Programação II

Sintaxe Básica
Estruturas de Controle
Funções

Delimitando o código PHP

- Para que o interpretador PHP reconheça o que é código HTML (ignorado por ele) e código PHP (processado por ele), este deverá estar separado do código HTML. Para isso, utilizamos a seguinte sintaxe:

```
<?php
  // linha de comando PHP 1;
  // linha de comando PHP 2;
   ...
   ...
  // Linha de comando PHP n;
?>
```

- O marcador **<?php** inicia o código PHP e o marcador **?>** finaliza o código PHP. Antes do inicio do código PHP e/ou após o mesmo, teremos tags HTML.

  - OBS: Se a opção short_open_tag no arquivo php.ini estiver em **on**, a tag **<?** pode ser utilizada no lugar de **<?php**.

Delimitando o código PHP

- Quando o Servidor HTTP encontra as tags **<?php** e **?>,** ele automaticamente inicia o interpretador PHP;
- Entre as tags **<?php** e **?>** pode-se escrever programas utilizando todos os recursos do PHP, como definição e chamada de funções, acesso a banco de dados, atribuição de valores a variáveis, fórmulas matemáticas, loops, ifs, etc;
- Poderemos ter vários trechos HTML intercalados com códigos PHP em um documento HTML, conforme podemos observar a seguir:

```
/// código HTML
/// código HTML
<?php
  /// código PHP;
?>
/// código HTML
/// código HTML
/// código HTML
<?php
  /// código PHP;
?>
/// código HTML
```

As tags HTML devem aparecer fora das tags **<?php ?>**, pois estas limitam um trecho de programa onde somente poderão aparecer comandos da linguagem PHP.
A única maneira de inserir tags HTML em trechos PHP seria utilizando os comandos ECHO ou PRINT para escrevê-los.

Um exemplo simples

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Exemplo</title>
    </head>
    <body>
        <p><?php echo "Olá!<br>Eu sou um script PHP!";?></p>
        <p>Hoje é dia <?php echo date('d/m/y'); ?></p>
        <p>E agora são <?php echo date('h:i:s'); ?></p>
    </body>
</html>
```

- O comando **echo** tem a função de enviar para a janela do navegador qualquer conteúdo, como por exemplo, frases e conteúdo de variáveis. Além do **echo**, pode-se utilizar o comando **print**.
- Uma sintaxe alternativa para `<?php echo $var;?>` é `<?=$var;?>`
- A função date retorna informações sobre data/hora, de acordo com os parâmetros recebidos.
  - Se a hora estiver errada, configure **date.timezone=America/Sao_Paulo** no php.ini e reinicie o Apache.
- O arquivo acima deve ser salvo com a extensão .php, dentro da pasta raiz do servidor HTTP.

Comentários

- No PHP, existem três tipos de marcadores de comentário, que são:

  - `//` e `#`, para comentários de apenas uma linha

```
<?php  echo "teste"; #isto é um teste  ?>
<?php  echo "teste"; //este teste é similar ao anterior ?>
```

  - `/* */` , para comentários de múltiplas linhas

```
<?php echo "teste"; /* Isto é um comentário com mais
de uma
linha  */
?>
```

Variáveis

- Toda variável em PHP tem seu nome composto pelo caracter $ e uma string, que deve iniciar por uma letra ou o caracter “_”.
- PHP é case sensitive, ou seja, diferencia letras maiúsculas e minúsculas.
  - `$nome` e `$Nome` são variáveis diferentes;
- Não é necessário declarar variáveis em PHP, bastando apenas atribuir-lhe um valor. A variável será criada e associada a um tipo (numérica, string, etc.), dependendo do valor que foi atribuído.
- Para armazenar um conteúdo qualquer na variável, deveremos utilizar a seguinte sintaxe:

```
$variavel = conteudo;
```

- Após ter armazenado um conteúdo qualquer em uma variável, você pode utilizar o comando **echo** para exibir o seu valor.

Strings

- Valores de Strings podem ser atribuídos de duas maneiras:
  - utilizando aspas simples (') – Desta maneira, o valor da variável será exatamente o texto contido entre as aspas
  - utilizando aspas duplas (") – Desta maneira, qualquer variável ou caracter de escape será expandido antes de ser atribuído.

```php
<?php
$teste = "testando";
$var = '---$teste---'; // aspas simples
echo $var;
?>
```

A saída desse script será ---$teste---

```php
<?php
$teste = "testando";
$var = "---$teste---"; // aspas duplas
echo $var;
?>
```

A saída desse script será ---testando---

Operadores do PHP

- O operador de atribuição é o `"="`
- O operador de concatenação de strings é o `"."`
- Os operadores aritméticos suportados são `+`, `-`, `*`, `/` e `%` (módulo)
- Operadores de comparação:

```
== ➔ igual
!= ➔ diferente de
>  ➔ maior que
<  ➔ menor que
>= ➔ maior igual a
<= ➔ menor igual a
```

Operadores do PHP

- Operadores Lógicos:

```
and → Operador and
&&  → Operador and (maior precedência)
or  → Operador or
||  → Operador or (maior precedência)
xor → Operador xor
!   → Operador not
```

- De incremento e decremento:

```
++ → incremento
-- → decremento
```

Constantes

- Para definir constantes dentro de um script PHP, utilizamos a função define(nome_da_constante, valor):

```
<?php
   define ("pi", 3.1415926536);
   $raio = 4;
   $circunf = 2 * pi * $raio;
   echo $circunf;
?>
```

Arrays

- Existem 3 tipos de arrays:
  - Numéricos: índices numéricos (padrão)
  - Associativos: cada chave de identificação é associada a um valor
    - chave => valor
  - Multidimensional (arrays de arrays)

Arrays Numéricos

```php
<?php
// índices atribuídos automaticamente:
$frutas = array("banana","maçã","kiwi","limão","morango");
echo "A fruta escolhida foi " . $frutas[3] . "<br>";

// índices atribuídos manualmente:
$verduras[0] = "Alface";
$verduras[1] = "Rúcula";
echo $verduras[0]. "<br>"; // imprime um valor

foreach ($verduras as $item){ // imprime todos
  echo $item."<br>";
}
?>
```

Arrays Associativos

```php
<?php
// nomes das frutas são as chaves:
$cor = array("banana" => "amarela", "kiwi" => "verde");
// ou
$cor["banana"] = "amarela";
echo $cor["banana"]; // imprime somente um valor

foreach($cor as $fruta=>$corFruta){ // imprime todos
  echo $fruta . " possui a cor " . $corFruta ."<br>";
}
?>
```

Arrays Multidimensionais

```
<?php
$vegetais = array(
    "frutas" => array("banana", "maçã", "kiwi"),
    "verduras" => array("alface", "rúcula"),
    "legumes" => array("cenoura","batata")
);
echo "Adoro ". $vegetais['verduras'][0]."!"; //imprime: Adoro alface!

echo "<pre>";
print_r($vegetais); // imprime todo o array em formato indentado
echo "</pre>";
?>
```

Arrays

- Para visualizar o conteúdo de um array (para fins de debug), podemos utilizar as funções **var_dump** (mostra o conteúdo e tipo de dados) ou **print_r** (sem tipo de dados, mais legível).
- Por exemplo, o código abaixo:

```
<?php
$frutas = array("banana", "maçã", "kiwi", "limão");
print_r($frutas);
?>
```

- Produzirá no navegador o resultado:

```
Array ([0] => banana [1] => maçã [2] => kiwi [3] => limão)
```

- Se incluirmos o código dentro de um elemento html <pre></pre>, podemos visualizar a estrutura do array de forma ainda mais clara, com quebras de linha e indentação.

Verificando se uma variável está setada

- Podemos realizar esta verificação através da função isset.
- A mesma retorna true (1) se a variável estiver setada (ainda que com uma string vazia ou o valor zero), e false (0) caso contrário.
- Ex:

```
if(isset($var)) {
    echo "A variável existe";
}
else {
    echo "A variável não existe";
}
```

Verificando se uma variável está vazia

- Podemos verificar através da função empty. Retorna true se a variável não contiver um valor (não estiver setada) ou possuir valor 0 (zero) ou uma string vazia. Caso contrário, retorna false.
- Ex:

```
if(empty($var)){
  echo "A variável está vazia";
}
```

Destruindo uma variável

- É possível desalocar uma variável através da função unset.
- A função destrói a variável, ou seja, libera a memória ocupada por ela, fazendo com que ela deixe de existir.
- Se a operação for bem sucedida, retorna true.
- Ex:

```
unset($var);

unset($frutas[0]); // desaloca um elemento do array
```

Variáveis de Ambiente

- As variáveis de ambiente são mecanismos universais, presentes em praticamente todas as implementações de sistemas operacionais.
- Elas correspondem a regiões reservadas da memória principal do computador, acessíveis pelas aplicações através de primitivas disponíveis em praticamente qualquer linguagem de programação.
- Através dessas variáveis pode-se descobrir diversas informações úteis, como o navegador que o usuário está utilizando, seu endereço IP, etc.
- Para saber quais as variáveis de ambiente disponíveis, execute o código:

```
<?
echo phpinfo();
?>
```

- Na seção **Environment**, essas variáveis são listadas.

Variáveis de Ambiente

- O PHP possui a função getenv() para obter o valor das variáveis de ambiente. Por exemplo:
  - para obter o IP do cliente:

```
$ip = getenv("REMOTE_ADDR");
```

  - Para obter o navegador utilizado pelo cliente:

```
$nav = getenv("HTTP_USER_AGENT");
```

  - Para obter a porta do servidor:

```
$port = getenv("SERVER_PORT");
```

- Assim por diante, para qualquer variável listada no phpinfo().

Superglobais

- São variáveis nativas do PHP que estão disponíveis em todos os escopos:
  - $GLOBALS: referencia todas as variáveis no escopo global;
  - $_SERVER: provê informações sobre o servidor e ambiente de execução (variáveis de ambiente);
  - $_GET: variáveis submetidas pelo método HTTP GET;
  - $_POST: variáveis submetidas pelo método HTTP POST;
  - $_FILES: variáveis de arquivos submetidos por upload;
  - $_COOKIE: cookies HTTP;
  - $_SESSION: variáveis de sessão;
  - $_REQUEST: variáveis de requisição HTTP (contém, GET, POST, COOKIE).

Estruturas de Controle

- Estruturas de Decisão:
  - IF / ELSE
  - IF / ELSEIF / ELSE
  - SWITCH / CASE
  - Operador Ternário
- Estruturas de Repetição
  - WHILE
  - DO / WHILE
  - FOR
  - FOREACH
- Controle de fluxo
  - BREAK
  - CONTINUE

IF / ELSE

- A sintaxe deste comando é:

```
if (expressão lógica){
   //comandos a serem executados se a expressão for verdadeira
}
else{
   //comandos a serem executados se a expressão for falsa
}
```

- Podemos também ter somente o if, sem else.
- No PHP, as expressões lógicas devem SEMPRE estar entre parênteses. Assim, `if $i==1` não funciona, mas `if ($i==1)` funciona corretamente.

IF / ELSEIF / ELSE

- Podemos também gerar uma sequência de testes, não restrita a apenas uma expressão lógica:

```
if (expressão lógica1){
    // comandos a serem executados se a expressão lógica 1 for verdadeira
}
elseif (expressão lógica2){
    // comandos a serem executados se a expressão lógica 2 for verdadeira
}
elseif (expressão lógica3){
    // comandos a serem executados se a expressão lógica 3 for verdadeira
}
else{
    // comandos a serem executados se nenhuma expressão lógica for verdadeira
}
```

SWITCH / CASE

- O comando **Switch** aceita apenas comparações lógicas de igualdade nos testes aplicados em sua estrutura

```
switch ($i) {
    case 0:
        echo "i é igual a zero";
        break;
    case 1:
        echo "i é igual a um";
        break;
    case 2:
        echo "i é igual a dois";
        break;
    default:
        echo "i não é igual a nenhuma das opções acima";
}
```

- O comando **break** deve ser empregado em cada caso para que a estrutura de testes seja abandonada, pois se o valor pretendido já foi encontrado, então não será necessário consultar as condições seguintes.

Operador Ternário

- O operador ternário do PHP funciona como um IF simplificado, e possui a seguinte sintaxe:

```
(expressao1) ? (expressao2) : (expressao3)
```

- O interpretador PHP avalia a primeira expressão. Se ela for verdadeira, a expressão retorna o valor de expressão2. Senão, retorna o valor de expressão3.
- Exemplo:

```
<?php
$x = 54;
$x > 50 ? $y = $x*2 : $y = $x*3;
echo $y; // imprime 108 (isto é, 54 * 2)
?>
```

WHILE

- While tem o objetivo de gerar um loop (laço), que será abandonado quando a expressão lógica avaliada for falsa. A expressão lógica a ser empregada consta do início da estrutura deste comando, conforme podemos observar na sintaxe abaixo:

```
while (expressão lógica){
    // lista de comandos que fazem parte do laço
}
```

DO / WHILE

- A estrutura de comando **Do / While** testa a condição ao final da estrutura. Dessa forma, o conjunto de comandos que fazem parte do laço (loop) será executada pelo menos uma vez. Veja a sintaxe do comando abaixo:

```
do {
  // lista de comandos que fazem parte do loop
}
while (expressão);
```

FOR

- O laço **for** é empregado quando queremos gerar um loop onde sabemos o número de vezes que o mesmo deverá ser executado, ou seja, determinamos o início e o fim do laço, além da possibilidade de configurar o incremento deste valor.
- A sintaxe deste comando pode ser vista abaixo:

```
for (valor_inicial; condição_final; incremento) {
   lista de comandos a serem executados durante o loop
}
```

FOREACH

- O comando FOREACH percorre cada posição de um vetor/matriz (no exemplo abaixo, $array). A cada iteração, o valor do elemento corrente é atribuído a uma variável (no exemplo, *$valor)* e o ponteiro interno da matriz é avançado em uma posição (assim, no próxima iteração será visto o próximo elemento).

```
foreach ($array as $valor){

    // instrucoes
}
```

BREAK

- O comando break pode ser utilizado em laços de repetição, além do uso já visto no comando switch.
  - Ao encontrar um break dentro de um desses laços, o interpretador PHP interrompe imediatamente a execução do laço, seguindo normalmente o fluxo do script.

```
<?php
$x = 1;
while ($x <= 50) {
  echo $x."<br>";
  if ($x == 20){
    echo "encerrando o loop<br>";
    break;
  }
  $x++;
}
echo "loop encerrado";
?>
```

CONTINUE

- O comando **continue** também deve ser utilizado no interior de um laço de repetição e faz com que o fluxo do programa volte para o início do mesmo.

```
<?php
for ($i = 0; $i < 50; $i++) {
  if (($i % 2)== 1)
    continue;
  echo $i."<BR>";
}
?>
```

- O exemplo acima é uma maneira (embora ineficiente) de imprimir os números pares entre 0 e 49. Ao detectar um valor ímpar, o fluxo segue para a próxima iteração, ignorando os comandos abaixo de continue.

Functions

- Funções são blocos de código executados independentemente do script.
- Existem funções pré-definidas (ex: date, rand, etc), mas também é possível criar novas funções, definidas pelo programador.
- Após declaradas, as funções podem ser chamadas a qualquer momento dentro do programa.
- São úteis para deixar o código organizado e modular, além de evitar repetições de códigos toda vez que precisarmos executar a mesma tarefa.
- Podemos passar argumentos para as funções, os quais tornam-se variáveis locais da função.
- Uma função pode retornar um valor para o local em que foi chamada.
- Não é possível que uma função devolva mais do que um valor, mas é permitido fazer com que uma função devolva um valor composto, como listas ou arrays.

Functions

- A sintaxe para a criação de uma função em PHP é:

```
function nome_função ($arg_1, $arg_2,...,$arg_n){
    // comandos
    return $valor_retorno;
}
```

- **nome_função** deve ser um identificador único, e não pode iniciar com um número, nem conter vírgula, espaço, etc.

Functions (exemplo)

```
<?php
function quadrado($numero){
  $quadrado = $numero * $numero;
  return $quadrado;
}

$x = quadrado(3);
echo "O quadrado é: ".$x;
?>
```

Passagem de parâmetros por Valor

- Por padrão, a passagem de parâmetros no PHP é por valor.
- No exemplo abaixo, a variável $a é inicializada com o valor 3. Mesmo após a execução da função mais5, $a continua valendo 3, pois foi passada uma cópia da variável para a função, e não a variável original.

```php
<?php
function mais5($numero){
  $numero += 5;
}
$a = 3;
mais5($a);
echo $a; //$a continua a valer 3
?>
```

Passagem de parâmetros por Referência

- Para que um parâmetro seja passado por referência, devemos utilizar o modificador & antes do nome da variável na declaração da função.
- No exemplo abaixo, $a e $b são inicializadas com 1. Após a execução da função **mais5**, **$a** passa a valer 6, pois foi passada a referência (endereço na memória) da variável original. Já $b continua valendo 1, pois foi passada por valor.

```
<?php
function mais5(&$num1, $num2) {
   $num1 += 5;
   $num2 += 5;
}
$a = $b = 1;
mais5($a, $b);
echo $a."<br>".$b; //$a vale 6, $b ainda vale 1
?>
```

Argumentos com valores pré-definidos (default)

- Em PHP é possível ter valores default para argumentos de funções, ou seja, valores que serão assumidos no caso de nada ser passado no lugar do argumento.
- Quando algum parâmetro é declarado desta maneira, a passagem do mesmo na chamada da função torna-se opcional.

```php
<?php
function teste($texto = "Texto default") {
  echo $texto."<br>";
}
teste(); // imprime "Texto default"
teste("Texto alterado"); // imprime "Texto alterado"
?>
```

Escopo

- O **escopo** de uma variável define a porção do programa onde ela pode ser utilizada.
- Na maioria dos casos, todas as variáveis têm escopo global.
- Entretanto, em funções definidas pelo programador, um escopo local é criado.
- Uma variável de escopo global não pode ser utilizada no interior de uma função sem que haja uma declaração.

Escopo Local x Global

- No exemplo abaixo, não haverá saída alguma, pois a variável **$texto** é de escopo global, e não pode ser referida num escopo local. Acrescente a linha de código mostrada na caixa de texto e veja a diferença:

```
<?php
$texto = "Testando";

function testa_escopo(){
    global $texto;
  echo $texto;
}
testa_escopo();
?>
```

Teste com e sem esta linha

Escopo global (vetor $GLOBALS)

- Uma outra maneira de acessar variáveis globais dentro de uma função é utilizar um array pré-definido pelo PHP, cujo nome é **$GLOBALS**.
- O índice para a variável referida é o próprio nome da variável, sem o caracter $. O exemplo anterior e o abaixo produzem o mesmo resultado:

```
<?php
$texto = "Testando";
function testa_escopo(){
  echo $GLOBALS["texto"];
}
testa_escopo();
?>
```

Static

- Uma variável estática é visível num escopo local, mas é inicializada apenas uma vez e seu valor não é perdido quando a execução do script deixa esse escopo.

```
<?php
 function Teste() {
   $a = 0;
   echo $a."<br>";
   $a++;
 }
 Teste();
 Teste();
 Teste();
 ?>
```

static

- No exemplo acima, nas três chamadas à função Teste(), será impresso o valor zero. O comando $a++ na função é inútil, pois assim que for encerrada a execução da função, a variável $a perde seu valor.
- Agora inclua a palavra static no local indicado e verifique que a cada chamada à função Teste(), o valor será incrementado.

require e include

- As instruções require e include têm por objetivo inserir pedaços de código PHP no script atual. A diferença entre eles está na forma como tratam os erros gerados na carga do arquivo:

  - **require**: produzirá um erro fatal de nível E_COMPILE_ERROR e parará o script:

```php
<?php
require "function.php";
echo "Aqui o programa continua";
?>
```

Fatal error: require(): Failed opening required 'function.php' (include_path='C:\xampp\php\PEAR') in **C:\xampp\htdocs\data.php** on line **11**

  - **include**: apenas emitirá um alerta (E_WARNING), permitindo que o script continue:

```php
<?php
include "function.php";
echo "Aqui o programa continua";
?>
```

Warning: include(): Failed opening 'function.php' for inclusion (include_path='C:\xampp\php\PEAR') in **C:\xampp\htdocs\data.php** on line **11**
Aqui o programa continua

require_once e include_once

- Essas instruções possuem as mesmas funcionalidades que require e include, porém com uma diferença fundamental: o PHP verifica se o arquivo que está tentando incluir já foi incluído anteriormente. Se já tiver sido, ele não será incluído novamente.

```
<?php
include_once("includes/cabecalho.php");
?>
```

Bibliotecas de Funções

- Utilizando functions e o recurso de include/require, podemos construir nossas próprias bibliotecas de funções e incluí-las em nossas páginas PHP.

functions.php

```php
<?php

function doNiceStuff() {
  echo "fazendo algo legal<br>";
}

function doBoringStuff() {
  echo "fazendo algo chato<br>";
}

?>
```

teste.php

```php
<?php
include "functions.php";
doNiceStuff();

doBoringStuff();

?>
```