

## APOSTILA PHP – professor José Ilton Barbosa

### Índice

1. Introdução (indicações para o estudo da lógica).....	2
2. Aula 1: Tópicos: - Instalação do XAMPP e do NotePad++.....	2
3. Iniciando nossos primeiros códigos.....	3
4. Iniciando o aprendizado sobre estruturas de controle ou tomada de decisão.....	6
5. Operadores lógicos.....	7
6. Aula 2: - Iniciando com formulários em HTML.....	11
7. Aula 3: - Introdução ao uso de funções.....	16
8. Aula 4: - Arrays e estruturas de repetição (ou laços de repetição).....	19
9. Aula 5: - Programação Orientada a Objetos.....	23
10. Aula 6: - Ainda sobre POO – classificação/categorização de objetos.....	28
11. Aula 7: - Banco de dados.....	33
12. Aula 8: - Padrão MVC.....	38
13. Aula 8.1 : iniciando nosso sistema de agência de viagens.....	39

## Introdução

Apesar de ser uma apostila para iniciantes em programação, o foco dela não está na teoria da Lógica da Programação, que no caso, considera-se o primeiro passo para iniciar a aprendizagem nesta área. Aqui partimos de imediato para a prática, justamente no intuito de ser mais prático que teórico. Já metendo a mão na massa, mas inserindo a parte teórica aos poucos. Afinal a prática sem a teoria não funciona muito bem, assim como a teoria sem a prática.

Mas se preferir iniciar bem do zero mesmo, recomendo alguns links com conteúdo mais voltado para a parte teórica da Lógica da Programação:

- <https://www.schoolofnet.com/curso/frontend/logica-de-programacao/logica-de-programacao/>
- <https://www.infoescola.com/informatica/logica-de-programacao/>
- <https://becode.com.br/melhor-forma-de-aprender-logica-de-programacao/>

**Aula 1: Tópicos:** - Instalação do XAMPP e do NotePad++ (ou editor Sublime) – para trabalharmos com código PHP; - Iniciando Lógica de Programação utilizando a linguagem PHP; - Criando o Primeiro código.

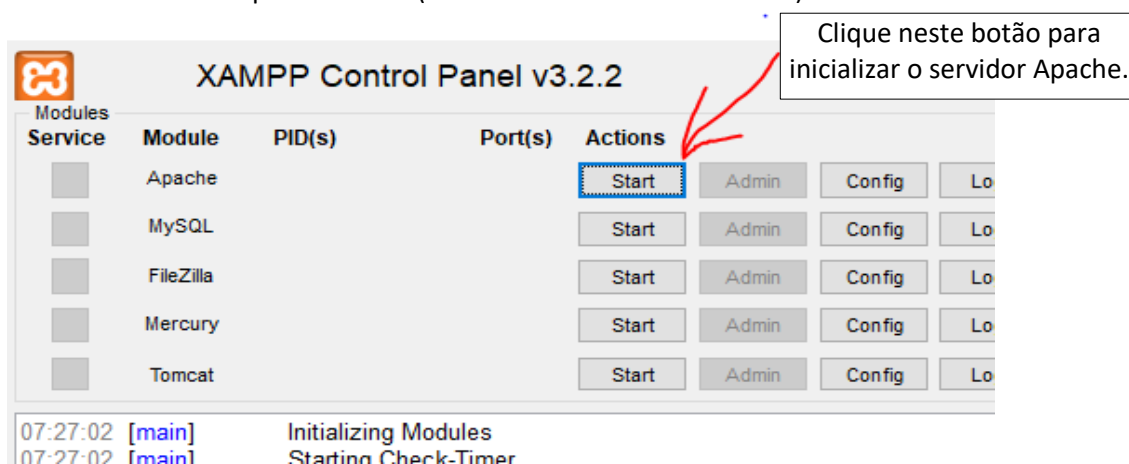
### 1 - Primeiros passos – preparar o ambiente para trabalharmos com programação PHP.

#### 1º. Passo – Instalar o XAMPP

O XAMPP é um aplicativo que faz com que o seu computador possa ter os recursos de um servidor WEB. Basicamente seu computador passa a ser um servidor WEB local, que é o recurso que possibilita ao browser de internet (Chrome ou Mozilla, Internet Explorer ou outro), exibir o resultado dos seus códigos em PHP como se estivesse navegando na internet.

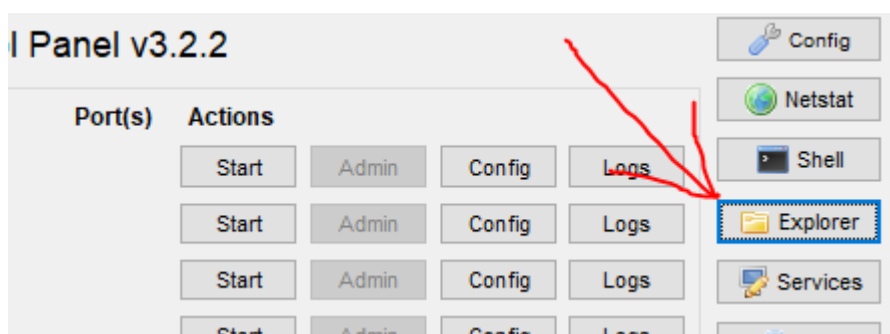
No Google digite XAMPP e já encontrará a página para baixa-lo e instala-lo.

Esta é a Tela do XAMPP após instalado (o Painel de controle do XAMPP)



#### 2º. Passo – Acessar a pasta do XAMPP e dentro dela, acessar a pasta “htdocs”.

Para acessar esta pasta, basta clicar no botão “Explorer” que está no Painel do XAMPP.



A large green arrow points down to a brown cardboard box. The box has a small cartoon character with glasses and a green shirt on it, and the text "Notepad++" is written on the side. Below the box is a green button with the word "DOWNLOAD" in white capital letters.

The screenshot shows the Notepad++ application window with the title bar 'new 1 - Notepad++ [Administrator]'. The menu bar includes 'Arquivo', 'Editar', 'Localizar', 'Visualizar', 'Formatar', 'Linguagem', 'Configurações', 'Ferramentas', 'Macro', 'Executar', and 'Plugins'. The 'Linguagem' menu is open, showing a list of languages: A, B, C, D, E, F, Gui4Cli, H, I, J, KIXtart, L, M, N, O, P, R, and S. The 'P' option is highlighted, and a sub-menu is displayed with 'Pascal', 'Perl', and 'PHP'. A blue arrow points to the 'Linguagem' menu, and another blue arrow points to 'PHP'.

```
1 <?php
2
3 $a = 5;
4 $b = 7;
5 $c = $a + $b;
6
7 echo $c;
8
```

3

## Explicando o código

**Linha 1** – Todo programa em PHP deve iniciar-se com “<?php”. Esta é a forma de indicar para o seu servidor (Xampp), que você está programando em PHP, que você está querendo executar um código em PHP.

Deixamos a **linha 2** vazia, somente para dar um espaço. Por questão de melhor estética.

**Linha 3** – Aqui estamos criando nossa primeira **variável**. **O que é uma variável?** - Variável é um recurso utilizado em programação, que possibilita armazenar temporariamente, na memória do computador, um valor qualquer, que será utilizado posteriormente. O nome desta variável em PHP, deve sempre iniciar-se com o símbolo dólar (\$). Nesse caso da linha 3, o nome que atribuímos à variável foi “\$a”. Este nome é na verdade, um nome de endereço de memória que possibilita ao computador localizar na memória dele o valor que lhe foi armazenado. Neste caso da linha 3, estamos armazenando o valor 5 no endereço de memória que chamamos de \$a. A estes endereços de memória damos o nome de “**Variável**”. Falando em linguagem mais técnica, estamos armazenando o valor 5 na variável \$a. Ou, podemos dizer ainda “estamos atribuindo o valor 5 à variável \$a”.

**Linha 4** – criamos uma variável com o nome de “\$b” e armazenamos nela o valor 7;

**Linha 5** – armazenamos na variável \$c o resultado da soma entre \$a e \$b. Logo, fica fácil deduzir que a variável \$c está armazenando o valor 12.

Deixamos a linha 6 em branco somente por uma questão estética, para ficar melhor a visualização do código.

**Linha 7** - Nosso primeiro comando PHP. O comando “**echo**”. Este comando é utilizado quando precisamos exibir algum conteúdo na tela do computador (ou seja na tela do nosso browser de internet). No caso da linha 7, temos um comando que exibe na tela o resultado da soma dos dois números. Logo será exibido o valor 12.

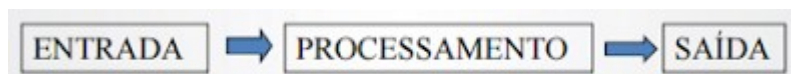
Entendeu bem até aqui? - dúvidas? - envie e-mail para [jiltonbarbosa@gmail.com](mailto:jiltonbarbosa@gmail.com)

## Um pouco de fundamentação teórica

### Entrada de dados, processamento de dados e saída de dados

Para resolver qualquer problema, seja na vida real ou no mundo virtual, é necessário seguir basicamente estes três passos: entrada de dados, processamento e saída de dados (resultado).

Para solucionar um problema, é necessário conhecer o problema (receber informações sobre o problema) – esta então é a etapa de entrada de dados; em seguida é necessário pensar em como resolver o problema (esta é a etapa de processamento dos dados); e finalmente apontar uma solução ou resolver o problema (esta é a saída de dados).



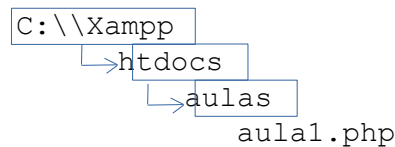
Tomando como exemplo o código que está no 2º. Passo desta apostila (página 2), já é possível identificar nele as três etapas. No código o problema a ser resolvido e o cálculo da soma de dois números (números 5 e 7). Vamos ver agora como resolver este problema.

Podemos dizer que nas linhas 3 e 4 temos as entradas de dados: o sistema está recebendo como informações sobre o problema, os valores 5 e 7. Na linha 5, temos a etapa de processamento de dados, que é a realização do cálculo dos dois números. E finalmente, na linha 7 temos a saída de dados. O programa irá exibir na tela o resultado da soma dos dois números.

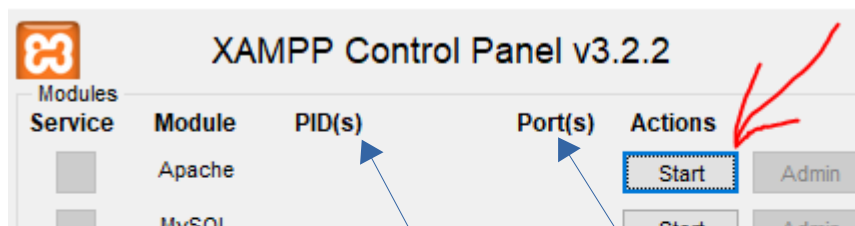
**Para estudar um pouco mais sobre este assunto, recomendo acessar o link abaixo:**

<http://aprendendolingprog.blogspot.com/2016/06/entrada-processamento-e-saida-logica.html>

**3º Passo** - Depois que você digitou o código do 2º passo, grave este arquivo dentro da pasta que você criou no 4º passo do tópico 1 desta aula. Lembrando que a pasta terá que ter sido criada dentro da pasta “htdocs” que, por sua vez, está dentro da pasta “Xampp”. A pasta “Xampp” está em “C:” (na pasta raiz do seu computador). O nome do arquivo poderá ser **aula1.php**. Lembrando que o nome do arquivo não pode conter espaços em branco e nem acentuação.



**4º Passo** – Abrir o arquivo **aula1.php** no browser de internet. - No painel do Xampp, observe se você já clicou no botão “Start” do servidor Apache.



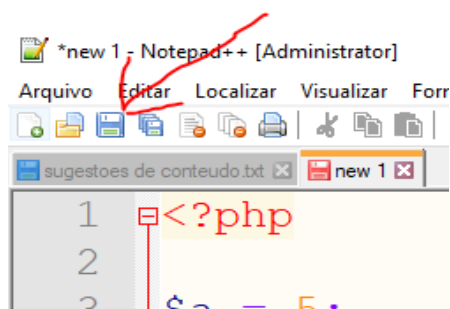
Se já tiver clicado, ele exibirá valores na coluna PID(s) e na coluna Port(s)

Agora abra o browser de internet e digite o endereço localhost/aulas/aula1.php. Observe se você utilizou alguma letra em maiúscula. Deve digitar exatamente como está o nome da pasta e do arquivo PHP.

Agora, ao precionar a tecla “Enter”, você deverá ver na tela somente o valor 12, que é o resultado da soma das duas variáveis do seu código PHP.

Se na tela foi exibida uma mensagem de erro, ou se não apareceu o número 12, é porque algo deve estar errado no seu código ou no nome do arquivo. Geralmente quando acontece algum erro de código, a mensagem de erro que é exibida na tela, indica exatamente em qual linha do código o erro ocorreu.

Para um segundo teste do seu código. Você pode voltar no código e alterar os valores das variáveis \$a e \$b, para o valor que você quiser. Grave novamente o arquivo, bastando clicar no ícone em forma de disquete que está no canto superior esquerdo da tela do NotePad++. É o terceiro ícone.



Agora volte para o browser de internet e clique em “Atualizar página” ou pressione a tecla F5 para atualizar, para poder visualizar o novo valor resultante da alteração que você fez no seu código PHP.

Podemos também alterar a operação. Ao invés de soma (+) pode utilizar o sinal de menos (-) para subtração, ou a barra (/) para divisão. Exemplo: **\$a - \$b** (subtração) ou **\$a \* \$b** (multiplicação) ou **\$a / \$b** (divisão).

### 3 – Iniciando o aprendizado sobre estruturas de controle ou tomada de decisão

Aqui você começa a ver como programar o seu sistema para que ele possa tomar decisões, a partir dos dados de entrada do seu código PHP.

Iniciamos com o comando IF. Este comando serve para o sistema executar uma ou mais linhas de comando somente se uma determinada condição for verdadeira. O comando IF testa uma condição, de modo que o resultado dos testes, será somente um dos dois valores (Verdadeiro ou Falso). Se o resultado do teste for verdadeiro, ele executa a linha de comando que estiver logo abaixo dele (no caso, a linha 8). Se não for verdadeiro, ele não executa a linha. Vejamos um exemplo:

```
1 <?php
2
3 $a = 5;
4 $b = 7;
5 $c = $a + $b;
6
7 if($c == 20)
8     echo $c;
```

Aqui acrescentamos ao nosso código anterior, o comando **IF**, conforme está na **linha 7**.

Agora se você executar este código no Browser, verá que nenhum valor será exibido na tela. Por que?

- O comando IF da linha 7, está testando se o valor da variável \$c é igual a 20. Observe que utilizamos o sinal "==" para fazer a comparação. Tem que ser desta forma.

Você já sabe que o valor da variável \$c será o valor 12, pois esta variável está armazenando o resultado da soma de 5 + 7.

Então percebe-se que 12 é diferente de 20, portanto o resultado da comparação que estamos fazendo será "Falso", portanto a linha 8 não será executada. Lembrando que a linha 8 será executada somente se o resultado da comparação for "Verdadeiro".

Agora se você alterar o valor 20 para 12, aí sim, você conseguirá exibir na tela o resultado da soma. Esta é a forma de implementarmos uma tomada de decisão no nosso código.

Vejamos um outro caso, agora utilizando os **comparadores de maior (>) ou menor (<)**.

```
1 <?php
2
3 $a = 5;
4 $b = 7;
5 $c = $a + $b;
6
7 if($c > 20)
8     echo $c;
```

Observe que agora, na linha 7, estamos utilizando o sinal ">". Isso quer dizer que o IF agora está verificando se o valor de \$c é **maior** que 20.

Se verdadeiro executa o comando da linha 8. Como sabemos que o valor de \$c é 12, então o comando da linha 8 não será executado.

- Agora altere no código o sinal ">" para "<" e veja o que acontece quando executar o programa no browser.

Ao inverter o sinal de maior para menor (<), o programa conseguirá executar o comando da linha 8, pois o IF vai verificar se \$c é menor que 20.

Nesse outro caso abaixo, utilizamos um outro comparador, o sinal de diferente (!=) ou não igual.

```
1 <?php
2
3 $a = 5;
4 $b = 7;
5 $c = $a + $b;
6
7 if($c != 20)
8     echo $c;
```

Aqui o código conseguirá exibir na tela o valor de \$c, pois o valor de \$c é diferente de 20.

Linha 7 – utilizando o comparador != (diferente ou "não igual"). Verificando se o valor de \$c é diferente de 20.

Como pode ver, existem várias formas de comparar valores. Além destes, ainda temos outros comparadores que veremos mais adiante, quando estivermos em um nível mais avançado de estudos.

A seguir, apresento uma tabela com todos eles, que retirei de um site, somente a nível de conhecimento.

Operadores de comparação, como o próprio nome já diz, compara dois valores retornando como resultado, o valor verdadeiro (TRUE) ou falso (FALSE).

Veja uma tabela com os operadores de comparação.

Operador	Nome	Exemplo	Resultado
==	Igual	<code>\$a == \$b</code>	Verdadeiro se <code>\$a</code> for igual a <code>\$b</code>
!=	Diferente	<code>\$a != \$b</code>	Verdadeiro se <code>\$a</code> <b>não</b> for igual a <code>\$b</code>
<>	Diferente	<code>\$a &lt;&gt; \$b</code>	Verdadeiro se <code>\$a</code> <b>não</b> for igual a <code>\$b</code>
===	Idêntico	<code>\$a === \$b</code>	Verdadeiro se <code>\$a</code> for igual a <code>\$b</code> e for do mesmo <b>tipo</b>
!==	Não idêntico	<code>\$a !== \$b</code>	Verdadeiro se <code>\$a</code> <b>não</b> for igual a <code>\$b</code> , ou eles não são do mesmo <b>tipo</b>
<	Menor que	<code>\$a &lt; \$b</code>	Verdadeiro se <code>\$a</code> for menor que <code>\$b</code>
>	Maior que	<code>\$a &gt; \$b</code>	Verdadeiro se <code>\$a</code> for maior que <code>\$b</code>
<=	Menor ou igual	<code>\$a &lt;= \$b</code>	Verdadeiro se <code>\$a</code> for menor ou igual a <code>\$b</code> .
>=	Maior ou igual	<code>\$a &gt;= \$b</code>	Verdadeiro se <code>\$a</code> for maior ou igual a <code>\$b</code> .

Fonte: <http://aprenderphp.com.br/artigo/operadores-de-comparacao-operadores-logicos-e-a-precedencia-dos-operadores-no-php>

Estes comparadores também podem ser utilizados para comparar textos, ou seja, não são utilizados somente com números. Veja os exemplos abaixo:

```
1 <?php
2
3 $a = " José ";
4 $b = " Ilton ";
5
6 if($b == " Ilton ")
7     echo $a.$b;
```

Neste exemplo estamos comparando se o valor da variável \$b é igual a " Ilton ". Se o resultado da comparação for "Verdadeiro", o programa conseguira executar a linha de comando número 7, ou seja, conseguirá exibir na tela o nome "José Ilton".

Na linha 7 estamos utilizando o recurso conhecido pelo nome de "concatenação", que é o ponto entre as duas variáveis, utilizado para juntar (unir) palavras. O ponto está unindo o valor da variável \$a com o valor da variável \$b.

```
1 <?php
2
3 $a = " José ";
4 $b = " Ilton ";
5
6 if($a == $b)
7     echo $a.$b;
```

Neste outro exemplo na linha 6, o IF está verificando se o conteúdo da variável \$a é igual ao conteúdo da variável \$b. Veja que o conteúdo de \$a é " José " e o conteúdo da variável \$b é " Ilton ", logo o resultado da comparação será FALSO. Sendo falso portanto, a linha de comando 7 não será executada, o que quer dizer que o programa não conseguirá exibir o nome "José Ilton" na tela.

Ainda sobre o operador de comparação IF, temos mais dois conteúdos para tratar. O primeiro é sobre o **uso de chaves** para delimitar as linhas de comando que deverão ser controladas pelo IF. Veja o exemplo abaixo:

```
1 <?php
2
3 $a = " José ";
4 $b = " Ilton ";
5
6 if($a == $b){
7     echo "Seu nome é: ";
8     echo "<BR>";
9     echo $a.$b;
10 }
```

**Linha 6** – Agora temos a **abertura da chave na linha 6 e o fechamento dela na linha 10**. Se entre as chaves tivéssemos somente uma única linha de comando, não seria necessário utilizar as chaves. Se não tivesse as linhas 8 e 9 por exemplo.

**Linhas 7 a 9** – linhas de comando que serão controladas pelo IF. Serão executadas somente se a condição for verdadeira, ou seja, somente se \$a for igual a \$b. Podemos ver que não são iguais, portanto as linhas de 7 a 9 não serão executadas neste caso.

Podemos ter quantas linhas forem necessárias entre estas chaves.

Atenção também à **identação** – espaçamento que as linhas 7 a 9 estão da margem esquerda. Esta disposição é somente por uma questão estética, muito utilizada por programadores profissionais. Melhora a visualização do código, de modo que facilita localizar possíveis erros no código.

Agora troque o sinal "==" por "!=" e faça o teste para ver se exibirá o conteúdo na tela.

O segundo assunto é sobre a possibilidade de se fazer mais de uma comparação em um mesmo IF. Aqui entramos no uso dos operadores lógicos: AND e OR, muito utilizados em programação.

### 3.1 – Operador lógico AND

Nos exemplos que vimos até este ponto, o comparador IF foi utilizado para fazer somente uma comparação entre duas variáveis (no caso comparações entre a variável \$a e a variável \$b). Agora veremos exemplos utilizando uma ou mais comparações. Veja o exemplo abaixo:

```
1 <?php
2
3 $a = " José ";
4 $b = " Ilton ";
5
6 if($a != $b && $a == "Teste"){
7     echo "Seu nome é: ";
8     echo "<BR>";
9     echo $a.$b;
10 }
```

**Linha 6** – Agora estamos fazendo duas comparações. Estamos verificando se \$a é diferente de \$b E se \$a é igual a "Teste".

O símbolo "&&" é o **operador lógico AND**. O operador AND é utilizado quando precisamos fazer duas ou mais comparações e quando é necessário que as duas comparações sejam VERDADEIRAS. Ou seja, as linhas de 7 a 9 somente serão executadas se as duas comparações

forem verdadeiras.

No caso deste código, veja que a primeira comparação (\$a != \$b) é uma comparação que resultará em VERDADEIRA e que o resultado da segunda comparação (\$a == "Teste") é FALSO. Então temos um resultado Verdadeiro e um resultado Falso na linha 6. Nesse caso, as linhas de 7 a 9 não serão executadas, pois não temos duas condições verdadeiras.

Agora veja estes casos em que **todas as condições são verdadeiras**, utilizando o AND:

```
1 <?php
2
3 $a = 3;
4 $b = 5;
5
6 if($a != 0 && $b == 5)
7     echo $a + $b;
```

```
1 <?php
2
3 $a = 3;
4 $b = 5;
5
6 if($a < $b && $b != 0)
7     echo $a + $b;
```

```
1 <?php
2
3 $a = " José ";
4 $b = " Ilton ";
5
6 if($a != $b && $b == " Ilton "){
7     echo "Seu nome é: ";
8     echo "<BR>";
9     echo $a.$b;
10 }
```

```
1 <?php
2
3 $a = " José ";
4 $b = " Ilton ";
5
6 if($a == " José " && $b == " Ilton "){
7     echo "Seu nome é: ";
8     echo "<BR>";
9     echo $a.$b;
10 }
```



### 3.1 – Operador lógico OR

O operador lógico OR é utilizado quando precisamos que somente uma das comparações seja VERDADEIRA, podendo as demais serem FALSAS.

```
1 <?php
2
3 $a = 3;
4 $b = 5;
5
6 if($a == $b || $b != 0)
7     echo $a + $b;
```

Na **linha 6**, utilizamos o operador lógico OR, representado por duas barras (pipes) (||).

Veja que a primeira comparação (\$a == \$b) é FALSA, mas a segunda comparação (\$b != 0) é VERDADEIRA, portanto a linha 7 será executada.

Lembrando que para executar a linha 7, basta que apenas uma das condições seja Verdadeira.

Podemos resumir tudo isso em uma tabela, de forma que fica melhor analisar e perceber as diferenças entre o uso do AND e do OR.

#### Uso do AND

Resultado da primeira comparação	Operador Lógico	Resultado da segunda comparação	Resultado final (a linha de comando será executada somente nos casos em que for Verdadeiro).
Verdadeiro	<b>AND (&amp;&amp;)</b>	Verdadeiro	VERDADEIRO
Verdadeiro	&&	Falso	FALSO
Falso	&&	Falso	FALSO

#### Uso do OR

Resultado da primeira comparação	Operador Lógico	Resultado da segunda comparação	Resultado final (a linha de comando será executada somente nos casos em que for Verdadeiro).
Verdadeiro	<b>OR (  )</b>	Verdadeiro	VERDADEIRO
Verdadeiro		Falso	VERDADEIRO
Falso		Falso	FALSO

## Atividades para prática da Aula 1

Dentro da pasta “htdocs” do Xampp, crie uma pasta com o nome de “aula 1”, dentro desta pasta crie os arquivos com os códigos de cada um dos exercícios em separado. O nome do arquivo do primeiro exercício deverá ser “exercicio1.php”, o próximo “exercio2.php” e assim por diante. Sem utilizar acentuação no nome do arquivo e nem espaços em branco.

1) Escreva um código em que duas variáveis recebem dois valores numéricos distintos e que uma terceira variável receberá a soma destas duas variáveis. O resultado da soma deverá ser exibido somente se o seu valor for maior que 10.

A barra é utilizada para divisão (\$a mais \$b dividido por dois)

2) Escreva um código em que duas variáveis recebem dois valores numéricos distintos e que uma terceira variável receberá como valor a seguinte fórmula  $(\$a + \$b) / 2$ . O resultado do cálculo deverá ser exibido somente se o seu valor for **maior ou igual** a 6. Se menor que 6 exibir mensagem “Reprovado”.

Veja a tabela na página 5

3) Escreva um código em que três variáveis recebem três valores numéricos distintos e que uma quarta variável receberá como valor a seguinte fórmula  $(\$a + \$b + \$c) / 3$ . Se o resultado do cálculo for maior ou igual a 6, exibir na tela a mensagem “Aprovado”. Se igual a 5, exibir mensagem “Recuperação”. Se menor que 5, mensagem “Reprovado”.

4) Escreva um código em que duas variáveis recebem dois valores numéricos distintos e que uma terceira variável receberá a soma destas duas variáveis. O resultado da soma deverá ser exibido somente se o seu valor for maior que 10 e se o valor da primeira variável for um valor positivo e diferente de zero.

5) Escreva um código em que duas variáveis recebem dois valores numéricos distintos e que uma terceira variável receberá como valor a seguinte fórmula  $(\$a * \$b) / 2$ . Se o resultado for maior que zero OU maior que 20, exibir mensagem “Valor incorreto”.

6) Escreva um código em que a variável \$idade recebe um valor numérico. Se este valor for menor que 0 ou maior que 120, exibir na tela “Valor incorreto”. Se o valor estiver entre 1 a 10, exibir na tela a mensagem “Você é uma criança”. Se valor maior ou igual a 10 E menor ou igual a 13, exibir na tela mensagem “Você é um adolescente”. Se valor maior que 13 e menor ou igual a 18, exibir na tela mensagem “Você é um jovem”. Se acima de 18 e menor que 60, exibir “Você é um adulto”, se acima de 60, “Você é um idoso”.

## Aula 2: - Iniciando com formulários em HTML para melhor aplicarmos nossos conhecimento em PHP.

Sendo o PHP uma linguagem de programação voltada para a WEB, logo se vê que é necessário algum conhecimento em HTML, que é a linguagem de marcação de texto, utilizada para criação de páginas de internet.

Por enquanto, a intenção ainda não é iniciarmos o estudo do HTML, mas somente demonstrar como criar um formulário em HTML, que possibilite o envio de dados ao PHP.

Voce pode ver até aqui, que nos exemplos anteriores estávamos atribuindo valores diretamente às nossas variáveis PHP. A partir de agora, utilizaremos os formulários para atribuir estes valores de forma dinâmica, ou seja, poderemos executar os testes utilizando valores diversos e não somente o valor que atribuímos diretamente a uma variável. A entrada de dados agora será por um formulário.

Na aula anterior falamos que um código geralmente tem três processos: entrada de dados (ou dados de entrada), processamento e saída de dados. Agora que utilizaremos formulário, nossa entrada de dados será via formulário.

Segue exemplo para melhor entendermos como funcionarão nossos próximos códigos. Inicialmente criaremos uma pasta com o nome de “**aula2**” e dentro dela criaremos um arquivo, que poderá ter o nome de **formulario1.html**, com o seguinte conteúdo:

Este é um formulário em que o usuário informará o login e a senha para acessar um determinado sistema. Os **textos em azul** são as marcações HTML, que por sua vez, são chamados de **TAG**.

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="UTF-8">
5      <title>tela de login</title>
6    </head>
7    <body>
8      <form action="processa.php" method="POST">
9        <p><label>Login:</label>
10       <input type="text" name="login"></p>
11       <p><label>Senha:</label>
12       <input type="password" name="senha"></p>
13       <p><input type="submit" value="Enviar"></p>
14     </form>
15   </body>
16 </html>
```

Da linha 1 a 7 temos a parte inicial que qualquer código HTML deve ter. Das linhas 8 a 14 temos o formulário que será utilizado como entrada de dados para o nosso código PHP.

Na linha 8 temos a TAG “form” que indica que estamos criando um formulário. Nessa TAG temos dois atributos importantes: o “action” que serve para enviar os dados do formulário para um arquivo PHP. No nosso caso, o nome do arquivo PHP é “**processa.php**”. Temos também o atributo “method”, que serve para indicar a forma como os dados serão enviados para o arquivo PHP. Nesse caso o método é o **POST**.

Nas linhas seguintes temos os campos **Login** e o campo **Senha**. Para criar campos utilizamos a TAG **<input>** e para criar a legenda dos campos, utilizamos TAG **<label>**. Observe que o nome de cada campo é especificado no parâmetro “NAME” que está dentro de cada TAG **<input>**. Na linha 10 temos o nome igual a “login” e na linha 12 temos o nome igual a “senha”. Percebeu?

Observe também o parâmetro “type”. Na linha 10, ele recebe o valor “text”, para definir que o campo login é do tipo “Texto”. Já na linha 12, o type é “password”, para indicar que é um campo de senha. E na linha 13 o type tem o valor “submit”, para definir que se trata de um botão, que você verá na tela, quando abrir o formulário no browser.

O próximo passo é, criar o arquivo **processa.php**. Este arquivo deverá ficar também na pasta “aula2”. Ele receberá os dados dos campos a serem preenchidos no formulário, no caso, o login e a senha, e em seguida fará o processamento dos dados.

Inicialmente o código PHP do arquivo “processa.php”, deverá ser o seguinte:

```
1 <?php
2
3 $login = $_POST['login'];
4 $senha = $_POST['senha'];
5
6 if($login == "teste")
7     echo "<p>Login correto</p>";
8 else
9     echo "<p>Login incorreto</p>";
10
11 if($senha == "teste")
12     echo "<p>Senha correta</p>";
13 else
14     echo "<p>Senha incorreta</p>";
15
16 if($login=="teste" && $senha=="teste")
17     echo "<p>Acesso OK</p>";
```

Pode ser que o PHP acuse erro nas variáveis das linhas 3 e 4. Por enquanto você pode ignorar estes erros. Mas caso já queira saber como resolvê-los, basta substituir o `$_POST['login'];` pelo seguinte:

`isset($_POST['login'])?$_POST['login']:null;`

Faça o mesmo para `$_POST['senha']`

Observe aqui o uso do comando **ELSE**. Se a senha informada no formulário não for a palavra “teste”, a linha 12 não será executada, mas sim a linha 14.

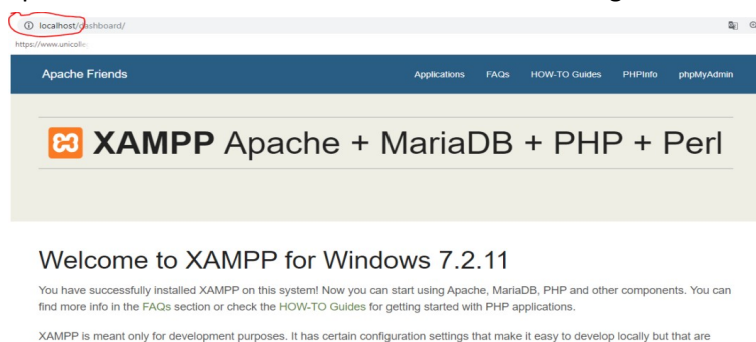
Nas **linhas 3 e 4** utilizamos o comando “`$_POST`” para capturar os dados que forem informados nos campos Login e Senha do nosso formulário. O comando para receber os dados oriundos do formulário utiliza-se do método “POST”, tendo em vista que o formulário utiliza-se deste método (method=“POST” - linha 8 do arquivo do formulário), para enviar seus dados. O login será armazenado na variável “`$login`” e a senha na variável “`$senha`”. – Todo nome de variável no PHP deve iniciar-se com o símbolo “dólar” (`$`).

**Na linha 6** estamos verificando se a variável `$login` tem o valor igual a “teste”. Ou seja, se o campo login do formulário foi preenchido com o valor “teste”. Se realmente foi preenchido com o valor “teste”, o PHP executará a linha de comando número 7, que é o comando “echo”, utilizado para exibir informações na tela do computador (no browser). Nessa linha 7, a mensagem que será exibida na tela será “Login correto”. Se não for digitada a palavra “teste” no formulário então será executada a linha 9 do nosso código PHP. Observe que na linha 8 temos o comando “**else**”, que significa “**se não**”. Ou seja, se Login igual a “teste”, execute a linha 7, senão for, execute a linha 9.

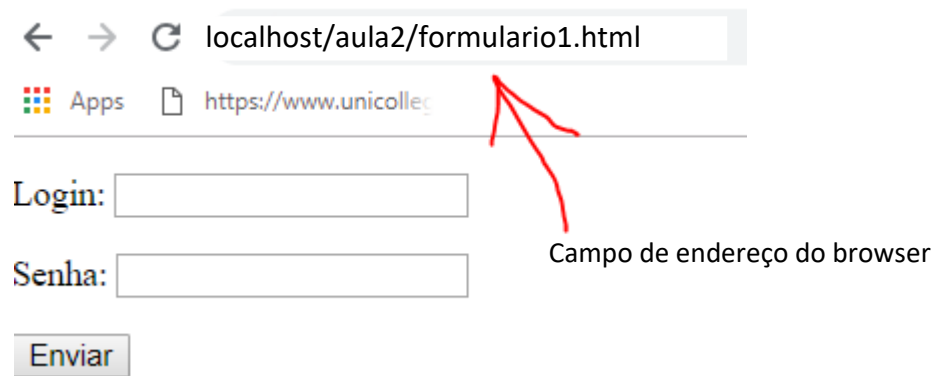
Essa mesma lógica acontece nas demais linhas do nosso código PHP. A ideia aqui é mostrar o uso básico do comando IF (linhas 11, 13 e 16).

### Agora é hora de exibir o formulário no browser.

Abra o seu browser de internet (pode ser o Chrome ou o Firefox, ou o Edge). No campo de endereço do browser, digite **localhost** e aperte a tecla ENTER. Se antes de tudo, você já tiver clicado na tecla “START” do XAMPP, a tela que deverá ser exibida no browser deverá ser a seguinte:



Agora no campo de endereço do browser digite “localhost/aula2/formulario1.html”. Lembrando que a palavra “aula2” é o nome da pasta que foi criada dentro da pasta “htdocs” e a palavra “formulario1.html” é o arquivo html que foi criado anteriormente. O browser deverá exibir o formulário na tela, da seguinte forma:



← → ↻ localhost/aula2/formulario1.html

Apps https://www.unicolleg

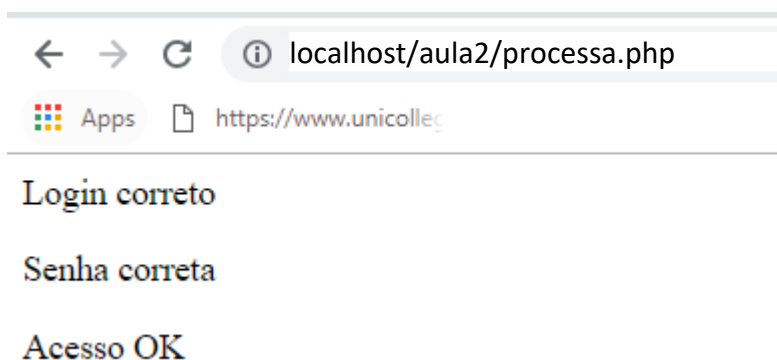
Login:

Senha:

Enviar

Campo de endereço do browser

Digite “teste” no campo login e no campo senha. Em seguida clique no botão “Enviar”. A tela a ser exibida deverá ser a seguinte:



← → ↻ ⓘ localhost/aula2/processa.php

Apps https://www.unicolleg

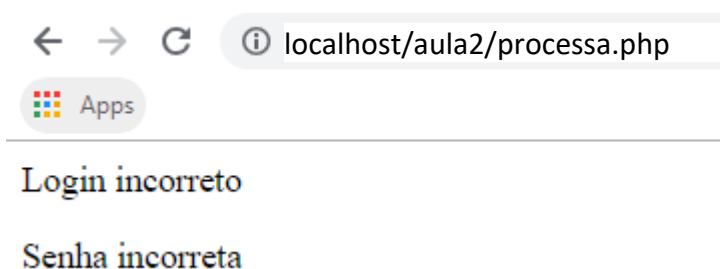
Login correto

Senha correta

Acesso OK

Observe que no local onde você digitou o endereço “localhost/formulario.html”, agora está aparecendo o nome do arquivo “processa.php”. E que na tela foi exibida as mensagens “Login correto”, “Senha correta” e “Acesso OK”, que são as mensagens que colocamos nas linhas 7, 12 e 17 do nosso código PHP.

Caso digitássemos qualquer outra coisa nos campos login e senha, que seja diferente da palavra “teste” e clicássemos no botão “Enviar”, a tela seguinte seria a seguinte:



← → ↻ ⓘ localhost/aula2/processa.php

Apps

Login incorreto

Senha incorreta

São justamente as mensagens que colocamos nas linhas 9 e 14 do nosso código PHP.

OK? Entendeu tudo? Se ficou com dúvida envia sua pergunta para [jiltonbarbosa@gmail.com](mailto:jiltonbarbosa@gmail.com)

Estude sobre HTML, fazendo consultas no Google. Por exemplo os sites:

<https://tableless.com.br/o-que-html-basico/> ou <https://www.devmedia.com.br/html-basico-codigos-html/16596>

A lógica de programação que implementamos no código PHP do arquivo “processa.php”, é somente um exemplo. Existem várias outras formas de implementar a mesma lógica. Ou seja, criar um código diferente que faça a mesma coisa que este código que já criamos. Segue abaixo uma modificação no código, mas que ainda mantém a mesma lógica e o mesmo funcionamento que o código atual.

```
1 <?php
2
3 $login = $_POST['login'];
4 $senha = $_POST['senha'];
5 $msg = null;
6
7 if($login == "teste")
8     $msg= "<p>Login correto</p>";
9 else
10     $msg="<p>Login incorreto</p>";
11
12 if($senha == "teste")
13     $msg.= "<p>Senha correta</p>";
14 else
15     $msg.= "<p>Senha incorreta</p>";
16
17 if($login=="teste" && $senha=="teste")
18     $msg="<p>Acesso OK</p>";
19
20 echo $msg;
```

Observe que não estamos utilizando as chaves nos IFs e nem no ELSE. Isso porque Cada IF e ELSE tem somente uma única linha de comando após eles.

Observe que o comando “echo” que tínhamos em cada comando “IF”, foi substituído por uma variável que chamamos de \$msg.

Na linha 5 a variável \$msg inicia-se com valor nulo (null). Na linha 8, se for digitada a palavra “teste” no campo “login” do formulário, a variável \$msg receberá como valor a frase “**Login correto**”.

Na linha 13, se o for digitada a palavra “teste” no campo “senha” do nosso formulário, a variável \$msg receberá o valor “senha incorreta”. Mas observe que antes do sinal de igual ( = ) da linha 13, colocamos um ponto ( . ). Este ponto serve para indicar que a variável \$msg vai acrescentar mais um valor a ela. Ou seja além de já ter armazenado a frase “Login correto” ela armazenará também a frase “Senha correta”. As duas frases estão na mesma variável.

As TAGs “<p>” e “</p>” são TAGS de HTML que estamos utilizando aqui para quebrar as linhas no momento em que as mensagens forem exibidas na tela.

O comando “echo” ficou somente na linha 20. Ele exibirá o conteúdo armazenado na variável \$msg.

Ainda podemos implementar essa mesma lógica de várias outras formas. Uma outra seria a seguinte:

```
1 <?php
2
3 $login = $_POST['login'];
4 $senha = $_POST['senha'];
5
6 if($login == "teste" && $senha == "teste")
7     echo "<p>Acesso OK</p>";
8 else
9     echo "<p>Login ou Senha incorretos</p>";
```

## Atividades para prática da Aula 2

Utilizando-se dessa mesma lógica e da mesma estrutura do formulário, desenvolva um outro formulário e um outro código PHP para o seguinte caso:

1 - Criar um formulário com os campos: Nome, idade, sexo (M – masculino, F – feminino). Se idade menor que 10 anos e sexo feminino, exibir mensagem “Você é criança feminina”. Se sexo masculino, “Você é criança do sexo masculino”. Se idade maior que 10 e menor que 14 – adolescente (verificando se feminino ou masculino). Se idade entre 15 e 18 – Jovem. Se entre 18 e 50 – Adulto. Acima de 50 – Idoso.

**Dica:** se no formulário da atividade anterior, você já tem dois campos “login” e “senha” como exemplos, então crie um novo formulário e os novos campos baseando-se neste que já criamos. Cada campo é criado no html utilizando a TAG <input>, então para criar o campo “nome” crie uma Tag “input” da seguinte forma:

```
<input type="text" name="nome"> - Veja que o nome do campo fica no parâmetro “name”
```

Já para o campo Sexo, crie a tag <input type="text" name="sexo">

Não esqueça de criar também as tags <label> para cada campo, conforme está no formulário anterior.

2 - O segredo para a aprendizagem é exercitar-se o quanto for necessário, até que se sinta seguro daquilo que você estiver fazendo. Por isso, recomendo que faça uma correção nos códigos do exercício da aula 1, de modo que eles passem a utilizar formulários, para atribuir valores às variáveis.

Conseguiu desenvolver o exercício 1 ?

Se sim, agora você pode substituir no formulário, o campo Sexo pelo seguinte código:

```
<p><label>Sexo:</label>  
  <input type="radio" name="sexo" value="M"> Masculino<br>  
  <input type="radio" name="sexo" value="F"> Feminino</p>
```

Veja aqui, que o parâmetro “type” não é mais do tipo “text”, mas sim do tipo “radio”. O parâmetro “name” tem o nome “sexo” nos dois campos, no entanto o parâmetro “value” tem valores diferentes (M e F).



### Aula 3: - Introdução ao uso de **funções**.

O uso de funções é um recurso muito utilizado na programação. Este recurso possibilita uma série de vantagens para o seu código, dentre elas: deixa o seu código mais organizado e evita ter vários comandos de código iguais sendo repetidos no seu código, por exemplo. - Vamos supor que em um código bem extenso que você criou, seja necessário utilizar uma mesma fórmula matemática em várias partes desse mesmo código. Nesse caso você terá que repetir uma esta fórmula matemática por várias vezes. - Utilizando-se de funções é possível que a fórmula matemática seja criada somente uma única vez e utilizada por várias vezes em vários trechos do mesmo código. Entendeu? Veja o exemplo abaixo:

```
1 <?php
2
3 function calcula () {
4     echo 5 + 7;
5 }
```

Para criarmos uma função é necessário utilizar o comando “**function**” (linha 3). Logo após o comando “function” tem que colocar o nome que daremos para a função, que, no caso é o nome “**calcula**”. Poderia ser qualquer outro nome que você quiser.

Após o nome da função, abrimos e fechamos os parênteses. Mais à frente falaremos mais sobre para que servem estes parênteses.

Fique sempre atento ao **uso das chaves**. Na linha 3 abrimos a chave e na linha 5 fechamos a chave. Estas chaves **são obrigatórias** e servem para indicar onde inicia e onde termina o código da função “calcula”.

**Outro detalhe:** - Se você executar este código no browser, verá que nada vai aparecer na tela. Por que nada aparece na tela? - Simplesmente porque você criou uma função, **mas não chamou a função**.

```
1 <?php
2 calcula();
3
4 function calcula () {
5     echo 5 + 7;
6 }
```

**Agora sim!** - veja que na linha 2 estamos chamando a função. Para chamar a função é preciso colocar o nome da função e os parênteses.

Agora vai aparecer na tela o valor 12.

Agora veremos um outro exemplo de função um pouco mais complexa:

```
1 <?php
2
3 $numero1 = $_POST['numero1'];
4 $numero2 = $_POST['numero2'];
5
6 calcula($numero1,$numero2);
7
8 function calcula ($numero1,$numero2) {
9     echo $numero1+$numero2 / 3 + 5.4;
10 }
```

As linhas 3 e 4 você já conhece. Veja que é necessário criar o formulário, da forma como já aprendeu nas aulas anteriores. O formulário deverá ter dois campos: numero1 e numero2.

Na linha 6 estamos chamando uma função. No caso deste exemplo, o nome da função é “**calcula**”. A função calcula está na linha 8.

Lembre-se da explicação que está no início desta apostila, falando sobre os três passos para se resolver um problema. Quais são eles? - Entrada, Processamento e Saída.

As funções que criamos em PHP, também utilizam-se destas três fases. A entrada de dados, por exemplo, são as variáveis \$numero1 e \$numero2. Veja que na função “calcula”, estas duas variáveis estão entre os parênteses da função. Toda **entrada de dados** de uma função deve estar entre estes parênteses.



O **processamento** da função é o cálculo que ela está executando (linha 9). E a saída é o comando “**echo**” (linha 9). Novamente lembrando, fique atento ao **Uso das Chaves**. Veja que a chave abre na linha 8 e fecha na linha 10, indicando o início e o fim do código da função.

Em uma função também temos o uso do comando “**RETURN**”. Este comando serve para a saída de dados de uma função. A princípio pode lhe parecer que o “**RETURN**” não seja muito útil, mas você verá mais à frente o quanto ele é importante.

Agora veja abaixo como fica o nosso código quando utilizamos o comando **RETURN** na nossa função.

```
1 <?php
2 echo calcula();
3
4 function calcula () {
5     return 5 + 7;
6 }
```

Veja que o comando “**echo**” agora está na **linha 2**. Antes estava dentro da função.

Na **linha 5** temos o uso do comando “**return**”. Ele serve para jogar para fora da função o resultado do cálculo. Este então é o comando de saída de dados da função.

Para fixar ainda mais o conhecimento sobre o uso de funções, segue mais um exemplo:

```
1 <?php
2
3 $numero1 = recebeValor($_POST['numero1']);
4 $numero2 = recebeValor($_POST['numero2']);
5
6 echo "<p>".$numero1."</p>";
7 echo "<p>".$numero2."</p>";
8
9 if(is_numeric($numero1) && is_numeric($numero2))
10     calcula($numero1,$numero2);
11
12 function recebeValor($num) {
13     if(!isset($num))
14         return "O valor do campo número está vazio";
15     else
16         return $num;
17 }
18
19
20 function calcula ($num1,$num2){
21     echo $num1+$num2 / 3 + 5.4;
22 }
```

**Linhas 3 e 4** – Estamos chamando uma função que tem o nome de “**recebeValor**”. Esta função foi criada para receber os valores que estão vindo do formulário. A função está na linha 12. Observe que ela está sendo utilizada duas vezes: a primeira vez na linha 3 e a segunda vez na linha 4. Uma mesma função sendo utilizada duas vezes. - É para isso que servem as funções, para serem utilizadas por várias vezes.

Perceba que a função “**recebeValor**” tem um **IF**. Este **IF** tem o comando “**isset()**”. Este comando serve para verificar se a variável \$numero1 ou \$numero2, tem algum valor armazenado nela. Ou seja, se algum valor foi digitado nos campos do formulário exibido no browser. Isso quer dizer que, se você abrir o formulário no browser e clicar no botão “**Enviar**” sem informar nenhum valor, nos campos “**número 1**” e “**número 2**”, o seu código PHP não vai conseguir executar o cálculo e exibirá uma mensagem de erro na tela.

**Linhas 13 a 16** – O código verifica se as variáveis numero1 e numero2 não estão em branco (vazias). Se uma das duas estiver vazia, ele executa a linha 14 – exibe na tela a mensagem “**O valor do campo número está vazio**”. Se não estiverem vazias (**ELSE**), ele executa a linha 16, que é o comando **RETURN**.

O comando RETURN, faz com que as variáveis \$numero1 e \$numero2 recebam os valores que foram digitados no formulário.

**Linha 9** – Aqui temos um comando novo: o comando **is\_numeric**. Ele serve para verificamos se o valor da variável é um número ou um texto. Se for um texto, ele não vai conseguir executar o cálculo.

**Na linha 13**, utilizamos a variável \$num, que serve para atender tanto a variável \$numero1 quanto a variável \$numero2. Como isso ocorre? Veja a explicação a seguir.

## ESCOPO DA FUNÇÃO

A princípio, entenda Escopo da Função, como sendo a área reservada para atuação somente dos comandos da função. Veja o exemplo abaixo:

```
1 <?php
2 $numero = 10;
3
4 exibeNumero();
5
6 function exibeNumero() {
7     echo $numero;
8 }
```

Voce acha que o número 10 será exibido na tela?

Se não exibir, por que não exibiu?

Quando falamos em “Escopo da Função”, estamos dizendo basicamente que a variável \$numero que está dentro da função não tem nada a ver com a variável \$numero que está na linha 2. São duas variáveis com o mesmo nome, só que uma está dentro da função e a outra está fora.

Neste código, para conseguirmos exibir o número 10 na tela, a função “exibeNumero” precisará ter um parâmetro, ou seja, uma variável entre os parênteses (linha 6). Por sua vez, a variável \$numero precisará estar entre os parênteses da linha 4. OK?

Então agora veja este código:

```
1 <?php
2 $numero = 10;
3
4 exibeNumero($numero);
5
6 function exibeNumero($n) {
7     echo $n;
8 }
```

A variável \$numero agora está entre os parênteses da linha 4, mas veja que na função que está na linha 6, não estamos utilizando a variável \$numero. Estamos utilizando a variável \$n.

Este código também exibe o valor 10 na tela.

A variável \$n na linha 6, recebeu o valor da variável \$numero, que foi passado pra ela através da linha 4. Entendeu?

Então, a variável que está dentro da função não tem nada a ver com a variável que está fora da função.

**Se não entendeu, pode me perguntar. Assim poderemos melhorar o texto desta apostila.**

## Atividades para prática da Aula 3

1 – Crie uma função que exiba na tela a frase “Estou utilizando uma função”.

2 – Com base no formulário de Login e Senha, crie uma função que verifica se o login e senha são verdadeiros.

3 – Crie uma função que verifica se uma pessoa é jovem ou adulta, a partir da sua idade informada em um formulário.

4 – Crie um formulário que tenha três campos: nota 1, nota 2 e nota 3. O código PHP deverá verificar se os campos foram preenchidos com valores numéricos. Se não forem número, exibir na tela uma mensagem. Crie uma função executa a soma das três notas e divide o resultado por 3, da seguinte forma:  $\$media = (\$nota1 + \$nota2 + \$nota3) / 3$ . Se o resultado do cálculo for um valor abaixo de 5, exibir na tela a mensagem “Aluno reprovado”. Se valor maior que 5 e menor que 7, exibir na tela mensagem “Aluno em recuperação”. Se valor acima de 7, exibir mensagem “Aluno aprovado”.

## Aula 4: - Arrays e estruturas de repetição (ou laços de repetição).

Se você entendeu bem para que serve uma variável, então entenderá também o que é um array. Uma variável serve para armazenar somente um valor, temporariamente na memória do computador. Já um array é um tipo de variável que consegue armazenar uma **lista de valores**, não somente um valor. Veja o exemplo abaixo:

```
$valores = array(10, 20, 30, 40);
```

A variável **\$valores** armazena quatro valores, portanto não é exatamente uma variável, mas sim um array.

Para criar um array utilizamos o termo **array**.

Agora como fazemos para exibir os valores de um array na tela?

Se o comando for somente **echo \$valores**, teremos um erro de PHP, pois agora temos que exibir vários valores. Então teremos que utilizar de um recurso do array, para conseguir imprimir cada um dos valores.

Por ser uma lista de valores no array, teremos então um endereço para cada valor, ficando então da seguinte forma: - Para exibir na tela o primeiro valor do array, o comando será **echo \$valores[0]**. Este valor zero entre colchetes indica a posição em que está o primeiro valor da lista de valores do array. Nesse caso, será exibido na tela o valor 10. Sendo assim, para exibir o valor 20, que está na segunda posição do array, o comando será **echo \$valores[1]** e assim por diante.

Vejamos este outro exemplo, em que estamos utilizando nomes de pessoas, ao invés de números. Para utilizar nomes, precisamos colocá-los entre aspas simples.

```
$nomes = array('jose francisco', 'maria sousa', 'antonio santos', 'carlos');
```

Para exibir todos os nomes da lista na tela, teríamos que fazer o seguinte:

```
echo $nomes[0];
```

```
echo $nomes[1];
```

```
echo $nomes[2];
```

```
echo $nomes[3];
```

Digite este código, salve num arquivo e teste no seu browser.

São 4 nomes, mas a contagem inicia-se pelo zero, portanto o comando **echo** vai até a posição 3.

Mas se a lista tivesse 50 nomes, por exemplo, teríamos que escrever o comando **echo** 50 vezes? Não seria bom não é? E para isso que temos as estruturas de repetição, que veremos a seguir.

Veja o código abaixo:

```
1 <?php
2 $valores = array(1, 2, 3, 4);
3 foreach ($valores as $a) {
4     echo $a."<BR>";
5 }
```

Perceba o uso das chaves.

Digite este código, salve num arquivo e teste no seu browser.

Na **linha 2** temos um array com 4 elementos. Na **linha 3**, temos o recurso que possibilita exibir na tela todos os elementos do array, sem precisar ficar repetindo o comando **echo**. Este recurso é o **"foreach"**.

O Foreach utiliza-se de uma variável a mais de apoio, para conseguir exibir todos os elementos do array. Neste caso acima, ele está utilizando uma variável que resolveu chamar de **\$a**. O Foreach repetirá o comando "echo" por quatro vezes. Cada vez que ele repetir o comando echo, a variável \$a, pega um dos elementos do array e exibe na tela a partir do comando echo da linha 4. O "<br>" é a TAG html para quebrar de linha (mudar de linha).

Temos também outros recursos de repetição, além do Foreach. Veremos agora o uso do **While**.

Veja agora o mesmo código que vimos antes, só que agora utilizando o **While** ao invés do Foreach.

```
1 <?php
2 $valores = array(1, 2, 3, 4);
3 $i=0;
4 while ($i < count($valores)) {
5     echo $valores[$i]."<BR>";
6     $i=$i+1;
7 }
```

Digite este código, salve num arquivo e teste no seu browser.

O While, utiliza-se também de uma variável de apoio, que nesse caso é a **\$i**. Esta variável chamamos de índice. Na linha 3 ela recebe o valor zero, que é a primeira posição do array.

Na linha 4, temos o comando **while**. Traduzindo para português, a palavra While significa “**Enquanto**”. Podemos dizer então que, na linha 4 temos o seguinte comando: “**Enquanto o valor da variável \$i for menor que o tamanho do array, execute as linhas 5 e 6**”.

Mas como sabemos o tamanho do array? - Para isso temos ainda na linha 4, o comando **count(\$valores)**. Este comando conta quantos elementos temos dentro do array. Se executarmos o comando “echo count(\$valores)”, será exibido na tela o valor 4. Então, “Enquanto o valor da variável \$i for menor que 4, o comando echo da linha 5 será executado.

**Linha 5** - veja que utilizamos a variável **\$i** para indicar a posição de cada elemento do array.

**Linha 6** - Cada vez que o comando While repetir as linhas 5 e 6, a variável **\$i** aumenta de valor. Veja que na linha 3 a variável **\$i** começa valendo zero, mas cada vez que o while executa a linha 6, o valor de **\$i** aumenta. A variável **\$i** recebe como valor, o valor dela mesma somando por mais 1.

Entendeu? Se não entendeu, releia este tópico. Se mesmo assim não entender, me pergunte por favor.

Além do Foreach e do While, temos ainda um outro recurso de repetição, que é o **FOR**. O mesmo código que vimos acima, ficará da seguinte forma se utilizarmos o FOR:

```
1 <?php
2 $valores = array(1, 2, 3, 4);
3
4 for ($i=0;$i < count($valores);$i++) {
5     echo $valores[$i]."<BR>";
6 }
```

Perceba o uso do **ponto e vírgula** e das chaves.

Digite este código, salve num arquivo e teste no seu browser.

**Linha 4** - Veja que também utilizamos uma variável de apoio, que no caso mantivemos a variável **\$i**. A parte que utilizamos o comando “**count**” continua igual ao que utilizamos no While. Agora a variável **\$i** está diferente, mas faz a mesma coisa que vimos no exemplo do uso do While. Inicia-se valendo zero e vai aumentando de valor à medida que o comando FOR repete o comando echo da linha 5.

**\$i++** é a mesma coisa que **\$i=\$i+1**; Digamos que é uma forma abreviada.

Agora troque os números por nomes, colocando-os entre aspas simples, e teste novamente.

Veja que são três recursos que praticamente fazem a mesma coisa, tem a mesma utilidade, mas quando você estiver programando efetivamente, desenvolvendo sistemas de verdade, verá que existem situações em que um destes recursos de repetição, será mais adequado que o outro. Dependerá da situação, do tipo de problema.

## Atividades para prática da Aula 4

1 – No exemplo abaixo, temos uma estrutura de repetição exibindo na tela os valores de um array. Neste código implementamos um IF que faz com que o terceiro elemento do array não seja exibido na tela.

```
1 <?php
2 $valores = array(1, 2, 3, 4);
3
4 for ($i=0;$i < count($valores);$i++) {
5     if($i <> 2)
6         echo $valores[$i]."<BR>";
7 }
```

Temos um IF na linha 5, que verifica a posição do elemento no array. Se a posição for diferente da posição 2, ele exibirá o valor do array na tela. Dessa forma o valor que está na posição 2 não será exibido na tela.

Qual valor não será listado na tela?

2 - Reescreva o código acima, utilizando o While, ao invés do FOR e altere o número 2 para 3, na linha 5. Verifique qual valor não foi exibido na tela agora.

3 – Reescreva o código acima utilizando o FOREACH, e colocando nomes ao invés de números no array e verifique qual nome não será exibido na tela.

4 – Crie um array com 10 elementos e exiba todos eles na tela, utilizando de um dos recursos de repetição.

5 – Crie um formulário com dois campos: o primeiro campo terá o nome “numeroinicial” e o segundo campo terá o nome “numerosfinal”. Crie um código PHP que receba estes dois números em duas variáveis diferentes. Então utilize o FOR ou o WHILE para imprimir na tela uma contagem que inicie pelo numero inicial informado no formulário e termine no número final informado no formulário.

**Dica:** Se utilizar o WHILE, a linha do while ficará da seguinte forma:

```
while($numeroinicial < $numerosfinal){
    echo $numeroinicial++;
    echo "<BR>";
}
```

### Conteúdo adicional sobre arrays

<https://www.devmedia.com.br/php-declaracao-e-atribuicao-de-arrays-em-php/38621#acesso>

## Arrays multidimensionais

É possível que um array seja utilizado como valor para outro array. Sendo esse o caso, dizemos que este é um array multidimensional. Abaixo podemos conferir como criar um **array multidimensional em PHP**:

```
01 $linguagens = array(  
02     array("PHP", "PHP: Hypertext Preprocessor"),  
03     array("SQL", "Structured Query Language")  
04 );
```

Então, para acessar os valores presentes nesse array precisamos de dois índices, visto que se trata de um array de duas posições:

```
echo $linguagens[0][1];
```

RUN

A partir do primeiro índice `0`, estamos acessando o array na primeira posição de `$linguagens`. Na sequência, acessamos a segunda posição neste array a partir do índice `1`. Ao final de sua execução, o código acima imprime o valor `PHP: Hypertext Preprocessor`.

**Agora, veja como utilizar nomes ao invés de números nos índices do array:**

```
01 $funcionarios = array(  
02     array("nome" => "Alex", "idade" => 21, "salario" => 1285.27, "ativo" => true),  
03     array("nome" => "Emerson", "idade" => 35, "salario" => 3885.27, "ativo" => false),  
04     array("nome" => "Osvaldo", "idade" => 54, "salario" => 5285.27, "ativo" => true),  
05 );  
06  
07 $bonificacao = 10;  
08  
09 foreach($funcionarios as $funcionario){  
10     if($funcionario["ativo"]){  
11         $funcionario["salario"] += $funcionario["salario"] * ($bonificacao/100);  
12  
13         echo "Funcionario: {$funcionario['nome']} - {$funcionario['salario']}\n";  
14     } else {  
15         echo "Funcionario: {$funcionario['nome']} - INATIVO\n";  
16     }  
17 }
```

RUN

**Linhas 01 a 05:** Temos o nosso array com os seguintes dados dos funcionários: nome, idade e se ele se encontra ou não ativo;

**Linha 07:** Declaramos uma variável chamada bonificação, que é utilizada para acrescentar 10% do valor do salário dos funcionários ativos;

**Linhas 09 a 17:** Para simplificar o processo de modificação desses valores, percorremos a lista de funcionários em um `foreach`, imprimindo ao final de cada iteração o resultado do cálculo, quando aplicável.

## Aula 5: - Programação Orientada a Objetos.

Nas aulas anteriores e até o momento, você criou códigos de programação de forma estruturada, onde as linhas do seu código eram executadas uma após a outra. Agora veremos uma outra forma de programar, que na verdade, é a forma mais utilizada no mercado de trabalho.

Programação Orientada a Objetos, ou simplesmente POO, é uma maneira de programar que divide os programas em pequenos blocos de código e elementos que chamaremos de Objetos. É uma maneira de programar fazendo com que os programas tenham um comportamento e características muito parecidas com a vida real.

Para saber um pouco mais sobre este conceito, leia este artigo:

<https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264>

E mais este:

<https://www.oficinadanet.com.br/post/14614-programacao-orientada-a-objetos>

Juntando seu conhecimento já adquirido sobre o uso de funções e o uso de arrays, se tiver entendido bem, vai conseguir evoluir bem em POO.

Para entender exatamente do que se trata a POO, é necessário que entenda suas quatro principais características: abstração, Encapsulamento, Herança e Polimorfismo. Aqui colocarei bem resumidamente, mas espero que já tenha lido os artigos citados acima, que explicam com mais detalhe estes quatro pilares da POO.

### 5.1. Abstração

Como já foi dito, a ideia da POO é fazer com que o programa tenha comportamentos e características muito parecidas com pessoas ou coisas do mundo real.

Já foi dito também que, trabalharemos com pequenos blocos de código separados, que chamaremos de Objetos. Estes objetos precisam ter um nome específico. Você poderá escolher o nome que quiser, mas precisa ter um nome.

O objeto então, deve parecer muito com objetos do mundo real. Temos que imaginar isso, portanto Abstração é justamente conseguir imaginar isso na sua cabeça. Um objeto que se pareça com o mundo real. E, para parecer com o mundo real, este objeto precisa ter um nome e ser útil para fazer alguma coisa de importante.

Um carro, por exemplo – ele é um objeto e tem várias funções. Basicamente serve para te levar de um lugar para outro. Mas dentro dele temos várias funcionalidades: tem freio, tem rodas, tem bancos, tem portas, etc. Cada função desta pode ser considerada como funções do carro ou objetos do carro. Então o carro é um objeto, que também tem outros objetos dentro dele.

Lembra-se do estudo sobre funções que vimos na aula 3? Pois é, são estas funções que referem-se às funções que, por exemplo, um carro pode ter.

O ser humano também pode ser considerado ou representado como um objeto na programação POO. O ser humano tem nome e tem várias funções: ele fala, ele anda, ele come... e tem vários objetos como: braço, perna, cabeça, mão...

Quando conseguimos imaginar isso, estamos utilizando da característica chamada Abstração em POO.

### 5.2. Encapsulamento

O encapsulamento é uma característica da POO, que ajuda você a programar sem se preocupar com detalhes mais complexos dos objetos. Por exemplo o ser humano. Se você olha para uma pessoa, você vê o rosto, o corpo, os



olhos, os cabelos, mas não precisa se preocupar com a parte interna da pessoa (com aquilo que você não vê), tipo os órgãos internos, intestinos, coração, fígado...

Um carro por exemplo, se você rodar a chave ele vai ligar, mas não precisa que você saiba o que acontece dentro do motor do carro para que você saiba dirigir. Basta que você saiba rodar a chave. Isso é Encapsulamento. O objeto esconde de você a parte complexa, basta que você saiba para que ele serve.

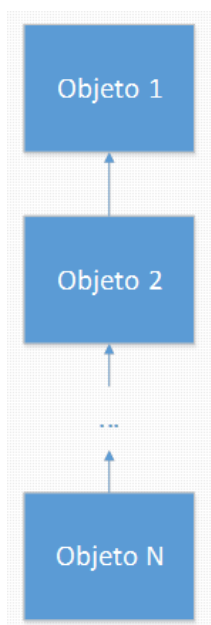
### 5.3. Herança

Lembra-se do que você aprendeu sobre funções? Lembra-se que uma mesma função pode ser utilizada em várias partes diferentes de um mesmo programa? Pois é, isso se chama reaproveitamento de código. Um mesmo código pode ser utilizado em várias partes do programa, sem que você precise ficar digitando a mesma coisa por várias vezes.

Herança é justamente a capacidade de criar características e funções que pode ser utilizadas por vários objetos diferentes, sem que seja preciso ficar criando a mesma característica para todos os objetos do seu programa.

Podemos utilizar a princípio, como exemplo, as características que um filho herda dos seus pais. Os filhos geralmente se parecem com seus pais. Nesse caso, estamos falando de Herança Direta. Mas se o filho herda características dos seus avós, daí estamos falando de Herança Indireta.

Vejamos os exemplos abaixo:



O **objeto 2**, herda **diretamente** as características do **objeto 1**. Já o **objeto N** herda diretamente características do objeto 2 e, **indiretamente** características do objeto 1.

Isso quer dizer que, por exemplo, se no **objeto 1** você criar uma função para calcular a soma de dois números, o **objeto 2** poderá utilizar esta função, sem que seja necessário você ter que recriar ou reescrever o código da função de cálculo dentro do objeto 2. O **objeto N**, por sua vez, também poderá utilizar a função de cálculo do **objeto 1**.

Isso é Herança. Depois veremos como fazer isso no código PHP.

### 5.4. Polimorfismo

Basicamente é a capacidade que um objeto tem de se modificar. Na natureza, por exemplo, temos alguns animais que se modificam para se adaptar a alguma situação específica. Veja, por exemplo o Camaleão, que muda de cor para se esconder dos seus predadores.

Na programação temos casos em que é necessário alterar a função de um objeto em um momento específico. Os casos em que precisamos alterar as funções internas de um objeto, no momento em que ele estiver sendo executado, é o que chamamos de polimorfismo.

Podemos fazer uma comparação bem básica entre pai e filho. O filho herda do pai a capacidade de comer (a funcionalidade de comer). No entanto, o pai come utilizando um garfo, mas o filho come utilizando uma colher.



Esses quatro pilares são essenciais no entendimento de qualquer linguagem orientada a objetos e da orientação a objetos como um todo. Quando você começar a praticar todas elas no código PHP, você conseguirá assimilar ainda mais estes conceitos.

## 5.5. Classes e Objetos

Começemos explicando o que é uma classe. Digamos que classe é um pedaço de código em que você consegue escrever nele, as características e funções que teremos em um objeto. Por exemplo, se você quiser criar um código que represente um carro, é necessário que você crie então um trecho de código que tenha as características e funções que um carro pode ter.

Para representar as características do carro, utilizaremos variáveis em PHP. Sendo que os nomes das variáveis devem representar as características deste carro. Por exemplo, um carro tem nome e tem várias partes dentro dele, então precisamos criar as variáveis para representar este nome e estas partes do carro. Poderia ser por exemplo, as seguintes variáveis: \$nome, \$quantidade\_de\_acentos, \$quantidade\_de\_pneus, \$quantidade\_de\_portas, \$tipo\_de\_combustivel, etc. Tudo isso são características e podem ter muitas outras características. Só que não precisa colocar no seu código todas as características possíveis que um carro pode ter. Basta colocar aquelas características que você vai precisar no seu código. Veremos mais detalhes sobre isso mais adiante.

Um carro tem funcionalidades, por exemplo, ligar o carro, frear o carro, andar para trás, andar para frente, abrir a porta, fechar a porta, ligar o farol.... Para representar estas funcionalidades, precisaremos criar as funções em PHP.

Então agora, você viu que precisamos criar em PHP, as variáveis e as funções para representar um objeto específico do mundo real.

Agora, como agrupar estas características e funções dentro de um trecho de código, de forma que ele fique separado das outras partes do código? Podemos dizer que, basta criar um arquivo separado contendo esta parte do código. Então, para cada objeto que precisamos criar, teremos que ter primeiro um arquivo separado. Ok? Entendeu até aqui?

Estes pedaços separados de código é o que chamaremos de **Classes**. Então uma classe é um pedaço de código que contem características e funções de um objeto específico. Digamos assim.

Portanto, se você quiser criar um programa em PHP que represente o funcionamento de um carro, por exemplo, é necessário que primeiro você crie a Classe que vai representar este carro. OK?

Esta classe deverá ser útil para criar qualquer tipo de carro. Serve para criar um fusca, ou uma BMW, ou um caminhão, ou um ônibus por exemplo. Ela é simplesmente uma classe para criar carros.

Até aqui você já deve ter entendido o que é uma Classe. Agora, **O que é um objeto?**

Vamos supor que você vai criar um cadastro de carros para uma agência que vende carros, por exemplo. Primeiramente, no seu código você deverá criar a classe que servirá para representar os carros da agência e, em seguida, no seu código você precisará utilizar esta classe carro, para criar o cadastro de cada um dos carros da agência. Pois bem, o Objeto seria então o cadastro de cada um dos carros. Ou seja, cada objeto precisou utilizar a classe carro para que fosse possível criar o cadastro. OK?

O mesmo seria para criar um cadastro de pessoas. Primeiro cria-se a classe Pessoa e, a partir dela você poderá criar o cadastro de várias pessoas. Cada pessoa seria um objeto da classe Pessoa.

Em fim, as Classes servem para criar os Objetos.

## Vejamos agora como criar uma classe em PHP.

```
1  <?
2  // Criando nossa classe
3  class Aluno {
4      var $nome;
5      var $turma;
6      var $idade;
7  }
8  >
```

Aqui estamos criando a classe Aluno. Ela servirá para criar os cadastros dos alunos. Cada aluno cadastrado é um objeto da classe Aluno. Então, veja que, a partir de uma classe podemos criar vários objetos.

Veja que nesta classe temos somente três características do aluno: nome, turma e idade. Mas poderíamos ter quantas características que fossem necessárias. A estas variáveis da classe, chamamos de ATRIBUTOS da classe.

Nesta classe ainda não temos as funções, mas somente as características.

**Linha 3** - utilizamos a palavra “class” para criar a classe. O nome desta classe é “**Aluno**”. Veja que após o nome da classe precisamos abrir uma chave e depois fechá-la no final do código (linha 7). Perceba também que o nome da classe inicia-se com letra maiúscula. A comunidade de programadores, combinou entre eles que, todo nome de classe deve iniciar-se com letra maiúscula. Esta é uma forma de padronizar a maneira de programar, de modo que não fique cada programador fazendo as coisas de maneira diferente dos outros programadores.

**Linhas 4 a 6** – Criamos as variáveis da classe Aluno. Veja que para cada variável foi necessário colocar a palavra “var”. Os nomes das variáveis deve sempre iniciar-se com letra minúscula.

### Como utilizar uma classe (instanciando uma classe)

Esta classe você deverá criar em um arquivo separado, e recomenda-se que este arquivo seja gravado no computador com o mesmo nome da classe. Então se a classe é Aluno, o nome do arquivo será Aluno.php.

Agora criaremos um outro arquivo que utilizará a classe Aluno. Veja abaixo:

```
1  <?
2  //Instanciando nossa classe
3  require_once("Aluno.php");
4  $manuel = new Aluno;
5  $manuel->nome = "Emanuel Gonçalves";
6  $manuel->turma = 3001;
7  $manuel->idade = 19;
8  >
```

**Linha 3** – utilizamos o recurso “**require\_once**”. Este recurso serve para pegar o arquivo “Aluno.php” e colocá-lo dentro deste novo arquivo que você acabou de criar. Somente assim você conseguirá utilizar a classe Aluno.

**Linha 4** – Aqui estamos **criando o objeto Aluno**. Veja que estamos utilizando a classe Aluno, para criar o objeto Aluno, da forma como foi explicado no início desta aula. A este recurso damos o nome de “**Instanciação**”, ou seja, estamos criando na linha 4, uma instância da classe

Aluno. Acostume-se com este novo termo. Ele será muito utilizado em POO. Para criarmos uma instância da classe Aluno, utilizamos o comando “**new**”. Na verdade o que está acontecendo aqui é o seguinte: todo o conteúdo do arquivo Aluno foi colocado dentro da variável **\$manuel**. Desta forma consigo utilizar todas as variáveis que criamos na classe aluno, como faremos nas linhas seguintes do código.

**Linhas 5 a 7** – veja que agora estamos utilizando as variáveis da classe Aluno para colocar os valores nela. Na variável **nome**, por exemplo, colocamos o nome da pessoa: “**Emanuel Gonçalves**”. Assim também colocamos a turma e a idade. A partir daí temos um **objeto do tipo Aluno**. Veja que para colocar os valores precisamos utilizar uma **seta**. A mesma variável da linha 4, foi repetida nas linhas 5 a 7. Isso porque a variável “\$manuel” possui dentro dela todo o conteúdo do arquivo “Aluno.php”.

Isto que você está vendo nas linhas 5 a 7, é um aprofundamento do conceito de Arrays que você viu em aulas anteriores. Lembra-se que utilizava-se um índice para localizar cada elemento dentro de um array? Só que agora, no lugar do índice, estamos utilizando os nomes das variáveis.

Entendeu até aqui? Se não entendeu, pergunte-me.

## Atividades para prática da Aula 5

Responda às perguntas, por escrito, como uma forma de ajudar você a fixar os conceitos vistos até aqui. São muitos detalhes para você aprender, portanto é importante praticar estes exercícios para que tudo fique muito bem entendido e fixado na sua cabeça.

1 – O que é Programação Orientada a Objetos?

2 – O que é necessário ter estudado antes para entender melhor a POO ?

3 – Quais os quatro pilares da POO ?

4 – Escreva resumidamente, em 3 ou 4 linhas o que entendeu sobre cada um dos pilares da POO.

5 – Numere a segunda coluna de acordo com a primeira.

1	Abstração		O objeto deve esconder a parte complexa, de modo a facilitar seu entendimento e seu uso.
2	Encapsulamento		As funções de um objeto podem ter a capacidade de mudar seu modo de funcionamento.
3	Herança		O objeto em POO deve ficar muito parecido com um objeto do mundo real.
4	Polimorfismo		O filho herda as características de seus pais.

6 – O que é uma classe e para que serve?

7 – O que é um objeto?

8 – Escreva um código que tenha a classe Carro com as seguintes atributos: marca, modelo, cor, ano e cidade.

9 – Escreva um código que utilize a classe Carro, com base no exemplo do código que está na página 26, só que, ao invés de \$manuel utilize a variável \$carro.

10 – Na linha seguinte do código da questão 9, escreva o código: **echo \$carro→marca; echo \$carro→modelo** e em seguida escreva também o comando **echo** para as demais variáveis do código. Execute o código no browser e veja se os dados serão exibidos na tela.

## Aula 6 - Ainda sobre POO – como classificar/categorizar os objetos que o nosso sistema precisará ter.

Se estamos falando de **Classes**, isso implica em saber **classificar** as coisas (ou os elementos) que utilizaremos no sistema que estamos planejando desenvolver. Classificar no sentido de agrupar as coisas que tem a ver umas com as outras. Por exemplo, vamos supor que você precise arrumar uma casa que está toda bagunçada, com tudo fora do lugar, - a primeira coisa a fazer é identificar os objetos que são da cozinha, os que são do quarto, os que são da sala e assim por diante. Ao conseguir identificar e colocá-los todos em seus devidos lugares, você está na verdade fazendo uma classificação, ou seja, agrupando as coisas por classes. É como se você tivesse criado as classes, quarto, cozinha, sala, banheiro, etc.

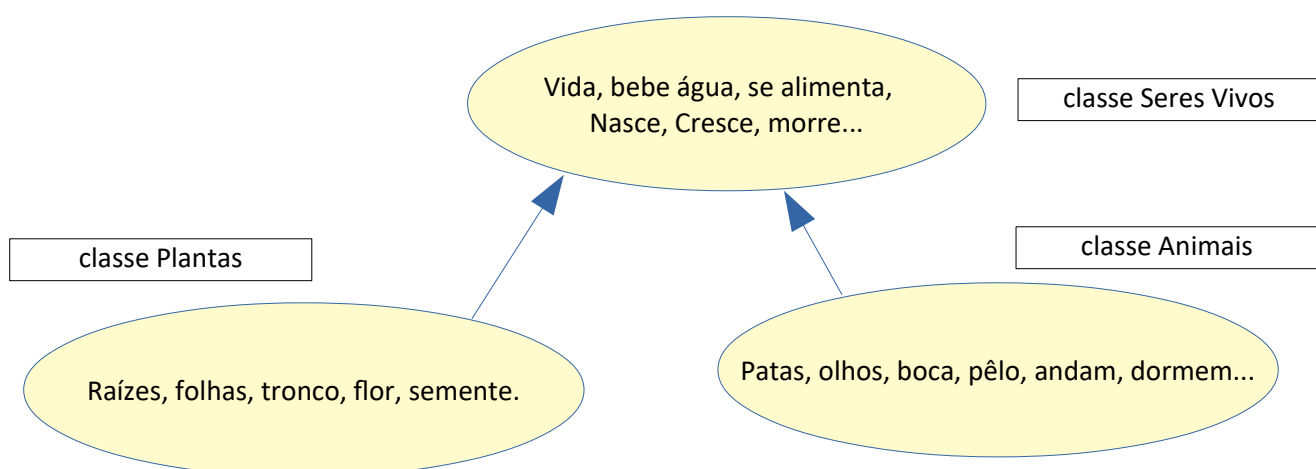
Ainda falando sobre classificação, veja que, no caso da arrumação da casa bagunçada, primeiro fazemos uma classificação bem geral. Colocamos as coisas nos cômodos corretos da casa. Mas dentro de cada cômodo, temos os armários e os locais apropriados para guardar cada um dos objetos, então é necessário fazer uma classificação mais específica dos objetos dentro de cada cômodo. Por exemplo: as panelas pertencem à cozinha, mas precisam ficar guardadas no armário de baixo, já os copos que também pertencem à cozinha ficam no armário de cima... e assim por diante.



Vamos agora falar de outros tipos de classificação, pegando como exemplo a natureza. Na natureza temos vários tipos de animais e plantas. Se entendeu o que é classificar, já saberia escrever aqui as características (atributos) que pertencem aos animais e os atributos que pertencem às plantas. Então você teria a classe com o nome Animal e outra classe com o nome de Planta. Mas veja que, além destas duas classes você poderá ter uma outra classe principal, que teria os atributos que são iguais tanto para plantas como para animais. Poderia ser a classe “Seres Vivos”.

Quais seriam os atributos que servem tanto para as plantas como para os animais? Ambos tem vida, precisam de água pra viver, precisam de alimentos, vivem no planeta terra, nascem, crescem e depois morrem.

Representando em um gráfico, ficaria da seguinte forma:



Lembra-se do estudo que fizemos sobre **Herança**? Aqui temos um exemplo de herança. Ambas as classes Plantas e Animais estão herdando as características da classe Seres Vivos.

Então, se eu lhe perguntasse quais as características da classe Plantas no desenho acima, quais você me diria? - As características da classe Planta são todas que estão dentro dela, mas também são as que estão na classe Seres Vivos, pois ela está herdando as características da classe Seres Vivos.

Tranquilo até aqui? Entendido?

Vamos a mais alguns exemplos para ajudar a fixar ainda mais este conceito de classificação.

Por exemplo, vamos considerar um cachorro como nosso “objeto” de estudo:



**Figura 1 - Cachorro**

Analisando este objeto, o cachorro, podemos deduzir que o mesmo possui algumas características que pertencem apenas a ele próprio. Por exemplo, um cachorro possui:

- *Um nome,*
- *Uma idade,*
- *Um comprimento de pêlos,*
- *Uma cor dos pelos,*
- *Uma cor dos olhos,*
- *Um peso,*
- ...

Em POO, utilizamos a palavra **ATRIBUTOS**, ao invés de Características. Então estes são os atributos da classe Cachorro.

Além dos atributos um cachorro também tem AÇÕES. Ele late, corre, anda, deita, dorme... A estas ações nós chamaremos de **MÉTODOS** (que na verdade são funções, da forma como aprendemos em aulas anteriores).

Então, podemos concluir que uma classe é composta de ATRIBUTOS e MÉTODOS (características e ações).

Os atributos de uma classe recebem valores. Por exemplo, um dos atributos da classe cachorro é a variável “nome”. Esta variável receberá um valor que, no caso é o nome do cachorro.

O cachorro do nosso exemplo poderia ser representado pelos seguintes atributos:



Cachorro	
Nome:	Pluto
Idade:	2 anos
Comprimento dos pêlos:	Curtos
Cor dos pêlos:	Bege
Cor dos olhos	Castanhos
Peso:	5 Kg

Um outro objeto “cachorro” apresentaria valores diferentes para os mesmos atributos, por exemplo:



Cachorro	
Nome:	Snoopy
Idade:	4 anos
Comprimento dos pêlos:	Compridos
Cor dos pêlos:	Cinza
Cor dos olhos	Pretos
Peso:	8 Kg

Veja que utilizamos a mesma classe Cachorro, mas agora com informações diferentes.

## Modelando/projetando um sistema

Vamos então a um exercício prático, que nos servirá de exemplo neste estudo:

### ➤ Exercício 1:

Para atender as necessidades de informação de uma biblioteca universitária foi proposto um sistema que deve atender as seguintes características:

- O cadastro dos usuários da biblioteca com endereço completo. Os usuários podem ser classificados em três grupos: Professores, Alunos e Funcionários.
- O cadastro das obras da biblioteca, que podem ser classificadas em: Livros científicos, periódicos científicos, periódicos informativos, periódicos diversos, entretenimento, etc.
- A língua em que se encontra o exemplar da obra.
- A mídia onde se encontra o exemplar da obra.
- Os autores da obra com o controle da nacionalidade do autor.
- As editoras dos exemplares com o ano de edição de cada exemplar.

Identifique os possíveis objetos com seus respectivos atributos e métodos.

## DICAS PARA SOLUÇÃO

Não existe uma forma exata de modelar um sistema, mas deve-se sempre buscar a melhor forma de organizar suas classes. Neste exercício é possível já identificarmos as classes que precisamos criar. De início já podemos perceber que precisaremos criar a classe **Usuário** e que existem **três tipos de usuários**. Nesse caso, poderemos ter na classe Usuário, um atributo com o nome de “tipo”, de modo que possamos atribuir valores a ela conforme for o tipo do usuário. Podemos por exemplo, definir que quando o usuário for um professor, a variável “tipo” receberá o valor 1; quando for um Aluno, receberá o valor 2 e quando for um funcionário, valor 3. Assim conseguiremos diferenciar os tipos ou grupos de usuários.

No texto não foi informado que outros atributos teremos na classe Usuário, então fica a seu critério definir estes atributos. - A classe Usuário poderá ter por exemplo, os atributos Nome, idade, endereço, etc.

Por enquanto, não vamos nos preocupar com os métodos (funções), mas somente com os atributos.

Em seguida temos o cadastro de Obras da biblioteca. Perceba que há vários tipos de obras na biblioteca. Nesse caso, precisaremos criar um atributo na classe livro com o nome de “tipo de livro”. Com este atributo poderemos diferenciar um livro de outro, utilizando um número para cada tipo, da mesma forma como fizemos com a classe Usuario.

O que diferencia um do outro são os valores que cada objeto recebe.

Além do atributo “Tipo de livro”, podemos já identificar alguns outros atributos da classe Livro, que já estão no próprio texto. Quais são eles? - a língua (idioma), a mídia, os autores e as editoras.

Faça a representação gráfica destas classes com seus atributos e em seguida tente criar o código de cada classe.

Exemplo de representação gráfica de uma estrutura de classes:

Nome da classe
atributo 1
atributo 2
atributo 3
...

Aqui não colocamos valores nos atributos da classe. Basta somente o nome da classe e o nome dos atributos.

## Aprofundando um pouco mais neste exercício

Na classe Usuário foi sugerido que criássemos um atributo com o nome de “endereço”. Mas veja que um endereço contem várias informações como: CEP, rua, bairro, País, cidade, número da casa, etc. Então como faríamos para guardar todas estas informações dentro de uma única variável (Endereço) ?

Daí você vai lembrar do que já aprendemos sobre Array. Certo? Só que agora que estamos aprendendo sobre Programação Orientada a Objetos, o correto não é utilizar Array, mas sim os Objetos. Ou seja, ao invés de criarmos um array para guardar todas as informações do Endereço, criaremos um Objeto, que também serve para guardar várias informações. Nesse caso criaremos a classe “Endereco” e os atributos desta classe serão aqueles que já citamos.

Agora dentro da Classe “Usuario” utilizaremos uma outra classe, a classe Endereco. Sim, é possível utilizar uma classe dentro da outra. Mas isso não quer dizer que teremos as duas classes dentro de um mesmo arquivo. Lembra-se do comando “require\_once”, que utilizamos anteriormente (página 26) ?

Sendo assim o atributo “endereco” que teremos na Classe Usuário, na verdade será um objeto da classe Endereco que acabamos de criar.

A mesma coisa acontece com o atributo “Autores” que temos na classe “Obras” da biblioteca. Um livro pode ter vários autores então, como vou guardar dentro de uma única variável o nome de todos os autores? Nesse caso, este atributo da classe “Obras”, deve ser capaz de guardar uma lista de autores. - Veja que este caso é um pouco diferente do problema do atributo “endereco” da classe Usuário, mas a solução é bem parecida. Veremos isso mais adiante.

Por hora, vamos praticar mais alguns exercícios que ajude a fixar melhor estes conteúdos.



## Atividades de POO – Aula 6

1 - Para atender a necessidade de informação de uma loja de Material de construção, foi proposto um sistema com as seguintes características:

- Necessário ter cadastro de Usuários, com nome, endereço, telefone, login e senha.
- Necessário ter cadastro de Clientes com nome, endereço, telefone, sexo, data de nascimento.
- O endereço precisa ter logradouro, bairro, cidade, cep, uf.
- Necessário cadastrar os produtos da loja, com os seguintes campos: código, descrição, peso, fabricante, número do código de barras.

Acrescente ao sistema o que mais achar necessário.

### DICAS

a) Identifique quais classes este sistema terá.

b) Observe que A classe Usuário e a classe Cliente tem alguns atributos iguais. Nesse caso, a recomendação é criar uma classe Pai com o nome de “Pessoa” e as classes filhas “Usuario” e “Cliente”, de forma que os atributos que são iguais não fiquem repetidos nas duas classes, mas sim somente na classe Pessoa. Veja o exemplo que está representando no gráfico da página 28.

Crie o código das classes deste sistema e faça o código que utilizará estas classes para atribuir valores da forma como está o código da página 26, em que foi utilizado o “require\_once” e o comando “**new()**”, para criar instâncias das classes.

2 – Uma agência de viagens precisa de um sistema para gerenciar seu negócio. O sistema precisa ter as seguintes especificações.

- Cadastro de clientes: nome, endereço, telefone, e-mail, data de nascimento, estado civil, profissão e cpf.
- Cadastro de Funcionários: nome, endereço, telefone, data de nascimento, função.
- Cadastro de hotéis: nome, quantidade de quartos, quantidade de estrelas, telefone e endereço.
- Todos os endereços precisam ter: País, cidade, cep, rua, bairro, número, complemento.
- Cadastro de empresas de aviação: nome, telefones, site, e-mail, pessoa de contato.
- Agenda de viagens: nome do cliente, data da ida e data da volta da viagem, hotel e empresa de aviao.

Crie o código das classes deste sistema da mesma forma que fez no exercício 1.

CONTEÚDO PARA ESTUDO: <https://www.devmedia.com.br/introducao-a-orientacao-a-objetos-em-php/26762>

Neste link, aprenda sobre: visibilidade dos atributos da classe (public, private e protected); uso das funções “Get” e “Set” para atribuir valores e exibir valores dos atributos; Uso de herança nas classes e Uso de construtores nas classes.

Qualquer dúvida, só perguntar.



## Aula 7: - Banco de dados.

Todo sistema precisa armazenar dados, de forma que possa ser acessado pelos usuários, para leitura, alteração, inclusão ou exclusão dos seus dados. Por exemplo o cadastro de alunos de uma escola, precisa estar guardado ou armazenado em algum lugar, de forma que possa ser acessado pelo sistema de cadastros.

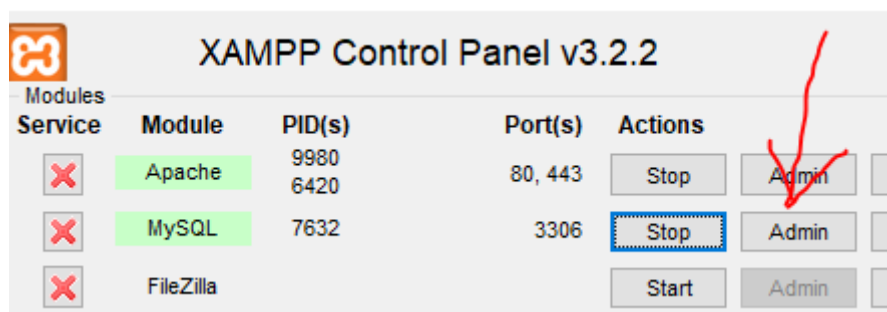
O banco de dados é o recurso que utilizamos para armazenar estes dados. Ele nada mais é do que um conjunto de tabelas que se relacionam entre si. Imagine aqui, como sendo uma tabela do word, por exemplo, ou uma planilha do Excel.

Agora que aprendemos o que é uma classe, com atributos e métodos, imaginemos a tabela do banco de dados como sendo uma classe. A diferença é que a tabela do banco possui somente atributos. Sendo assim, da mesma forma como especificamos os atributos de uma classe, especificaremos também os atributos de cada uma das tabelas do banco de dados. Cada tabela precisa ter um nome, como também cada um dos seus atributos precisa ter um nome. Além do nome, cada atributo precisa ter um tipo (para guardar letras ou números) e um tamanho específico (quantidade máxima de caracteres ou números que posso guardar nela).

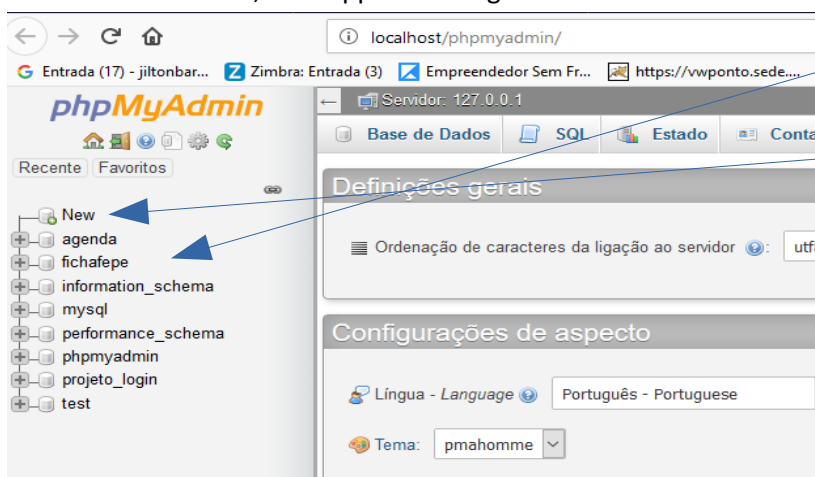
Para fazer uso de um banco de dados no nosso sistema, precisamos utilizar um Gerenciador de Banco de Dados (SGDB). Existem vários tipos de banco de dados e para cada um deles existe um gerenciador próprio dele. No nosso caso, utilizaremos o gerenciador de banco de dados MySQL e também o sistema PhpMyAdmin para facilitar o uso deste gerenciador.

Além do MySQL existem no mercado vários outros bancos de dados diferentes, como o PostgreSQL, Oracle, SQL Server (da Microsoft), o Access, Paradox, DB2 e muitos outros. O MySQL é um dos mais utilizados, por ser gratuito, robusto e simples de usar. No caso de sistemas de grande porte, que exigem um alto grau de processamento de dados, recomenda-se SGDBs mais robustos que o MySQL, como é o caso do Oracle, SQL Server, PostgreSQL e outros.

Para acessarmos o PhpMyAdmin, que nos possibilitará utilizar o MySQL, deveremos clicar no ícone correspondente no painel de controle do Xampp.





Ao clicar neste link, o Xampp abre a seguinte tela:





## 7.1. Criando nosso primeiro banco de dados

Ao clicar no link “New”, indicado na imagem anterior. A tela será a seguinte:

### Base de Dados

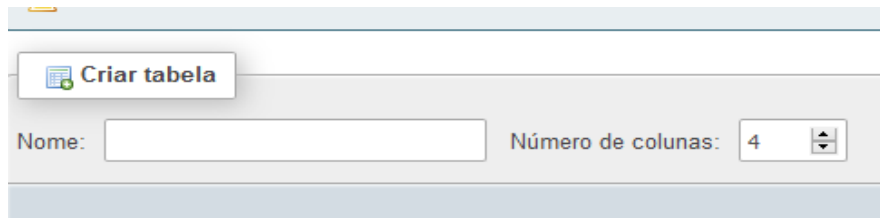
 Criar base de dados 

Nome da base de dados:   

Aqui você deverá informar o nome que dará ao seu banco de dados (este nome não pode ter acentuação e nem espaços em branco).

Tomemos como exemplo o **exercício número 2**, da aula anterior, em que foi pedido para criarmos as classes de uma agência de viagens. Nesse caso, o nome do nosso banco poderá ser “**agenciaviagens**” ou “**agencia\_viagens**” (veja que para não deixar espaço em branco no nome, você pode utilizar o traço **\_** (*underscore*), ou ainda “**agenciaViagens**”. Escolha a forma do nome que for da sua preferência.

Após informar o nome e clicar no botão “criar”, a tela seguinte será esta:













Aqui, ele pede para você criar a primeira tabela do seu banco de dados e também é necessário informar quantas colunas terá sua tabela. Para saber qual a quantidade de colunas, a princípio podemos considerar a mesma quantidade de atributos que tem uma das classes do seu sistema de viagens. Por exemplo, vamos supor que criaremos a tabela para guardar os dados dos clientes da nossa agência de viagens. Repare no exercício 2 da aula anterior, que nossa classe de clientes tem 8 atributos, portanto, sua tabela de clientes no banco de dados poderá ter inicialmente 8 colunas. Mas, já vou logo adiantando um outro detalhe: toda tabela normalmente precisa ter um campo para identificar cada um dos registros da sua tabela. Imagine por exemplo que sua tabela de clientes terá que guardar/armazenar uns 200 a 500 clientes ou mais. Nesse caso, cada cliente precisa ter um número de registro para ajudar a identificá-lo na lista de clientes. Então nossa tabela terá além das 8 colunas, uma coluna a mais, totalizando **9 colunas**.

O nome da tabela poderá ser “cliente” (recomenda-se utilizar nomes sempre no singular), e a quantidade de coluna será 9. Preencha estas informações e clique no botão “Executar” que está no lado direito da tela.

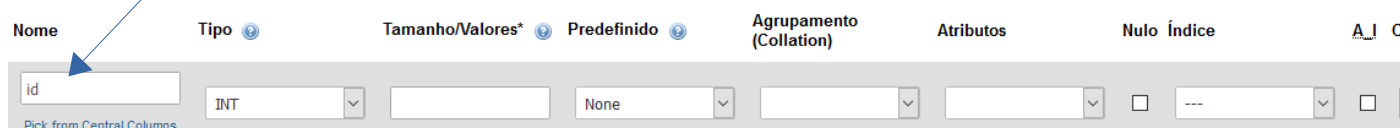
A tela seguinte será esta:

Table name:  Adicionar  column(s) 

Estrutura 								
Nome	Tipo 	Tamanho/Valores* 	Predefinido 	Agrupamento (Collation)	Atributos	Nulo	Índice	A_I Comentários
<input type="text" value=""/> <small>Pick from Central Columns</small>	INT 	<input type="text" value=""/>	None 	<input type="text" value=""/>	<input type="text" value=""/>	<input type="checkbox"/>	---	<input type="checkbox"/>
<input type="text" value=""/> <small>Pick from Central Columns</small>	INT 	<input type="text" value=""/>	None 	<input type="text" value=""/>	<input type="text" value=""/>	<input type="checkbox"/>	---	<input type="checkbox"/>
<input type="text" value=""/> <small>Pick from Central Columns</small>	INT 	<input type="text" value=""/>	None 	<input type="text" value=""/>	<input type="text" value=""/>	<input type="checkbox"/>	---	<input type="checkbox"/>

O phpMyAdmin listou os nove campos da sua tabela e para cada campo, além do nome você terá que informar alguns dados a mais.

O primeiro campo da tabela, recomenda-se que seja o campo de identificação dos registros. O nome para este campo que normalmente é utilizado entre os programadores de sistemas é o termo “id”, então coloque este nome no primeiro campo.

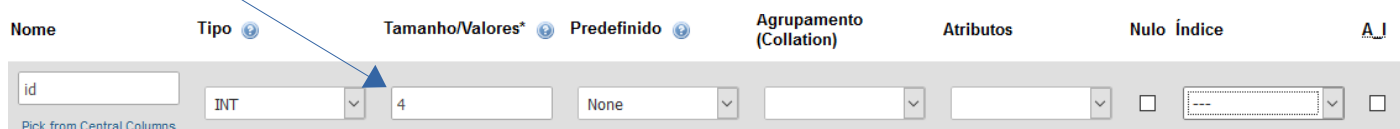


Agora veremos quais informações precisamos preencher nos campos seguintes do campo ID.

Veja que ele pede para informar o Tipo e já vem selecionado o tipo INT. INT quer dizer “tipo inteiro”. O tipo inteiro é o tipo que aceita armazenar nele somente dados do tipo numéricos. No caso do campo ID, ele deverá ser do tipo numérico, então deixemos como tipo INT.

Na coluna seguinte ele pede para informarmos o tamanho deste campo ID. Recomenda-se utilizar tamanhos que seja adequado para a quantidade de registros que você pretende armazenar no seu banco de dados. Por exemplo, se pretende guardar entre 200 a 500 registros, então o campo ID terá o tamanho 3, para conseguir guardar números com tamanho de até 3 dígitos. Mas se pretende guardar no seu banco entre 1000 a 2000 registros, então é necessário que o tamanho do campo ID seja de até 4 dígitos.

Vamos considerar que guardaremos de 1000 a 2000 registros, então o tamanho do campo que informaremos será de 4 dígitos:

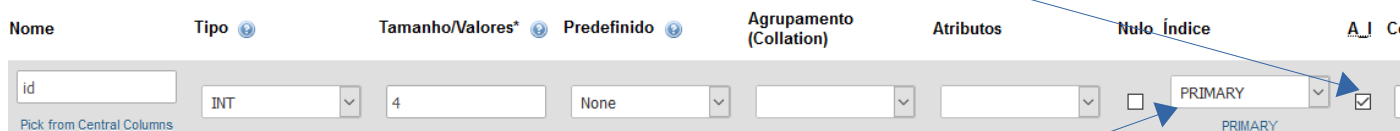


Os campos “Predefinido”, “Agrupamento” e “Atributos” não precisamos preencher nada agora. Já o campo “Nulo” serve para informar se o campo ID poderá aceitar valores vazios ou não. Se você marcar este campo, está informando que o campo ID poderá aceitar valores vazios. Mas, no caso do ID, que é um identificados para cada registro da lista de clientes, este campo não poderá ser NULO, então não marque esta opção. OK?

Em seguida, temos o campo “Índice”. Este campo é importante para o campo ID. - Preste atenção nesta dica: Se o campo ID é um identificador para cada cliente da sua lista de clientes, então este ID terá que ser um número que não se repete na sua lista, ou seja, nenhum cliente poderá ter o número do ID repetido. Cada um terá o seu.

A forma de evitar que o número do ID se repita no seu banco é selecionando no campo “Índice”, a opção “PRIMARY”.

Na coluna seguinte, deixe marcada a **opção “A\_I”**. A\_I significa “Auto Increment”, ou seja, “Auto numeração”. O número do ID irá aumentar sequencialmente e automaticamente, à medida que for adicionando novos clientes no seu banco de dados.



Aqui, deixe selecionado a opção “PRIMARY”, somente para o campo ID

Então, conforme está no exercício número 2, da aula anterior, temos os seguintes atributos na classe Cliente:

**nome, endereço, telefone, e-mail, data de nascimento, estado civil, profissão e cpf**

Cada um destes atributos será um campo (ou uma coluna) na nossa tabela do banco de dados).

Vamos agora preencher a linha seguinte com o atributo “nome”.

Nome	Tipo	Tamanho/Valores*	Predefinido	Agrupamento (Collation)	Atributos	Nulo	Índice	A_I	Come
id	INT	4	None			<input type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>	
nome	VARCHAR	100	None			<input type="checkbox"/>	---	<input type="checkbox"/>	

Pronto! - Observe que o nome do atributo da classe cliente vai ser o mesmo nome do campo(coluna), da nossa tabela. Veja que o tipo que utilizaremos para ele é o VARCHAR. Esse tipo serve para armazenarmos tanto letras quanto números, mas é mais apropriado para somente letras. E o tamanho do campo colocamos o valor 100, considerando que o nome de um cliente não terá mais que 100 letras e espaços. Não marcamos o campo “Nulo”, porque também não será aceito cadastro de clientes sem nome e também não selecionaremos nenhuma opção no campo Índice, por considerarmos que poderá ocorrer de dois clientes terem o mesmo nome.

Vamos ao campo seguinte: endereço.

Nome	Tipo	Tamanho/Valores*	Predefinido	Agrupamento (Collation)	Atributos	Nulo	Índice	A_I	Come
id	INT	4	None			<input type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>	
nome	VARCHAR	100	None			<input type="checkbox"/>	---	<input type="checkbox"/>	
endereço	VARCHAR	200	None			<input checked="" type="checkbox"/>	---	<input type="checkbox"/>	

O campo endereço também vai ser do tipo VARCHAR, com tamanho 200 e agora, somente para mostrar o uso da opção NULO, deixei-a marcada. Nesse caso, o sistema poderá aceitar clientes sem endereço, ou seja, este campo poderá ficar vazio. Outro detalhe importante: repare que o nome do campo ficou sem o cedilha “endereço”. Não se pode utilizar acentuação, nem espaços em branco no nome dos campos e nem cedilha. Por prática de mercado, todos os nomes dos campos devem ser em **letras minúsculas**.

Os campos seguintes “telefone”, “e-mail”, “profissao” e “cpf” também são VARCHAR, podem ter o tamanho 100 (somente o cpf que pode ter o tamanho 20) e também podem ser NULOS.

O nome do campo “data de nascimento” será “datanascimento” ou abreviando “dtnascimento” se preferir. O tipo deste campo poderia ser o tipo DATE, mas por enquanto utilizaremos o tipo VARCHAR para simplificar.

O campo “Estado Civil”, poderá ter o nome “estadocivil”. Este campo poder ser VARCHAR com tamanho 1 (um), para armazenar somente um letra. Por exemplo a letra “C” se o cliente for casado ou “S” para solteiro e etc.

Mas também, se for da sua preferência, este campo poderá ser do tipo INT e armazenará, por exemplo o número 1 para casado, 2 para solteiro, 3 para separado e assim por diante.

## ATIVIDADE 1

Crie os demais campos e, no final da tela clique no botão “Guarda” (no canto direito inferior), para gerar a tabela “cliente” no banco de dados.

Após clicar no botão “Guarda”, você verá a seguinte tela:

New	#	Nome	Tipo	Agrupamento (Collation)	Atributos	Nulo	Predefinido	Comentários	Extra	Ações
agencia_viagens	1	id	int(4)			Não	None		AUTO_INCREMENT	Muda
cliente	2	nome	varchar(100)	latin1_swedish_ci		Não	None			Muda
agenda	3	endereço	varchar(200)	latin1_swedish_ci		Sim	NULL			Muda
fichafone										

Veja que o nome do seu banco de dados já aparece na lista à esquerda e que logo abaixo do nome do banco já aparece o nome da tabela “cliente”. À direita temos os nomes dos campos criados , com seus tipos e demais informações.

## **ATIVIDADE 2**

Com base na atividade 1, crie as demais tabelas do seu sistema de agência de viagens, conforme especificado no exercício 2 da aula anterior : funcionários, hotéis, empresas de aviação, agenda de viagens.

## Aula 8: - Padrão MVC para desenvolvimento de sistemas.

O padrão MVC é uma maneira de organizarmos nosso código, separando cada parte dele em uma pasta específica do sistema que estivermos desenvolvendo.

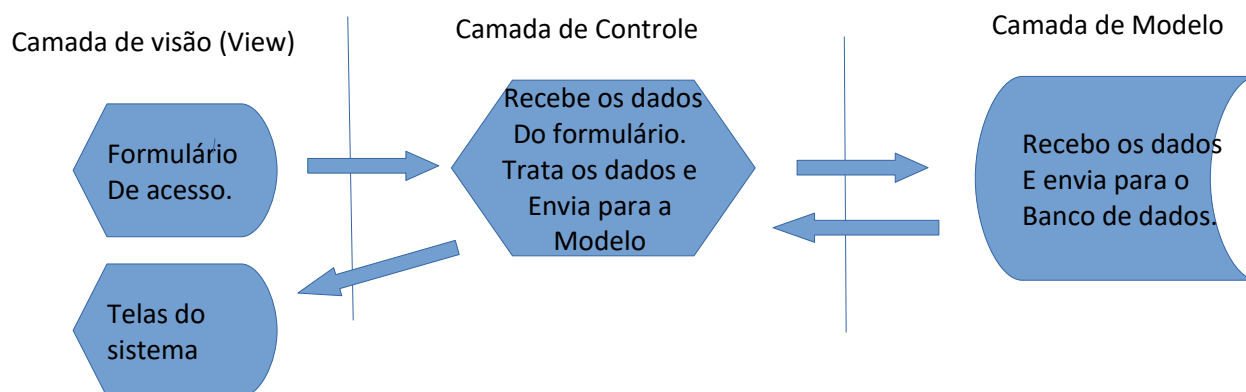
Essa maneira de organizarmos o nosso código separados em pastas é na mais que o projeto arquitetural do nosso sistema. Nesse caso estamos criando uma arquitetura MVC para o nosso sistema. Existem vários outros tipos/ modelos de arquitetura para organização do código de sistemas.

Neste modelo MVC, dizemos que os arquivos do sistema ficam distribuídos em 3 camadas, que não verdade são três pastas que deveremos criar. A primeira pasta é a **M** (Model ou Modelo), a segunda pasta é a **V** (View ou Visão) e a terceira é a **C** (Controller ou controle).

Na camada Model colocaremos os arquivos de código que contem as classes do nosso sistema, que possuem atributos semelhantes às tabelas e campos que criamos no nosso banco de dados. Por exemplo: se no banco de dados criamos uma tabela chamada Cliente, com os campos: nome e endereço, teremos que ter no nosso sistema uma classe com o nome Cliente, com os atributos nome e endereço.

Na camada de visão (View), teremos os arquivos das telas do nosso sistema. Geralmente são arquivos HTML.

Na camada de Controle (Controller), teremos os arquivos de código PHP, que recebem os dados das telas (HTML) e enviam estes dados para as classes da camada de Modelo. A camada de controle tem a função de fazer a ligação entre a camada de Visão e a camada de Modelo, ou seja, entre as telas do sistema e o banco de dados. Nesta camada de controle, faz-se também a verificação e validação dos dados que estão vindo das telas, antes de enviar para a camada de modelo. Por exemplo um formulário de login e senha. Na camada de visão temos o formulário, na camada de controle temos o código PHP que receberá estes dados do formulário, verificará se os campos de login e senha foram preenchidos corretamente e em seguida envia estes dados para a camada de Modelo, para verificar no banco de dados se o login e a senha estão corretos. Se estiverem corretos, a camada de controle recebe mensagem de OK da camada de modelo e então a camada de controle direciona o usuário para as demais telas do sistema.



Leia mais sobre a arquitetura MVC em:

<https://www.devmedia.com.br/conceito-de-mvc-e-sua-funcionalidade-usando-o-php/22324>

## 8.1 Atividade inicial do sistema de viagem.

Sabendo agora como montar a arquitetura de um sistema, já podemos montar então a arquitetura do nosso sistema de viagens. Criaremos uma pasta principal com o nome de “sistemaviagens” e dentro dela as três seguintes pastas: ‘visao’, ‘controle’ e ‘modelo’.

Dentro da pasta de visão já podemos criar nosso formulário de login e na pasta ‘Controle’, o código PHP que receberá os dados do formulário de login.

Na pasta de Modelo, criaremos o arquivo de conexão com o banco de dados. Este arquivo será uma classe, conforme está no código abaixo:

```
1  <?php
2
3  class DB_CONNECT {
4
5      protected $con;
6
7      // constructor
8      function __construct() {
9          // connecting to database
10         $this->connect();
11     }
12
13     function connect() {
14         // importando o arquivo de configuração do banco de dados
15         require_once __DIR__ . '/db_config.php';
16
17         // Connectando com o banco Mysql.
18         $this->con = mysqli_connect(DB_SERVER, DB_USER, DB_PASSWORD,DB_DATABASE) or die(mysql_error());
19
20         return $this->con;
21     }
22
23     function close() {
24         // closing db connection
25         if($this->con)
26             mysql_close($this->con);
27     }
28
29 }
```

Linha 3 – temos o nome da classe que aqui colocamos o nome de ‘DB\_CONNECT’.

Linha 5 – criamos a variável que vai armazenar a conexão do banco, para poder ser utilizada pelas demais classes do nosso sistema.

Linha 8 – Temos o construtor da nossa classe. Como você já sabe, o construtor é uma função que é executada automaticamente no momento em que a classe for instanciada. Ou seja, qualquer outra classe que precisar fazer uma conexão com o banco de dados, vai utilizar nossa classe DB\_CONNECT. Observe que o construtor está executando a função ‘connect()’. É esta função que possui o código de conexão com o banco de dados Mysql.

Linha 15 – nesta linha estamos importando o arquivo ‘db\_config.php’. Este arquivo contém as informações de login e senha, como também o nome do banco de dados que precisaremos acessar. Este arquivo também fica na pasta Modelo. No nosso caso o nome do banco de dados que criamos na aula anterior foi o “**agenciaviagens**” ou “**agencia\_viagens**”. Ficou a seu critério criar com o nome que quiser, mas aqui consideraremos o nome ‘agenciaviagens’.

Veja abaixo como deverá ficar o código do arquivo ‘db\_config.php’.

```
1  <?php
2  define('DB_USER', "root"); // usuário do banco de dados
3  define('DB_PASSWORD', ""); // senha do banco de dados - nesse caso não temos senha.
4  define('DB_DATABASE', "agenciaviagens"); //nome do banco
5  define('DB_SERVER', "localhost"); // servidor em que está o banco
6  ?>
```

A explicação já está em verde no próprio código.

Na linha 3, observe que deixamos a senha em branco (não há nenhum conteúdo entre as aspas). Por padrão o PHPMyAdmin não coloca senha para o usuário 'root'. Mas quando estiver desenvolvendo sistemas reais, é necessário utilizar senha de acesso ao banco de dados, de modo a garantir maior segurança aos dados.

Detalhe importante na linha 5: o termo 'localhost' significa que o banco de dados está no mesmo servidor que estará funcionando o seu sistema, ou seja, um servidor local. Como estamos utilizando o XAMPP então estamos utilizando um servidor local (no nosso próprio computador). Mas quando for utilizar servidores externos, como por exemplo o 'hostgator', o administrador deste servidor deverá lhe informar qual endereço você deverá colocar no lugar do 'localhost'.

Na pasta modelo ainda teremos os códigos das classes que representam as tabelas do banco de dados e os arquivos que executam os comandos de banco de dados, para gravar dados, alterar, excluir ou listar. Todos estes arquivos deverão ser classes do nosso sistema.

Coloque agora, dentro da pasta Modelo, o arquivo da classe "Cliente" que já criamos em aulas anteriores.

Se não tiver criado ainda, reveja as aulas sobre classes.

Além da classe cliente, precisamos criar agora a classe que conterá os comandos de SQL para gravar, alterar, excluir e listar os dados dos clientes. Esta classe poderá chamar-se 'Cliente\_DB.php'.

O código dela será o seguinte:

```
1  <?php
2  include_once ('db_connect.php');
3  include_once ('cliente.php');
4
5  class clientes_db {
6
7      protected $db, $con;
8
9      function __construct(){
10         if($this->con==null){
11             $this->con = new DB_CONNECT();
12             $this->db = $this->con->connect();
13         }
14     }
15
16     function cadastra ($cad){
17         $nome=$cad->getNome();
18         $endereco=$cad->getEndereco();
19
20         $sql_insert="INSERT INTO cliente (nome, endereco)
21                     VALUES ('$nome', '$endereco')";
22
23         $result = mysqli_query($this->db,$sql_insert)
24         or die ("<p class='mensagem'>Erro ao tentar cadastrar cliente. Contate o suporte.
25                 </p><p align='center'><a href='javascript:history.go(-1)'>Voltar</a></p>");
26
27         if ($result)
28             echo "<p class='erro'>Cliente cadastrado com sucesso!</p>";
29     }
30 }
```

**Linhas 2 e 3** – estamos importando a classe de conexão com o banco de dados e a do cliente.

**Linha 7** – criamos as variáveis que serão utilizadas para realizarmos as operações no banco de dados.

**Linha 9** – temos o construtor que realizará a conexão com o banco de dados. Observe que ele armazena a conexão nas variáveis que criamos na linha 7. Observe também que estas variáveis estão sendo utilizadas ao longo de toda a classe (veja a linha 23, por exemplo).



**Linha 16** – temos a função que realiza o cadastro dos dados do cliente no nosso banco de dados. Observe que a variável `$cad`, contem dentro dela, toda a estrutura da nossa classe `Cliente`.

A variável `$cad`, será preenchida na camada de controle com os dados vindos do formulário de cadastro de cliente. Veremos isso com mais detalhes quando estivermos implementando o código das nossas classes de controle.

**Linha 17 e 18** – estamos pegando os valores da classe `Cliente` e atribuindo às variáveis que criamos na nossa classe de modelo, para poder utilizá-las no comando SQL que segue na linha seguinte.

**Linhas 20/21** - temos aqui um comando SQL, que executa o cadastro dos dados do cliente no banco. Observe que estamos armazenando este comando em uma variável com o nome de `$sql_insert` (pode-se utilizar qualquer nome para esta variável). Observe também que o comando SQL está entre aspas, pois para o PHP é somente um texto qualquer, que só será transformado em comando SQL na linha 23.

**Linha 23** - Esta é a linha que executa o comando SQL. Observe que nesta linha estamos utilizando a variável que criamos para armazenar a operação de conexão com o banco de dados (variável `$db`). Se ocorrer algum erro na execução deste comando SQL, uma mensagem de erro será exibida na tela, por meio do comando "OR DIE". A mensagem está em letras cinzas no código acima. Observe que a mensagem contem um comando em JavaScript, que possibilita voltar para a tela de cadastro do cliente.

**Linha 27** – Se o comando SQL for executado com sucesso, o sistema exibe na tela a mensagem que está na linha 28.

Continuando o código da classe 'Clientes\_DB':

```
31 function altera($cad){
32     $id = $cad->getId();
33     $nome=$cad->getNome();
34     $endereco=$cad->getEndereco();
35
36     $sql_altera="UPDATE cliente SET nome='$nome',
37                                     endereco='$endereco'
38
39                                     WHERE id=".$id;
40
41     $result = mysqli_query($this->db,$sql_altera)
42     or die ("<p class='erro'>Erro ao tentar alterar dados do cliente. Contate o suporte.
43             </p><p align='center'><a href='javascript:history.go(-1)'>Voltar</a></p>");
44
45     if ($result)
46         echo "<p class='erro'>Dados do cliente alterados com sucesso!</p>";
47 }
48
49 function exclui($idcliente){
50
51     $sql_exclui="DELETE FROM cliente WHERE id=".$idcliente;
52     $result = mysqli_query($this->db,$sql_exclui)
53     or die ("<p class='erro'>Erro ao tentar excluir cliente. Contate o suporte.
54             </p><p align='center'><a href='javascript:history.go(-1)'>Voltar</a></p>");
55     if ($result)
56         echo "<p class='erro'>Dados do cliente excluídos com sucesso!</p>";
57 }
```

**Linha 31** – temos a função para alterar os dados do cliente. Observe que as primeiras linhas desta função são iguais às linhas da função de cadastro, acrescentando apenas a variável `$id`.

**Linha 36/39** – Temos o comando SQL que executa a alteração dos dados do cliente. Observe que não pode faltar a condição que está na linha 39, de forma que ele altere somente os dados de apenas um cliente. Se não

colocar esta informação, o comando SQL alterará de uma só vez os dados de todos os clientes, repetindo os mesmos dados de um cliente para todos os clientes. Dessa forma você perderá os dados do seu banco e isso é muito ruim. Imagine se você tiver mais de 1000 clientes cadastrados e esquecer de colocar a condição WHERE no seu comando SQL! - vai ser um desastre.

**Linha 49** – Temos a função com o comando de exclusão dos dados do cliente. Aqui também jamais poderá esquecer a condição WHERE com o valor do ID do cliente. Se esquecer esta condição, todos os dados da tabela de clientes serão apagados.

Continuando o código da classe 'Clientes\_DB':

```
61 function listCliente(){
62     $sql = "SELECT * FROM cliente ORDER BY nome";
63     $result = mysqli_query($this->db, $sql);
64
65     $list = array();
66     $i=0;
67
68     while ($row_sql=mysqli_fetch_array($result))
69     {
70         $cliente = new Cliente();
71         $cliente->setId($row_sql['id']);
72         $cliente->setNome($row_sql['nome']);
73         $cliente->setEndereco($row_sql['endereco']);
74
75         $list[]=$cliente;
76         $i++;
77     }
78
79     mysqli_free_result($result);
80
81     return $list;
82 }
83
84 function listClienteById($idcliente){
85     $cliente = new Cliente();
86
87     $sql = "SELECT * FROM cliente where id='$idcliente'";
88     $result = mysqli_query($this->db, $sql);
89     $row_sql=mysqli_fetch_array($result);
90
91     $cliente->setId($row_sql['id']);
92     $cliente->setNome($row_sql['nome']);
93     $cliente->setEndereco($row_sql['endereco']);
94
95     mysqli_free_result($result);
96
97     return $cliente;
98 }
99
100 }//fim da classe cliente_DB
```

**Linha 61** – temos a função que lista todos os clientes que estiverem cadastrados no banco de dados, ordenados pelo nome, em ordem alfabética.

**Linha 65** - Criamos um array, que armazenará a lista de clientes, para poder exibi-la na tela do sistema.

**Linha 68** – Estamos utilizando um WHILE, para percorremos toda a lista que veio do banco de dados e armazenar cada um dos registros na classe Cliente (Linha 70 a 73), e em seguida armazenar esta classe no array que

criamos na linha 65. Veja que na linha 75 estamos guardando no array a classe Cliente, já com os dados vindo do banco .

**Linha 81** - O comando “RETURN” envia a lista de clientes para a camada de controle. Esta por sua vez, pegará estes dados e enviará para a camada de visão, para poder ser exibida toda a lista para o usuário do sistema.

**Linha 84** – Esta função é bem semelhante à função de listagem dos clientes. A diferença que aqui ele vai trazer do banco de dados somente um cliente. Esta função é utilizada quando precisamos editar na tela do sistema os dados do cliente para serem alterados.

Nesta classe poderemos criar ainda muitas outras funções que sejam necessárias para trabalharmos com os dados dos clientes. Por exemplo, poderíamos ter uma função que busca, ou faz uma pesquisa pelo nome do cliente.

Podemos ter funções que listam clientes por cidade, por idade, por data de nascimento, para sabermos que são os aniversariantes do mês, e assim por diante.

Veja que é um pouco trabalhoso criar todos estes comandos uma a uma. Hoje em dia os profissionais que trabalham com desenvolvimento de sistemas, costumam utilizarem Frameworks, que são ferramentas de programação que facilitam muito a criação destas classes e comandos. Mais adiante poderemos conhecer um destes frameworks. Existem vários deles no mercado, muitos gratuitos. Já posso adiantar que um deles é o CodeIgniter. Pesquise no Google sobre ele, pelo menos por curiosidade por enquanto.