

# Programação II

Professores:

Claudio Esperança

Inês Dutra

## Aula 4

Professor:

*Inês Dutra*

Comandos simples, loops,  
condicionais e arrays

# PHP: Comandos simples

- **echo**: já foi bastante usado anteriormente
- Atribuições: também bastante utilizadas nos exemplos anteriores
- Incremento de variáveis: também já utilizado
- **exit** or **die**: utilizados para terminar o programa
- Chamadas de funções definidas pelo usuário (discutidas mais tarde)

# PHP: Condicionais

- Comandos condicionais muito semelhantes aos de qualquer linguagem

```
if <condição>
    <comando>

if <condição>
    <comando>
else
    <outro comando>
```

# PHP: Condicionais

- Para blocos de comandos: utilizar chaves

```
if <condição>
→ {
    <comandos>
→ }
if <condição>
→ {
    <comandos>
→ }
else
→ {
    <comandos>
→ }
```

# PHP: Condicionais

- Outra forma de escrever código com **if-else**:

```
if ($var < 5)
    echo "Valor menor do que 5";
elseif ($var < 10)
    echo "Valor entre 5 e 10";
elseif ($var < 20)
    echo "valor entre 10 e 20 ";
```

# PHP: Condicionais

- Outro tipo de **condicional** com várias alternativas: **switch**

```
switch ($menu)
{
    case 1:
        echo "Você selecionou 1";
        break; ←
    case 2:
        echo "Você selecionou 2";
        break; ←
    case 3:
        echo "Você selecionou 3";
        break; ←
    default:
        echo "Opção inválida";
}
```

# PHP: Condicionais

- Operadores lógicos: `&&`, `||`
- Complementam os operadores relacionais já estudados.

```
if ($var == 5) || ($var == 6)
    echo "valor igual a 5 ou 6";
```

# PHP: Condicionais

- Exemplos de comparações:

```
if (5.0 === 5)  
    echo "nunca entra aqui";
```

```
if (5.0 == 5)  
    echo "sempre passa por aqui";
```

[Abrir editor](#)[Abrir navegador](#)

# PHP: Loops

- Loops de propósito geral:
  - ▶ `while`
  - ▶ `do...while`
  - ▶ `for`
- Loop para **arrays**:
  - ▶ `foreach`

# PHP: Loops

- Exemplos:

```
$contador = 1;  
while ($contador < 11)  
{  
    echo $contador;  
    echo " ";  
    $contador++;  
}
```

[Abrir editor](#)[Abrir navegador](#)

# PHP: Loops

- **Do...while**

```
$contador = 1;  
do  
{  
    echo $contador;  
    echo " ";  
    $contador++;  
} while ($contador < 11);
```

[Abrir editor](#)[Abrir navegador](#)

# PHP: Loops

- **for**

```
for ($cont=1; $cont<11; $cont++)  
{  
    echo $cont;  
    echo " ";  
}
```

[Abrir editor](#)[Abrir navegador](#)

# PHP: Loops

- Três partes opcionais:
  - ▶ Comandos iniciais
  - ▶ Condições do loop
  - ▶ Comandos de término do loop
- Pode ser complexo

# PHP: Loops

- Exemplo:

```
for ($x=0, $y=0; $x<10, $y<$z; $x++, $y+=2)
```

- Atenção:

► Comandos deste tipo podem comprometer a legibilidade do código

# PHP: Loops

- Algumas vezes é necessário terminar um loop antes da condição ser alcançada
- Neste casos, utiliza-se
  - ▶ break
  - ▶ continue

# PHP: Loops, usando break

```
$cont = 0;  
while ( $cont < 5 )  
{  
    $cont++;  
    if ( $cont == 3 )  
    {  
        echo "break<br>";  
        break;  
    }  
    echo "Fim do while: cont=$cont<br>";  
}  
echo "Depois do break<p>";
```

[Abrir editor](#)[Abrir navegador](#)

# PHP: Loops, usando continue

```
$cont = 0;  
While ( $cont < 5 )  
{  
    $cont++;  
    if ( $cont == 3 )  
    {  
        echo "continue<br>";  
        continue;  
    }  
    echo "Fim do while: cont = $cont<br>";  
}  
echo "Depois do loop<br>";
```

[Abrir editor](#)[Abrir navegador](#)

# PHP: Loops, Programa Exemplo

- Programa que cria tabuadas de multiplicação de 1 a 12
- Programa simples, que não necessita ser escrito em PHP, nem aproveita os recursos da linguagem, mas ilustrativo e didático
- Pode sobrecarregar o servidor
- Clique nos ícones correspondentes para ver o fonte PHP e o resultado da execução do programa

[Abrir editor](#)[Abrir navegador](#)

# PHP: Arrays

- **Arrays** em PHP são estruturas de dados mais sofisticadas e flexíveis do que em outras linguagens
- Um array é uma seqüência ordenada de **elementos**
- Arrays podem ser indexados numericamente ou por strings
- Um array pode conter valores do tipo inteiro, booleano, string, ponto flutuante ou objetos compostos: objetos e mesmo arrays

# PHP: Arrays

- Por que utilizar arrays?
  - ▶ Porque podemos representar uma lista de valores que pertencem a uma mesma categoria sob um mesmo nome e realizar operações conjuntas sobre esta lista de valores

# PHP: Criação de arrays

- Utiliza-se a função **array()**

```
$numeros = array(5, 4, 3, 2, 1);
$palavras = array("Web", "Database", "Applications");
// Escrever o terceiro elemento do array de inteiros
echo $numeros[2]; // escreve o número 3
// Escrever o primeiro elemento do array de strings
echo $palavras[0] // escreve a palavra Web
```

NB: o índice do primeiro elemento é zero (0)

# PHP: Arrays

- Arrays podem ser **criados** para começar com qualquer índice

```
$numeros = array(1=>"um", "dois", "tres", "quatro")
```

Este exemplo define um array de nome **\$numeros** com 4 elementos, onde o primeiro elemento é obtido através do índice 1

# PHP: Arrays

- Outras formas de inicializar elementos de um array:

```
$animal[] = "tigre";  
$animal[] = "girafa";  
$animal[] = "elefante";
```

- Esta seqüência inicializa um array de nome **\$animal** com três valores onde o primeiro elemento é indexado de 0.

# PHP: Arrays (mais exemplos)

```
$moeda =  
    array("BR"=>"Real", "EUA"=>"dolar");
```

- Neste exemplo, o array `$moeda` é criado com índices do tipo string:

```
$moeda['BR'] = "Real";  
$moeda['EUA'] = "Dolar";
```

[Abrir editor](#)[Abrir navegador](#)

# PHP: Arrays

- Como escrever valores de arrays?
- Exemplo:

```
echo $animal[0]
```

- Pode ser necessária utilização de chaves:

```
echo "A unidade monetária brasileira é o $moeda['BR'];"
```

# PHP: Arrays

- Outra forma de escrever arrays:

```
print_r($moeda);
```

- Resultado:

```
Array
(
    [BR] => Real
    [EUA] => Dolar
)
```

[Abrir editor](#)[Abrir navegador](#)

# PHP: Arrays

- Utilizando somente o comando `print_r`, o texto não fica formatado, pois o navegador interpreta o texto como se fosse HTML
- É necessário dizer ao navegador que o texto formatado PHP não deve ser convertido para HTML
- Neste caso, o comando deve ser:

```
echo "<pre>";  
print_r($moeda);  
echo "</pre>";
```

[Abrir editor](#)[Abrir navegador](#)

# PHP: Arrays

- Para remover elementos de um array:
  - ▶ Coloque string vazia na posição que desejar
    - ex: `$moeda['EUA'] = "";`
  - ▶ Utilize o comando unset
    - `unset ($moeda['EUA']);`
- No primeiro caso, o array continua com 2 elementos, porém o valor do segundo elemento foi apagado
- No segundo caso, o array realmente fica com a dimensão menor, passando a ter apenas 1 elemento

[Abrir editor](#)[Abrir navegador](#)

# PHP: Ordenação de Arrays

- Utilize **sort** para ordenar arrays que tenham chaves numéricas (caso do array **\$animal**)

```
sort($animal);
```

```
$animal[0] = "tigre";  
$animal[1] = "girafa";  
$animal[2] = "elefante";
```



```
$animal[0] = "elefante";  
$animal[1] = "girafa";  
$animal[2] = "tigre";
```

- Utilize **asort** para ordenar arrays que tenham chave alfanumérica (caso do array **\$moeda**)

[Abrir editor](#)[Abrir navegador](#)

# PHP: Arrays

- Atenção:
- **asort (\$animal)** não modifica as chaves!!!

```
$animal[0] = "tigre";  
$animal[1] = "girafa";  
$animal[2] = "elefante";
```



```
$animal[2] = "elefante";  
$animal[1] = "girafa";  
$animal[0] = "tigre";
```

# PHP: Arrays

- Outras formas de ordenação:
  - ▶ **rsort**: ordenação reversa de valores, muda as chaves numéricas
  - ▶ **arsort**: ordenação reversa de valores, mantém as chaves
  - ▶ **ksort**: ordenação de chaves
  - ▶ **krsort**: ordenação reversa de chaves
  - ▶ **usort**: ordena a partir de uma função de ordenação indicada pelo usuário

# PHP: Arrays

- Outras formas de obter elementos de arrays:
  - ▶ `list`
  - ▶ `extract`

# PHP: Arrays

- Uso de `list`

```
$tshirtinfo = array("Tam"=>"g", "cor"=>"azul",
"preço"=>12.00);
asort($tshirtinfo);
list($primvalor,$segvalor)=$tshirtinfo;
echo $primvalor,<br>;
echo $segvalor,<br>;
```

Resultado:

azul  
g

Abrir editor

Abrir navegador

# PHP: Arrays

- Uso de **extract**

```
$Tam = "m";  
$tshirtinfo = array("Tam"    => "g",  
                     "cor"   => "azul",  
                     "preço" => 12.00);  
  
asort($tshirtinfo);  
  
extract($tshirtinfo, EXTR_PREFIX_SAME, "xxx");  
echo $Tam, " ", $cor, " ", $preço, " ", $xxx_Tam;
```

Resultado:

m azul 12.00 g

Abrir editor

Abrir navegador

# PHP: Arrays

- Funções que permitem percorrer um array:
  - ▶ De forma manual: um ponteiro é associado ao array, e se, nenhum elemento foi acessado até o momento, este ponteiro aponta para o primeiro elemento do array
    - `current`, `next`, `previous`, `end`, `reset`
  - ▶ De forma automática
    - `foreach`

# PHP: Arrays

- **current (\$arrayname)** : devolve o elemento corrente do array. Se nenhum elemento foi acessado até o momento, devolve o primeiro elemento do array. Não move o ponteiro
- **next (\$arrayname)** : move o ponteiro para o próximo elemento e devolve o valor correspondente

# PHP: Arrays

- **previous (\$arrayname)** : move o ponteiro para o elemento anterior ao elemento corrente e devolve o valor do novo elemento
- **end (\$arrayname)** : move o ponteiro para o último elemento do array
- **reset (\$arrayname)** : move o ponteiro para o início do array, ou seja, posiciona o ponteiro no primeiro elemento

# PHP: Arrays

- Vantagem de se percorrer arrays de forma manual:  
flexibilidade
  - ▶ Podemos percorrer o array em qualquer ordem
  - ▶ Podemos pular valores
- Muitas vezes, precisamos percorrer o array do início até o final, um valor de cada vez. Para isto, utilizamos  
**foreach**

# PHP: Arrays

- Formato geral do **foreach**

```
foreach ($array as $key => $value)
{
    bloco de comandos;
}
```

# PHP: Arrays

- Exemplo:

```
$capitais =  
    array("PA" => "Belém",  
          "AM" => "Manaus",  
          "PI" => "Teresina");  
  
ksort($capitais);  
  
foreach ($capitais as $estado => $cidade)  
{  
    echo "$cidade, $estado<br>";  
}
```

[Abrir editor](#)[Abrir navegador](#)

# PHP: Arrays Multidimensionais

- Por que precisamos de arrays multidimensionais?
- Por que muitas vezes, não temos os dados organizados em uma única categoria
- Neste caso, precisamos de uma forma que represente múltiplas categorias
- Um array multidimensional tem esta finalidade

# PHP: Arrays Multidimensionais

- Exemplo: suponha que temos uma lista de produtos numa loja de departamentos e queremos relacionar os preços dos produtos
- Se utilizarmos um array de uma única dimensão para indexar produtos, a busca por um produto pode ser muito ineficiente
- Por outro lado, podemos indexar os produtos por categoria/nome

# PHP: Arrays Multidimensionais

- Exemplo:

```
$prod['roupa']['tshirt'] = 12.00;  
$prod['roupa']['calças'] = 32.00;  
$prod['cama']['colcha'] = 45.00;  
$prod['cama']['lençol'] = 22.00;  
$prod['móvel']['mesa'] = 120.00;  
$prod['móvel']['sofá'] = 640.00;
```

# PHP: Arrays Multidimensionais

- Este array tem duas dimensões, onde a primeira dimensão tem 3 chaves: **roupa**, **cama** e **móvel**
- O valor de cada chave é um outro array de chaves/valores
- Por exemplo, o valor da chave **roupa** é um array com o par de valores: **tshirt/12.00** e **calças/32.00**

# PHP: Arrays Multidimensionais

- Como acessar elementos de um array multidimensional utilizando **foreach**?

```
echo "<table border=1>";  
foreach ( $prod as $categoria )  
{  
    foreach ( $categoria as $pr => $preço )  
    {  
        $f_preço = sprintf("%01.2f", $preço);  
        echo "<tr><td>$pr:</td>  
              <td>R\$f_preço</td></tr>";  
    }  
}  
echo "</table>";
```

[Abrir editor](#)[Abrir navegador](#)

# PHP: Arrays

- Como obter o número de elementos de um array, cujo tamanho é desconhecido:
  - ▶ `sizeof($arrayname)`