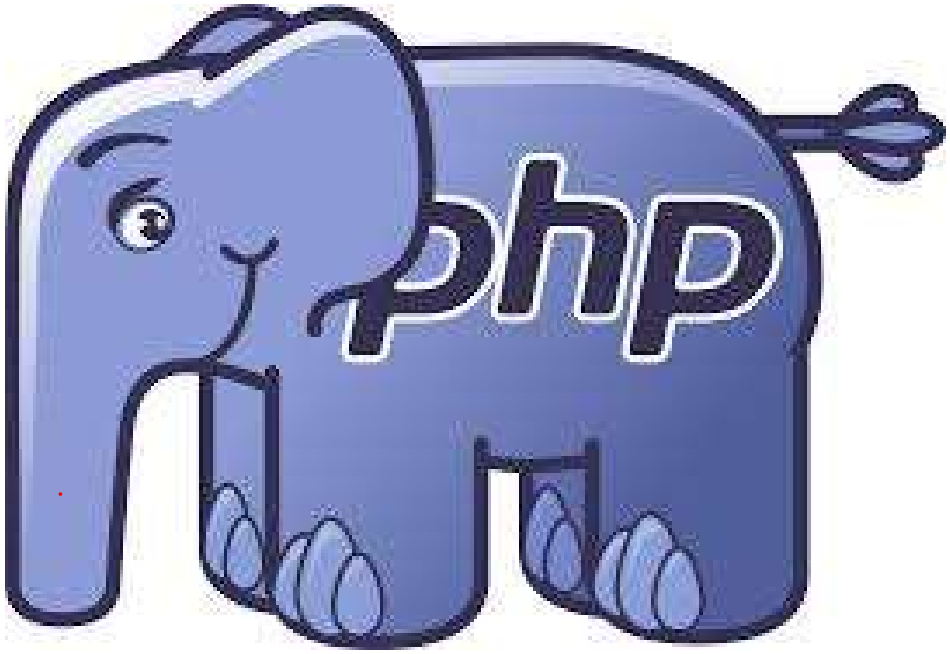


AULA 3

EXPLICADORES.NET

@explicadoresnet

As variáveis comuns (também chamadas de variáveis escalares) podem armazenar apenas um valor por vez. Um array (vetor) pode armazenar vários valores ao mesmo tempo, pois se trata de uma estrutura de armazenamento que, assim como as variáveis, possui um identificador, mas além disso há um índice associado (que pode ser um número ou um texto), e cada índice indica uma posição de memória em que fica armazenado um elemento do array. O índice deve aparecer entre colchetes ([]) logo após o identificador do array.

Arrays

```
1 $aluno [0] = 8.0;
  $aluno [1] = 9.0;
  $aluno [2] = 2.5;

  $aluno [] = 8.7;     // Acrescenta ao próximo espaço livre

2 $notas = array range (1, 10);

3 $notas = array (8.5, 7.0, 4.5, 8.0);
  ARRAY [8.5, 7.0, 4.5, 8.0];

4 $cadastro ["nome"] = "José";
  $cadastro ["idade"] = 23;
  $cadastro ["peso"] = 93.5;   // Chave associativa

5 $pessoa = array ("nome" => "Paulo", "idade" =>30);
```

1 → $ALUNO | 8.0 | 9.0 | 2.5 | 8.7 | índices 0 1 2 3

2 → $NOTAS | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | índices 0 1 2 3 4 5 6 7 8 9

3 → $NOTAS | 8.5 | 7.0 | 4.5 | 8.0 | índices 0 1 2 3

4 → $CADASTRO (STRING) | José | 23 | 93.5 | NOME | IDADE | PESO

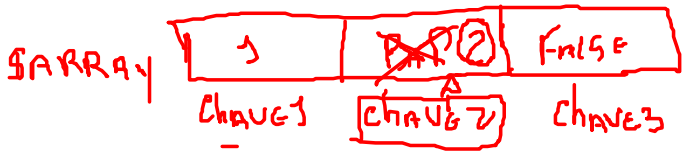
5 → $PESSOA | PAULO | 30 | NOME | IDADE

```php
$array = array(1, 2, 3);
```

```php
$array = [1, 2, 3];
```

```php
$array = array(
  "chave1" => 1,
  "chave2" => "PHP",
  "chave3" => false
);
```

```php
$notas = array range (1, 10);
$cadastro ["nome"] = "José";
$cadastro ["idade"] = 23;
$cadastro ["peso"] = 93.5;
$aluno [] = 8.7;     // Acrescenta ao próximo espaço livre
```




```php
$aluno [0] = 8.0;
$aluno [1] = 9.0;
$aluno [2] = 2.5;
```

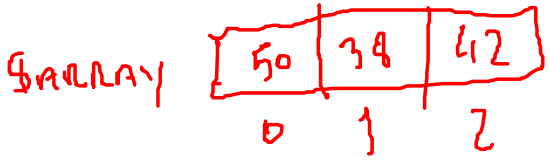
EXPLICADORES.NET

@explicadoresnet

Acessando os índices de um array

```php
echo $array[0];
echo $array[1];
echo $array[2];
```

50
34
42




```php
echo $array["chave2"];
```

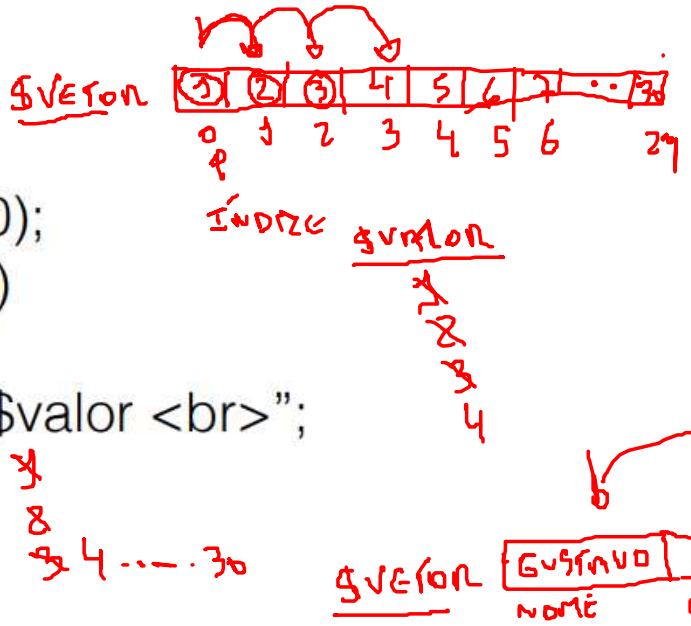
PHP

```php
$array["chave2"] = 2;
```


Para Cada

FOR APRIMORADO

```
$vetor = array range(1,30);
foreach ($vetor as $valor)
{
    echo "O valor atual é $valor <br>";
}
```

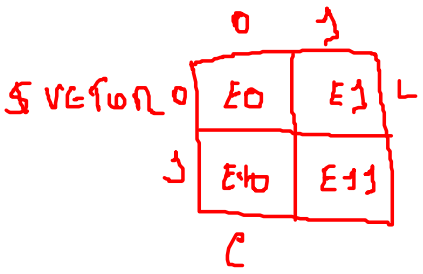



```
$vetor = array ("nome" => "Gustavo", "profissão" =>
"Professor", "idade" => 36);
foreach ($vetor as $chave => $valor)
{
    echo "Campo $chave contém $valor <br>";
}
```

$chave	$valor
NOME	GUSTAVO
PROFISSÃO	PROFESSOR
IDADE	36

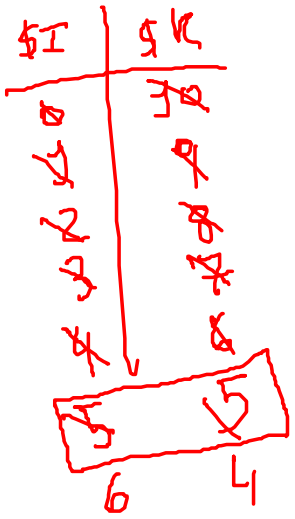
```php
<?php
    $vetor[0][0]= "elemento00";
    $vetor[0][1]= "elemento01";
    $vetor[1][0]= "elemento10";
    $vetor[1][1]= "elemento11";
    for($i=0 ; $i<2 ; $i++)
    {
        for($k=0 ; $k<2 ; $k++)
        {

            echo "O elemento da posição $i,$k é ";
            echo $vetor [$i][$k] . "<br>";
        }
    }
```



$I	$k
0	0
1	1
	0
	1

E0 E1 E10 E11



```php
<?php
    for( $i=0, $k=10  ;  $i<10  ;  $i++, $k--)
    {
        echo "\$i vale $i e \$k vale $k";
        if ($i==$k)
          { echo " (os valores são iguais!)"; }
        echo "<br>";
    }
?>
```


Controlando o fluxo de execução

Existem comandos que podem ser usados em conjunto com essas estruturas de controle que acabamos de ver. Esses comandos são o *break* e o *continue*.

Vamos ver qual é a utilidade de cada um deles:

break

O *break* termina a execução do comando atual, que pode ser um *if*, *for*, *while*, ou *switch*. Quando o *break* é encontrado dentro de algumas dessas estruturas, o fluxo de execução passa para o primeiro comando após o término dessa estrutura. É um recurso que podemos utilizar para forçar a saída de um laço ou de um comando condicional. Acompanhe o exemplo a seguir:

Interrompendo o fluxo

```
$vetor = array (1, 4, 6, 2, -1, 8, 7, 5);
foreach ($vetor as $valor)
{
    if ($valor == -1) {
        break;
    }
    echo "O valor atual é $valor <br>";
}
```

1 4 6 2

$valor

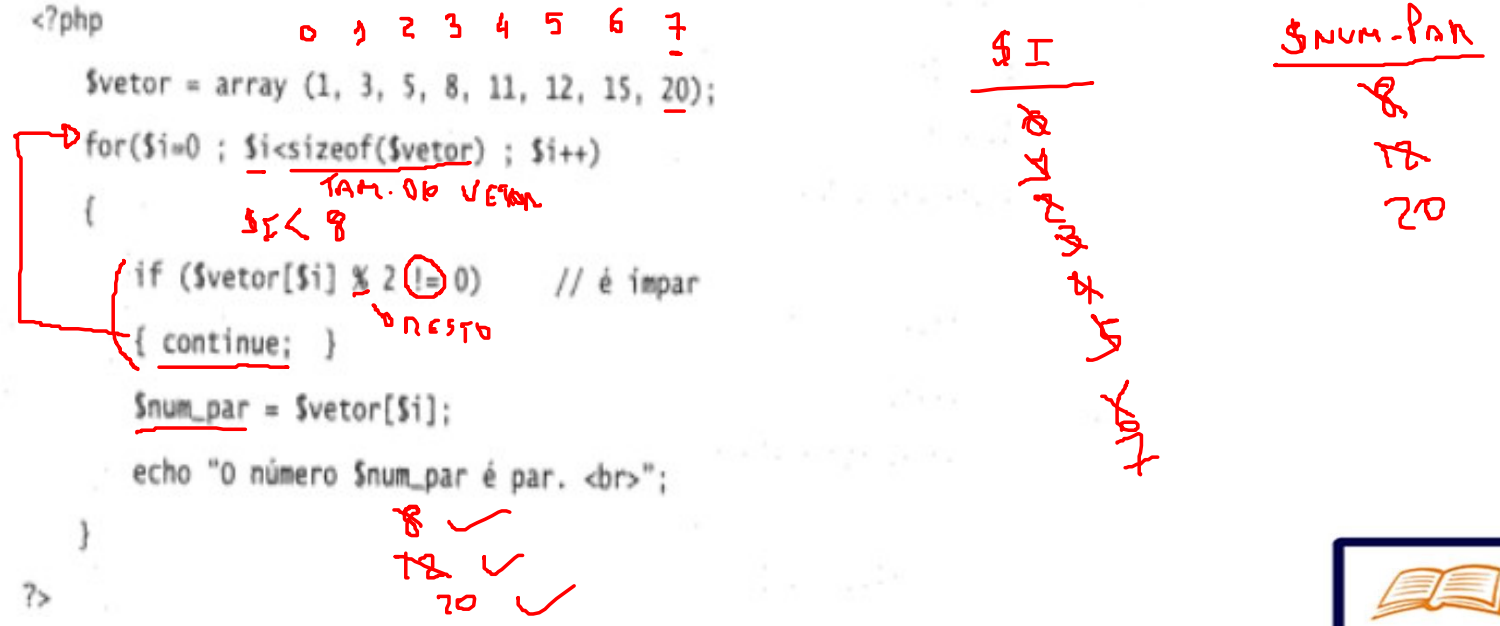
1 4 6 2 -1

continue

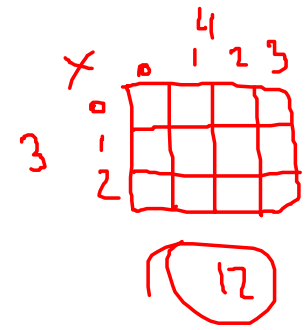
O comando *continue* é utilizado dentro dos comandos de repetição para ignorar o restante das instruções pertencentes ao laço corrente, e ir para a próxima iteração (voltando ao início do laço). Veja o exemplo a seguir:

exemplo5_13.php

```
<?php
    $vetor = array (1, 3, 5, 8, 11, 12, 15, 20);
    for($i=0 ; $i<sizeof($vetor) ; $i++)
    {
      if ($vetor[$i] % 2 != 0)     // é impar
      { continue; }
      $num_par = $vetor[$i];
      echo "O número $num_par é par. <br>";
    }
?>
```

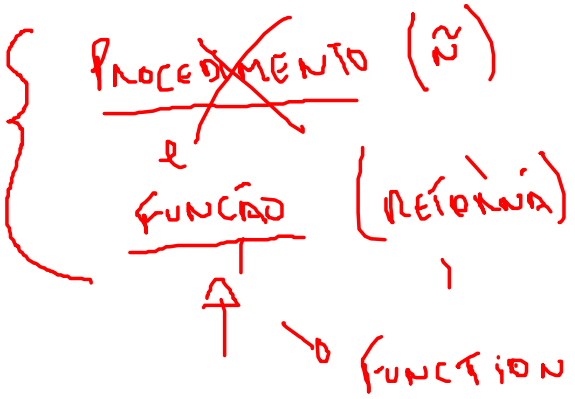


Modificando o fluxo



```
$vetor = array (1, 4, 6, 2, -1, 8, 7, 5);
for ($i = 0; $i < sizeof($vetor); $i++)
{
  if ($vetor[$i] % 2 == 0) {
    continue;
  }
  echo "O valor atual é $valor <br>";
}
```


Rotinas



Funções

```
function soma ($a, $b, $c) {
    $s = $a + $b + $c;
    echo "A soma entre $a , $b e $c é igual a $s";
}

function soma ($a, $b, $c) {
    $s = $a + $b + $c;
    return $s;
}
```

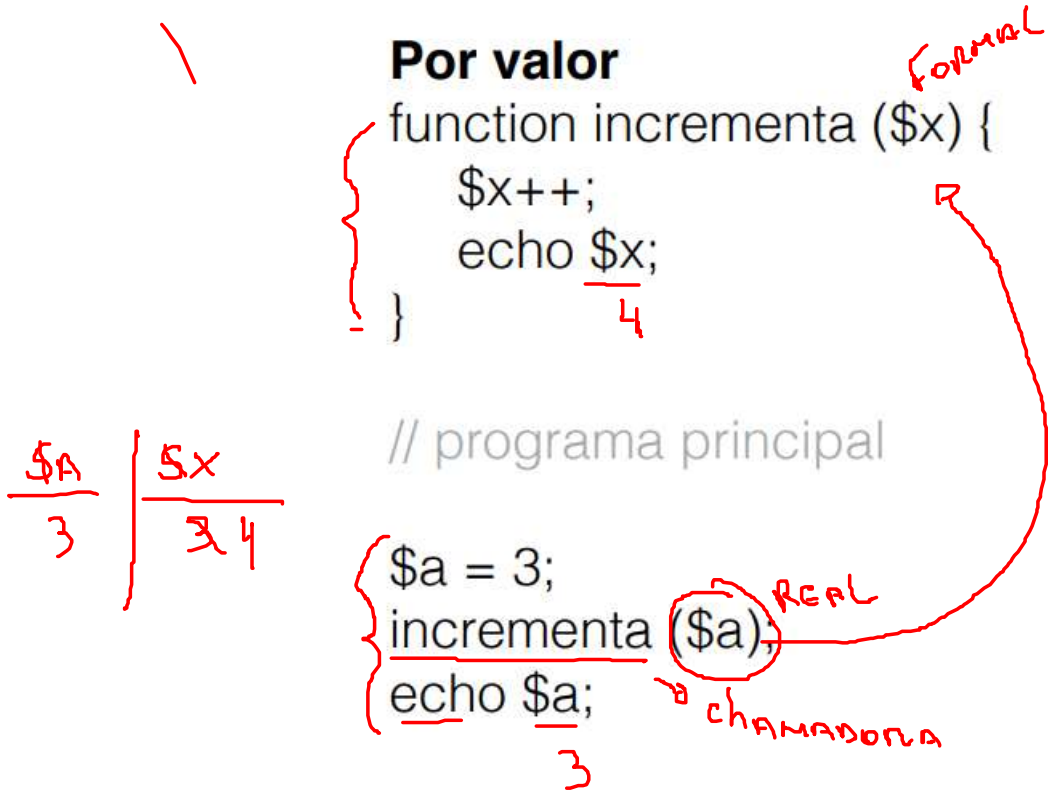

Passagem de parâmetro

Por valor

```
function incrementa ($x) {
    $x++;
    echo $x;
}

// programa principal

$a = 3;
incrementa ($a);
echo $a;
```



Passagem de parâmetro

Por referência

```
function incrementa (&$x) {
    $x++;
    echo $x;
}

// programa principal

$a = 3;
incrementa ($a);
echo $a;
```

Considere a seguinte função:

```
function nada ($a=10 , $b, $c)
{
    ...
}
```

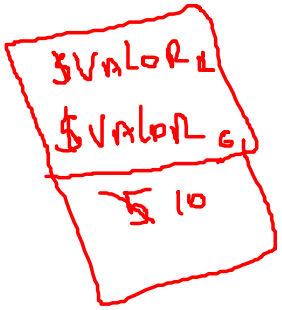
Essa definição está errada, pois como o envio do parâmetro $a é opcional, poderíamos fazer a seguinte chamada:

```
nada (1, 5);
```

Isso causaria um erro, porque o PHP interpretará os dois parâmetros passados como $a e $b, e ocorreria um erro de execução devido à falta do parâmetro $c. Portanto, o modo correto de definir a função seria:

```
function nada ($b, $c, $a=10)
{
    ...
}
```


```php
<?php
function dobro ($valor)
{
    $valor = 2 * $valor;
}
function duplica (&$valor)
{
    $valor = 2 * $valor;
}
$valor = 5;
dobro ($valor);
echo $valor . "<br>";
duplica ($valor);
echo $valor;
?>
```



Funções recursivas

Chamamos de funções recursivas aquelas funções que chamam a elas mesmas. Veja um exemplo simples desse tipo de função:

exemplo6_10.php

```
<?php
    function teste ($valor)
    {
        if ($valor!=0)
        {
            echo "Foi chamada a função teste passando o valor $valor <br>";
            teste ($valor-1);
        }
    }
    teste (7);
?>
```


$A = "ALEX";

$$A = 20;

$A → ALEX

$ALEX → 20

```php
<?php
$Fab= 20;
$b=&$Fab;
$b=5;
echo $Fab;
```

$b → 20

5

```php
<?php
for ($i = 1; $i <= 10; ++$i) {
    echo $i;
    $i++;
}
```

1 3 5 7 9

$I → 1

```php
<?php
    function incrementa(&$x){
        $x++;
        global $a;
        echo "valor de x: $x <br>";

        echo "valor de a: $a <br> ";
    }

    $a=3;
    incrementa($a);
    echo "PP valor de a: $a";
?>
```

valor de x: 4
valor de a:
PP valor de a: 4

```php
<?php
    function incrementa(&$x){
        $x++;
        global $a;
        echo "valor de x: $x <br>";

        echo "valor de a: $a <br> ";
    }

    $a=3;
    incrementa($a);
    echo "PP valor de a: $a";
?>
```