Aula 7 – Estruturas de Repetição

Objetivo: Conceituar Estruturas de Repetição

7.1. – Estruturas de Repetição

Quando programamos em PHP precisamos repetir o mesmo código para que ele seja executado várias vezes. As estruturas de repetição foram criadas para fazer este serviço para nós, sem que precisemos repetir o código. Em PHP temos quatro estruturas de repetição:

- **while** - repete o bloco de código enquanto uma condição especificada for verdadeira;
- **do...while** - repete o bloco de código uma vez, depois repete novamente enquanto a condição especificada for verdadeira;
- **for** - repete o bloco de código um determinado número de vezes;
- **foreach** - repete o bloco de código para cada elemento de um array.

7.1.1 – While

Repete o bloco de código enquanto uma condição especificada for verdadeira. O exemplo da Figura 31 define a variável $i inicialmente com o valor de 1 ($i = 1). Em seguida, o loop **while** continuará a ser executado enquanto i for menor ou igual a 5 ($i <= 5). A variável i vai aumentar em 1 a cada vez que o loop é executado ($i++;)

```
<html>
<body>

<?php
$i=1; // $i começa com o valor de 1
while($i<=5) {  // repete enquanto for menor que ou igual a 5
  echo "The number is " . $i . "<br>";
  $i++; // soma 1 ao valor de $i
}
?>

</body>
</html>
```

Figura 31 – Loop while. Fonte: Próprio Autor.

7.1.2 - Do...While

Esta estrutura repete o bloco de código uma vez e, depois, repete novamente enquanto a condição especificada for verdadeira. O exemplo da Figura 32 define uma variável $i inicialmente com o valor de 1 ($i = 1;). Então, ele começa o do...while. O ciclo irá somar a variável i com 1 e, em seguida,

escrever alguma saída. Em seguida, a condição é verificada (i é menor ou igual a 5), se for, o bloco de código é repetido. Se não for, a repetição é interrompida e a próxima linha depois do do...while é lida.

```
<html>
<body>

<?php
$i=1; // $i inicia com o valor de 1
do { // executa o bloco
  $i++;
  echo "The number is " . $i . "<br>";
} while ($i<=5); // Se $i for menor ou igual a 5, repete novamente
?>

</body>
</html>
```

Figura 32 – Estrutura do...while. Fonte: Próprio Autor

7.1.3 – For

Usada quando se sabe com antecedência o número de repetições. O exemplo da Figura 33 define um ciclo que começa com $i = 1. A repetição continuará a funcionar enquanto a variável $i for menor que ou igual a 5. A variável $i vai aumentar em 1 a cada vez que o loop é executado.

```
<html>
<body>

<?php
for ($i=1; $i<=5; $i++) {
  echo "O número é " . $i . "<br>";
}
?>

</body>
</html>
```

Figura 33 – Estrutura de repetição for. Fonte: Próprio Autor

Aula 8 – Funções em PHP

Objetivo: Conceituar e conhecer as funções referentes ao PHP.

8.1 – Funções

Funções são pequenos programas que ajudam sua aplicação em uma tarefa. Vamos supor que você tenha uma aplicação que calcula uma nota fiscal, mas, para dar o total da nota você tem que fazer outro cálculo, o do frete. Esse cálculo do frete não faz parte do cálculo da nota fiscal, porque ele pode existir ou não. Além disso, o usuário pode tentar outros lugares de entrega para baratear o frete. Podemos criar uma função que faça só esse cálculo. Usando funções, o código fica mais simples e de fácil manutenção. Podemos até utilizar funções de outras pessoas, sem nem mesmo saber como foram programadas.

Uma função precisa de um nome e de um bloco de código que será executado quando chamamos a função pelo seu nome. Coloque um nome que reflita o que função irá fazer. A sintaxe da função consiste em:

```
function nomeDaFuncao() {
bloco de código;
}
```

8.2 – Adicionando Parâmetros

Parâmetros são como variáveis que recebem um valor quando a função é executada. Normalmente esses parâmetros são necessários para a realização da tarefa. Por exemplo, uma função que soma dois números. Para poder somar, ela vai precisar dos dois números que são os parâmetros.

```
<html>
<body>

<?php
// para trabalho a função precisa receber dois valores
// o primeiro vai para $nome, o segundo para $sobrenome
function apresentacao($nome, $sobrenome) {
    echo "Oi, " . $nome . " " . $sobrenome;
}


apresentacao("Fulano", "de Tal"); // na chamada da função passamos os valores.
apresentacao("Beltrano", "de Tal"); // observe a ordem, é importante.
?>

</body>
</html>
```

Figura 34 - Passagem de parâmetros para funções. Fonte: Próprio Autor

8.3 – Retornando Valores

Podemos retornar um valor computado pela função utilizando a palavra-chave **return**. Este comando retorna o que estiver à direita e encerra a função. A Figura 35 mostra um exemplo de retorno de função. A função calcula a soma de dois números, $x e $y. Ela deve escrever uma página com "2+2=4".

```
<html>
<body>

<?php
function soma($x, $y) {
    $total=$x+$y;
    return $total; // aqui o valor de $total é retornado
}

echo "2 + 2 = " . soma(2, 2); // soma(2, 2) é substituido pelo retorno da função
?>

</body>
</html>
```

Figura 35 – Retorno de uma função. Fonte: Próprio Autor

8.4 – Formulários

Para poder receber informações do lado do cliente, ou seja, do usuário, utilizamos hiperlinks ou formulários. Recebemos informação do usuário quando ele clica em um hiperlink, chamado de método GET, ou quando nos envia um formulário, chamado de método POST.

8.4.1 – Método GET

Este método foi uma das primeiras formas de comunicação cliente/servidor via web, ele utiliza a URL do site para enviar dados. Não há necessidade de um formulário HTML, no entanto, é visível para todos. Digitar senhas utilizando esse método é inviável.

Como é realizado o envio de dados por GET:

www.meusite.com.br/index.php?nome=fulano&idade=30. Depois da interrogação são enviadas duas variáveis: nome=fulano, idade=30. O & separa uma variável da outra. Na Figura 36, a página PHP recebe os dados e escreve uma página de resposta.

```php
<html>
<body>

<?php
$usuario = $_GET["nome"]; //o parâmetro deve ser igual ao enviado.

echo "<h1>Oi! " . $usuario . "</h1>";
?>

</body>
</html>
```

Figura 36 – Página recebe os dados enviados e os processa. Fonte: Próprio Autor.

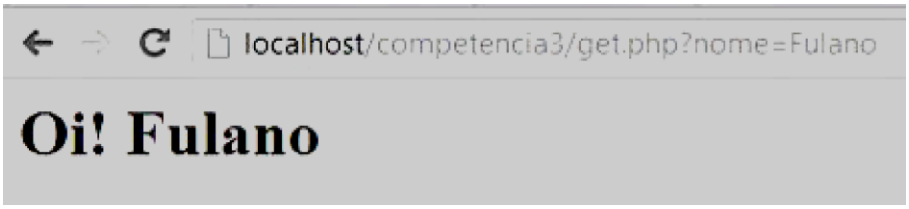

Figura 37 – O endereço da página contém o nome. Fonte: Próprio Autor.

8.4.2 – Método POST

O método POST utiliza um formulário HTML para enviar a informação. Veja no exemplo da Figura 38.

```php
<html>
<body>

<?php
$usuario = $_POST["nome"]; //o parâmetro deve ser igual ao enviado.
$tempo = $_POST["idade"];

echo "<p>Oi! " . $usuario . "<br />";
echo "Voce tem " . $tempo . " anos.</p>";
?>

</body>
</html>
```

Figura 38 – Página PHP recebe dados de um formulário HTML. Fonte: Próprio autor.

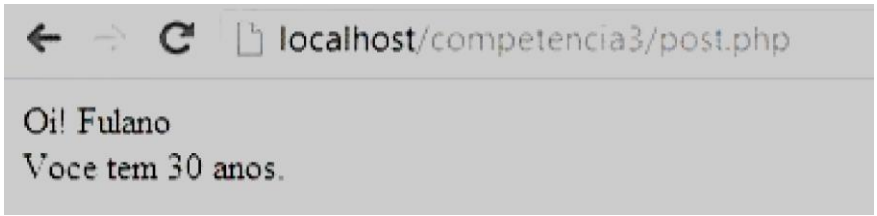

Figura 39 – Página de Saída. A URL não contém dados. Fonte: Próprio Autor.

Aula 9 - Introdução à Programação em Linguagem Orientada a Objetos

Objetivo: Conceituar Linguagem Orientada a Objetos.

9.1. - Programação Orientada a Objetos (POO)

O desenvolvedor novato percebe que muitas das ferramentas e linguagens atuais (PHP, Java, Ruby, Python, entre outras) são linguagens que utilizam o paradigma orientado a objetos**.** A princípio isso pode parecer ruim - "mais uma coisa para aprender", mas, na verdade é o que vai permitir ao desenvolvedor se aprimorar.

Sabendo POO, o desenvolvedor poderá resolver os mesmos problemas utilizando não somente o passo-a-passo tradicional, mas também uma modelagem do problema de uma forma mais natural/real.

Orientação de objetos (OO), em uma definição formal, é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas "objetos". Ou seja, é um modelo utilizado no desenvolvimento de software em que trabalhamos com unidades chamadas *objetos.*

De acordo com Santos (2003), a Programação Orientada a Objetos (POO) foi criada para tentar aproximar o mundo real e o mundo virtual, a ideia fundamental é tentar simular o mundo real dentro do computador. Para isso, nada mais natural do que utilizar objetos, afinal, nosso mundo é composto de objetos, certo?

Na Programação Orientada a Objetos, o programador (você) é responsável por moldar o mundo dos objetos e definir como os objetos devem interagir entre si.

Os objetos "conversam" uns com os outros através do envio de mensagens e o papel principal do programador é definir quais serão as mensagens que cada objeto pode receber e também qual a ação que o objeto deve realizar ao receber cada mensagem.

Isso possibilita a criação de códigos com baixo acoplamento e que podem ser facilmente reutilizados, o que são alguns dos principais motivos para se programar orientado a objetos.

Vejamos o exemplo para trocar uma lâmpada, agora usando a Programação Orientada a Objetos (POO). A primeira coisa a fazer é pensarmos em classes. Em outras palavras, coisas envolvidas no estudo de caso (trocar uma lâmpada):

- Pessoa, Lâmpada, Escada, Bocal da Lâmpada, Interruptor.

Definido isso, podemos combinar suas características e ações. Por exemplo: "uma Pessoa pega uma Lâmpada boa".

Apenas esse fato de dizer que a **“**Pessoa” “pega” “Lâmpada” “boa” já nos diz muita coisa, pois tudo isso pode ser modelado usando o paradigma de Orientação a Objetos. Em outras palavras, tudo não passa de modelar objetos que possuem ações e características.

Em nosso caso, “Pessoa” seria uma classe/objeto, “pega” seria uma ação da “Pessoa”, “Lâmpada” seria outra classe/objeto e “boa” seria o estado/característica/atributo da “Lâmpada”.

E por que estudar POO? Irei te lançar um desafio…

- Acesse o ranking da TIOBE, um dos rankings de linguagens de programação mais conceituados e confiáveis;
- Confira as linguagens que ocupam as primeiras posições do ranking;
- Agora acesse o Wikipédia e procure características sobre essa linguagem.

Eis o que em 90% dos casos você irá encontrar: “Essa linguagem de programação é baseada no paradigma.