valor 3 nesse exemplo
        $numero = 3.5;
        echo "</br>".round($numero); //mostra 4
```

```
	$numero = 3.4;
	echo "</br>".round($numero); // vira 3
	$numero = 14.56289;
	echo "</br>".round($numero); //15
	$numero = 12.4281;
	echo "</br>".round($numero); //vira 12
	//Se quiser manter alguma casa decimal fizemos o seguinte:
	echo "</br>".round($numero, 2);
?>
```

Função **rand()** gera um número aleatório. Nela passamos como parâmetro dois valores que significa o valor mínimo e o valor máximo respectivamente.

```
<?php
	$aleatorio = rand(1, 10);
	echo $aleatorio;
?>
```

Executamos o código e sempre que atualizar a página gera um número aleatório de 1 a 10. Como exemplo de aplicação, uso em sistemas de sorteios.

Função **max()** retorna o número maior de uma lista.

```
<?php
	$lista = [8, 23, 4, 2, 11, 9, 5];
	echo max($lista);
?>
```

Função **min()** retorna o menor número de uma lista.

```
<?php
	$lista = [8, 23, 4, 2, 11, 9, 5];
	echo min($lista);
?>
```

Com uso dessas duas últimas funções por exemplo, não é necessário fazer um comando de loop para verificação de qual valor é o menor ou maior.

Essas funções matemáticas são umas das principais que existe. Mas podem verificar todas as funções existentes na própria documentação do php no link abaixo:

https://www.php.net/manual/pt_BR/ref.math.php

Funções Nativas de String

Funções de manipulações de strings. Funções muito utilizadas na construção de sistemas. São várias funções de string, veja as principais:

Função **trim()** retira todos os espaços entre a string. Acontece quando tem um campo para o usuário digitar um dado e digita com espaços. Essa função é útil para corrigir casos assim.

```
<?php
    $nome = '      Sandra      ';
    echo trim($nome);
?>
```

Função **strlen()** retorna o tamanho da string.

```
<?php
    $nomeSujo = '      Sandra      ';
    $nomeLimpo = trim($nomeSujo);
    echo "Nome sujo: ".strlen($nomeSujo)."</br>";
    echo "Nome Limpo: ".strlen($nomeLimpo);
?>
```

Pode ver que na variável $nomeSujo, a função trim limpou os espaços extras antes e após da string. E a função strlen mostra o tamanho dessa string incluindo os espaços.

Função **strtolower()** transforma a string toda em minúscula.

```
<?php
    $nomeCompleto = 'Sandra Amorim';
    echo strtolower($nomeCompleto);
```

```
?>
```

Função **strtoupper()** transforma a string toda em maiúscula.

```
<?php
	$nomeCompleto = 'Sandra Amorim';
	echo strtoupper($nomeCompleto);
?>
```

Função str_replace() subtitui um caracter ou palavra que se está procurando por outra.

Sintaxe:

```
str_replace('procura', 'troca', busca_na_variavel)
```

```
<?php
	$nomeCompleto = 'Sandra Amorim';
	$nomeAlterado = str_replace('Amorim', 'Hedler', $nomeCompleto);
	echo $nomeAlterado;
?>
```

Outro exemplo de uso da função str_replace:

```
<?php
	$nomeCompleto = 'Sandra Amorim';
	$nomeAlterado = str_replace('a', '9', $nomeCompleto);
	echo $nomeAlterado;
?>
```

Função **substr()** que vem de substring, pega apenas uma parte da string.

Sintaxe:

substr(busca_Variavel, posicao_inicio, posicao_termina)

```
<?php
        $nomeCompleto = 'Sandra Amorim';
        //Aqui vou pegar apenas os 5 primeiros caracteres do nome completo
        $nome = substr($nomeCompleto, 0, 5);
        echo $nome;
?>
```

Observação: Se colocar no segundo parâmetro dessa função substr, um número negativo, vai começar da posição da direita para esquerda.

Exemplo:

```
<?php
        $nomeCompleto = 'Sandra Amorim';
        //Aqui vou pegar apenas os 5 primeiros caracteres do nome completo
        $nome = substr($nomeCompleto, -3, 3);
        echo $nome;
?>
```

Função **strpos()** busca determinado caracter ou palavra dentro de outra string e retorna a posição do que está procurando.

```
<?php
        $nomeCompleto = 'Sandra Amorim';
        $posicao = strpos($nomeCompleto, 'a');
        echo $posicao;
?>
```

Podemos combinar a strpos, com uma condição, por exemplo:

```
<?php
```

```
	$nomeCompleto = 'Sandra Amorim';
	$posicao = strpos($nomeCompleto, 'z');
	if($posicao > - 1){	//se menor que 0 não achou
		echo "Achou ! ";
	}else{
		echo "Nao achou !";
	}
?>
```

Função **ucfirst()** coloca em maiúscula apenas a primeira letra da primeira palavra.

```
<?php
	$nomeCompleto = 'sandra amorim';
	echo ucfirst($nomeCompleto);
?>
```

Função **ucwords()** coloca em maiúscula a primeira letra de cada palavra.

```
<?php
	$nomeCompleto = 'sandra amorim';
	echo ucwords($nomeCompleto);
?>
```

Função **explode()** que transforma cada item num array. Tem dois parâmetros, o primeiro o que vai procurar para ser o divisor, e o segundo é a própria string.

```
<?php
	$nomeCompleto = 'sandra amorim';
	$nome = explode(' ',$nomeCompleto); //qdo encontrar espaço vai ser o divisor
	print_r($nome);
?>
```

Conforme exemplo anterior, vai mostrará um array assim:

Array ([0] => sandra [1] => amorim)

Ou seja, um array com dois elementos, cada um ocupando uma posição.

Função **number_format()** formata um número com os milhares agrupados. O primeiro parâmetro é o próprio número, o segundo é a quantidade de decimais, o terceiro parâmetro é opcional e coloca o simbolo correspondente aos decimais, se for o padrão brasileiro é a vírgula. E o quarto parâmetro também é opcional e o símbolo para separar os milhares, no brasil é o ponto, assim como na Europa, na América do Norte é o inverso.

```
<?php
        $numero = 15567.12;
        echo "R$ ".number_format($numero, 2, ',', '.');
?>
```

Link para acesso as funções para string:

https://www.php.net/manual/pt_BR/ref.strings.php

Funções Nativas de Array

Funções nativas para manipulação de arrays.

Função **count()** mostra quantos itens tem no array.

```
<?php
        $lista = ['nome1', 'nome2', 'nome3', 'nome4'];
        echo "Total: ".count($lista);
?>
```

Função **array_diff()** pega a diferença entre o primeiro e o segundo, e gera um novo array com os itens da primeira lista que não estão na segunda. Que tem como primeiro parâmetro a lista geral, no segundo com quem quer fazer a comparação.

```
<?php
$lista = ['Caroline', 'Marcos', 'Edson', 'Alan', 'Carlos',  'Caren'];
```

```
$aprovados = ['Caroline', 'Caren', 'Marcos'];

$reprovados = array_diff($lista, $aprovados);

print_r($reprovados);

?>
```

Que mostra os dados num array: Array ([2] => Edson [3] => Alan [4] => Carlos)

Função **arry_filter()** filtra dados no array. Tem dois parâmetros, o primeiro é o próprio array e o segundo um callback function, que é uma função, temos que criar e mandar essa função.

```
<?php

$numeros = [10, 20, 25, 30, 70, 90, 16];

$filtrados = array_filter($numeros, function($item){ //vai fazer um loop em cada
                                                     //item, e receber aqui na $item

        if($item < 30){

                return true;

        }else{

                return false;

        }

});

print_r($filtrados);

?>
```

No exemplo acima, armazenou na variável $filtrado o array somente com os itens menores que 30: Array ([0] => 10 [1] => 20 [2] => 25 [6] => 16)

A função juntou todos os números que foram igual a true, gerou um novo array e armazenou na variável $filtrados.

Função **array_map()** similar ao array_filter, serve para executar alguma coisa. Tem dois parâmetros. O primeiro parâmetro é a função e no segundo parâmetro mandamos o array.

```
<?php

$numeros = [10, 20, 25, 30, 70, 90, 16];
```

```
$dobrados = array_map(function($item){
                    return $item * 2;
                }, $numeros);
print_r($dobrados);
?>
```

No exemplo acima, vai gera um loop no array e cada item vai rodar essa função dentro desse item. Fez um mapeamento no array $numeros e pegou item por item e multiplicou pelo valor 2. Mostra como saída o array:

Array ([0] => 20 [1] => 40 [2] => 50 [3] => 60 [4] => 140 [5] => 180 [6] => 32)

Função **array_pop()** remove o último item do array. Não precisa armazenar em nenhuma variável, recebe o parâmentro por referência.

```
<?php
$numeros = [10, 20, 25, 30, 70, 90, 16];
array_pop($numeros);
print_r($numeros);
?>
```

Mostra o array: Array ([0] => 10 [1] => 20 [2] => 25 [3] => 30 [4] => 70 [5] => 90)

Função **array_shift()** remove o primeiro item no array.

```
<?php
$numeros = [10, 20, 25, 30, 70, 90, 16];
array_shift($numeros);
print_r($numeros);
?>
```

Mostra o array: Array ([0] => 20 [1] => 25 [2] => 30 [3] => 70 [4] => 90 [5] => 16)

Tanto, o array_pop quanto array_shift passam o parâmetro por referência. Alteram o próprio array que especificamos.

Função **in_array()** busca dados no array, sem a necessidade de criar loop.

```
<?php
	$numeros = [10, 20, 25, 30, 70, 90, 16];
	if(in_array(90, $numeros)){ //vai procurar se existe o valor 90 dentro do array.
		echo 'Existe';
	}else{
		echo 'Nao existe';
	}
?>
```

Função **array_ search()** vai fazer uma busca e retornar a posição desse item. Se não achar o que busca não retorna nada.

```
<?php
	$numeros = [10, 20, 25, 30, 70, 90, 16];
	$pos = array_search(90, $numeros); //busca o valor 90 no array.
	echo $pos;
?>
```

Função **sort()** ordena o array em ordem crescente.

```
<?php
	$numeros = [10, 20, 25, 30, 70, 90, 16];
	sort($numeros);
	print_r($numeros);
?>
```

Mostra o array: Array ([0] => 10 [1] => 16 [2] => 20 [3] => 25 [4] => 30 [5] => 70 [6] => 90)

Função r**sort()** ordena o array em ordem decrescente.

```
<?php
    $numeros = [10, 20, 25, 30, 70, 90, 16];
    rsort($numeros);
    print_r($numeros);
?>
```

Mostra o array: Array ([0] => 90 [1] => 70 [2] => 30 [3] => 25 [4] => 20 [5] => 16 [6] => 10)

Função **asort()** ordena na forma crescente os itens no array, mantendo a chave do item no array.

```
<?php
    $numeros = [10, 20, 25, 30, 70, 90, 16];
    asort($numeros);
    print_r($numeros);
?>
```

Mostra o array: Array ([0] => 10 [6] => 16 [1] => 20 [2] => 25 [3] => 30 [4] => 70 [5] => 90)

Observem com uso do asort, manteve a chave associada originalmente a cada valor no array.

Função **arsort()** ordena de forma decrescente os itens no array, mantendo a chave associada do item no array.

```
<?php
    $numeros = [10, 20, 25, 30, 70, 90, 16];
    arsort($numeros);
    print_r($numeros);
?>
```

Mostra o array: Array ([5] => 90 [4] => 70 [3] => 30 [2] => 25 [1] => 20 [6] => 16 [0] => 10)

Função **implode()** junta dos dados em uma string. Como primeiro parâmetro espera um *glue*, que traduzindo significa cola, que vai juntar os itens. No segundo parâmetro mandamos o array.

```
<?php
```

```
    $nomes = ['Sandra', 'Hedler ', 'Amorim'];

    $nome = implode('', $nomes);

    //$nome = implode('@',$nomes); //aqui o @ junta

    echo $nome;

?>
```

Funções Nativas Data/Hora

No php, trabalhamos datas no formato de milessegundos. Temos como data 0 (zero) a data 01/01/1970. Data essa que o php começa a contar os milessegundos a partir desse ano 1970. Lembrando que tanto o php quanto o JS tem essa data como marco dos milessegundos.

```
<?php

    echo time();

?>
```

Essa função time retorna esse valor atualizado, contando da data 01/01/1970. Mas obviamente quando trabalhamos com data, não vamos exibir em milessegundos, exibimos a data real, no formato que estamos acostumados de visualizar.

Para isso usamos a função **date().** Essa função tem dois parâmetros, o primeiro é o formato que queremos exibir a data e o segundo parâmetro é opcional.

```
<?php

    echo date('d/m/Y'); //a letra Y em maiúscula.

    echo "</br>".date('d/m/Y H:i:s'); //data e hora

    echo "</br>".date('d/m/y'); //a letra y em minúsculo

    echo "</br>".date('d - m - Y');

    echo "</br>".date('d');

?>
```

Observem: Se ao rodar o exemplo acima, der diferença no horário, significa que seu servidor esteja configurado para horário greenwich. Vai pegar basicamente fuso horário da inglaterra. A hora padrão no mundo é a hora em algum lugar específico da Europa, na linha de greenwich. Onde estiver localizado em relação a isso, poderás ter horas positiva ou negativa.

No link a seguir, podem consultar como aplicar esses parâmetros na função date:

https://www.php.net/manual/pt_BR/function.date.php

Consultem e observem o por que das letras em maiúsculas e minúscula aplicadas nos parâmetros da função.

ATENÇÃO:

Quando começarmos a trabalhar com banco de dados, guardar informações, a data específicada sempre é salva no padrão internacional.

Formato do padrão internacional: ano, mês, dia.

date(Y - m - d)

Padrão americano: ano, dia, mês.

date('Y - d - m')

```
<?php
    echo date('Y-m-d');
?>
```

Então, quando salvamos uma data em algum lugar, salvamos no formato internacional.

Como pegamos uma data no formato internacional e convertemos para o formato que desejar ? Tem varias formas, uma é usar a função **strtotime()** que transforma o formato para time. Equivalente em milessegundo da data passa no parâmetro.

```
<?php
    $data = '2020-06-25';
    $time = strtotime($data);
    echo $time;
?>
```

Observem, que tendo o time específico podemos usar o segundo parâmetro da função date. Por exemplo:

```
<?php
    $data = '2020-06-25';
    $time = strtotime($data);
    echo date('d/m/Y', $time);
```

```
        echo "</br>".date('d/m/Y', 0); //aqui vai retornar de acordo com o time especificad
?>
```

Podemos exibir a data no formato que desejar, veja exemplo abaixo:

```
<?php
        $data = '2020-06-25';
        $time = strtotime($data);
        echo date('d/m/Y', strtotime($data));
?>
```

Exercício Prático (Funções)

Fazer uma função para receber uma string com uma data específica no padrão internacional. Enviar essa string pra uma função que você vai criar, a qual deverá retornar o dia da semana em português. Lembrando que não precisa retornar a data, só o dia da semana correspondente a essa data.