

## 5.0 Interações PHP com HTML

### Objetivos

Apresentar ao aluno como trabalhar com interações PHP dentro do código HTML; Mostrar exemplos de formulário, o uso de métodos POST e GET, cookies, listagem, seção e suas interações com o Browser.

Abordaremos neste capítulo algumas formas de interações utilizando a linguagem de programação PHP e a linguagem de marcação HTML. Além disso, mostraremos alguns componentes mais utilizados para a construção de um sistema ou uma página web, e de que forma o PHP pode receber, processar, e enviar essa informação.

Utilizamos linguagens para exibir os dados da aplicação, seja ela para simples conferência, em relatório ou ainda possibilitando a adição e exclusão de registros. Criaremos a princípio formulários e listagem.

### 5.1 Formulários

Podemos definir formulário como um conjunto de campos disponíveis ao usuário de forma agrupada para serem preenchidos com informações requisitadas pela aplicação (sistemas web ou páginas). Um formulário é composto por vários componentes, além de possuir botões de ação, no qual define o programa que processará os dados.

Em uma aplicação determinamos então a entrada de dados (no caso os formulários), e a saída de dados, que é toda e qualquer informação apresentada ao usuário pelo browser, de forma que ambas tenham uma ligação lógica e possa retornar um resultado onde todos os componentes da aplicação trabalhem de forma coerente.

#### ELEMENTOS DE UM FORMULÁRIO.

Para criarmos um formulário, utilizamos a tag <form> e dentro dela podemos dispor diversos elementos, onde, cada um deles representa uma propriedade em particular. A seguir explicaremos os principais componentes de um formulário.

Criando um formulário:

Todo formulário deve conter no mínimo as seguintes características, observe:

```
1 <form name=" " method=" " action=" ">
2   ... elementos ...
3 </form>
```

**name** < Nome atribuído ao formulário para sua identificação. **method** < Método POST ou GET como veremos mais adiante.

**action** < Caminho do arquivo que receberá os dados do formulário ao ser enviado.

Os elementos do formulário serão preenchidos com os componentes input onde a tag é <input> conforme o exemplo abaixo:

```
1 <form name=" " method=" " action=" ">
2   ...
3   <input name=" " value=" " type=" ">
4   ...
5 </form>
```

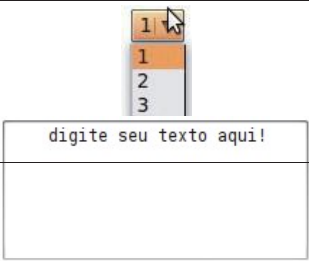
A tag input pode ser composta por vários elementos onde neles podemos definir o tipo, nome, e o valor padrão além de outras propriedades que não apresentaremos nessa apostila.

**name** < Nome atribuído ao componente para sua identificação. **value** < Valor padrão que pode ser atribuído no formulário.

**type** < Tipo de input, é nele onde definimos o tipo elemento o input vai representar.

Observe a tabela abaixo com a definição dos tipos que podemos atribuir ao elemento type do input.

type	Descrição	Exemplo:
texto	Elemento utilizado para entrada de texto simples, é um dos mais utilizados.	<input type="text" value="texto"/>
password	Elemento utilizado para entrada de senhas, exibe " " no lugar dos caracteres inseridos pelo usuário.	<input type="password" value="••••"/>
checkbox	Utilizado para exibir caixa de verificação, muito utilizado para perguntas booleanas.	1 <input checked="" type="checkbox"/> 2 <input checked="" type="checkbox"/>
radio	Exibe botões para seleção exclusiva, utilizado para seleção de apenas um item.	sexo: M <input checked="" type="radio"/> F <input type="radio"/>
file	Utilizado para seleção de arquivos, esse tipo é muito comum quando queremos enviar algo pelo navegador (browser).	<input type="file"/> Enviar arquivo...
hidden	É utilizado para armazenar um campo escondido dentro do formulário.	Não é visível ao usuário.
button	Usado para exibir um botão na tela, porém sua ação é definida por outra linguagem como javascript.	<input type="button" value="enviar"/>
submit	Botão usado para submeter os dados do formulário no servidor, ele envia as informações de acordo com as informações preenchidas na tag <form>.	<input type="submit" value="enviar"/>
reset	Utilizado para limpar todos os campos do formulário, voltando ao valor inicial.	<input type="reset" value="resetar"/>

<b>select</b>	Tipo utilizado para exibir uma lista de valores contido na lista de seleção do usuário (cada valor é guardado em uma tag <option>, porém só pode ser selecionado uma opção.	
<b>Tag:</b> <b>&lt;textarea&gt;</b>	Área de texto disponibilizado ao usuário, possui múltiplas linhas (rows) e colunas (cols) que podem ser determinados dentro de sua tag. Não é um tipo de input, mas é uma tag HTML para formulário.	

```
<textarea name="area" rows="5" cols="27">
  digite seu texto aqui!
</textarea>
```

A tag <textarea> pode ser definida conforme o exemplo de código abaixo:

Observe que definimos uma caixa de texto com 5 linhas e 27 colunas, isso define o tamanho que essa caixa vai ter dentro do formulário.

## 5.2 Exemplo de formulário.

Veremos agora exemplos de alguns formulários comumente encontrados, usando os componentes apresentado anteriormente. Não entraremos em detalhes do HTML, sendo apresentado de maneira direta, pois está subentendido que o aluno já conheça HTML. Observe os exemplos abaixo:

Tela de login:

**código-fonte:**

```
1 <form name="cadastro" method="post" action="gravar.php">
2   <table border="1" width="400px" bgcolor="#f0f0f0"
3     style="border-collapse: collapse">
4     <tr>
5       <td>Login :</td>
6       <td><input type="text" value="Nome completo."></td>
7     </tr>
8     <tr>
9       <td>Senha :</td>
10      <td><input type="password" value="12345678"></td>
11    </tr>
12  </table>
13  <input type="submit" value="Entrar">
14 </form>
```

Resultado:

Login :	<input type="text" value="Nome completo."/>
Senha :	<input type="password" value="12345678"/>
<input type="submit" value="Entrar"/>	

Tela de Cadastro:

**código-fonte:**

```
1 <form name="cadastro" method="post" action="gravar.php">
2   <table border="1" width="400px" bgcolor="#f0f0f0"
3     style="border-collapse: collapse">
4     <tr>
5       <td>Nome :</td>
6       <td><input type="text"></td>
7     </tr>
8     <tr>
9       <td>Endereço :</td>
10      <td><input type="text"></td>
11    </tr>
12    <tr>
13      <td>Cidade :</td>
14      <td><input type="text"></td>
15    </tr>
16    <tr>
17      <td>Estado :</td>
18      <td><input type="text" size="3" maxlength="2"></td>
19    </tr>
20  </table>
21  <input type="submit" value="Cadastrar">
22 </form>
```

Resultado com dados preenchidos:

Nome :	<input type="text" value="Alex Sousa"/>
Endereço :	<input type="text" value="Rua 123"/>
Cidade :	<input type="text" value="Fortaleza"/>
Estado :	<input type="text" value="CE"/>
<input type="submit" value="Cadastrar"/>	

Tela de envio de dados e arquivos:  
código-fonte:

```

1 <form name="cadastro" method="post" action="gravar.php">
2   <table border="1" width="400px" bgcolor="#f0f0f0"
3     style="border-collapse: collapse">
4     <tr>
5       <td>Titulo:</td>
6       <td><input type="text" ></td>
7     </tr>
8     <tr>
9       <td>Nome do autor:</td>
10      <td><input type="text" ></td>
11    </tr>
12    <tr>
13      <td>Imagem:</td>
14      <td><input type="file" ></td>
15    </tr>
16    <tr>
17      <td>Descrição:</td>
18      <td><textarea rows="6" cols="35">
19        Digite sua descrição aqui.
20      </textarea></td>
21    </tr>
22  </table>
23  <input type="submit" value="Enviar dados">
24 </form>

```

Resultado:

Titulo:	<input type="text"/>
Nome do autor:	<input type="text"/>
Imagem:	<input type="file"/> Enviar arquivo...
Descrição:	Digite sua descrição aqui. <input type="text"/>
<input type="submit" value="Enviar dados"/>	

Com esses exemplos já podemos trabalhar com as informações até agora mostrada. Para isso vamos conhecer os dois principais métodos de envio de dados de um formulário para um arquivo PHP.

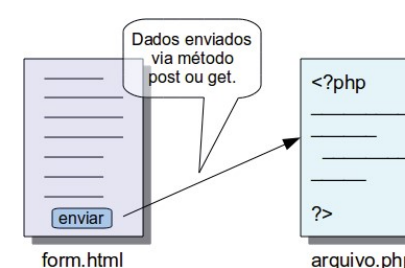
### 5.3 Métodos Post e Get

Quando falamos em como enviar dados para um formulário, deve vir em mente os métodos GET e POST, que são os métodos utilizados. Mas quando fazemos uma requisição HTTP, sempre utilizamos um desses métodos, normalmente o GET.

Se você digita um endereço na barra de endereço seu navegador e aperta a tecla enter (ou clica no botão ir), o navegador faz uma requisição HTTP para o servidor do endereço digitado e o método dessa requisição é o GET. Se você clica em um link em um site, o navegador também se encarrega de fazer uma requisição HTTP com o método GET para buscar o conteúdo da página que você clicou.

Esse mecanismo funciona da seguinte forma. Primeiro temos em um formulário, um botão ou link. Quando clicamos em umas dessas propriedades, estamos enviando uma requisição. A forma de enviar pode ser definida pelo método get ou post e deve ser enviada para algum arquivo, que ao receber, processe a informação devolvendo resultado ou não.

Veja  
abaixo:



a ilustração

### 5.4 Método Get

O método GET utiliza a própria URI (normalmente chamada de URL) para enviar dados ao servidor. Quando enviamos um formulário pelo método GET, o navegador pega as informações do formulário e coloca junto com a URI de onde o formulário vai ser enviado e envia, separando o endereço da URI dos dados do formulário por um "?" (ponto de interrogação) e "&".

Quando você busca algo no Google, ele faz uma requisição utilizando o método GET, você pode ver na barra de endereço do seu navegador que o endereço ficou com um ponto de interrogação no meio, e depois do ponto de interrogação você pode ler, dentre outros caracteres, o que você pesquisou no Google.

Abaixo temos um exemplo de uma URL do gmail da google. Observe:

google.com <https://mail.google.com/mail/?hl=pt-br&shva=1>

Podemos notar a passagem de dois valores:

?hl = pt-br, logo após &shva=1, ou seja, temos então a criação da "variável" **hl** que recebe o valor pt-br, e também a "variável" **shva** que recebe como valor 1. Veja exemplos de códigos com links enviando valores por métodos GET.

```

1 <a href="gravar.php?pagina=3&conteudo=4">Informação</a>
2 <a href="gravar.php?pg=24&user=anonimo">Entrar</a>
3 <form method="get" action="gravar.php">
4     Nome:<input name="nome" type="text">
5     <input type="submit" value="Enviar">
6 </form>

```

Ao clicarmos em um desses links, podemos observar o seguinte comportamento na URL do navegador:

**Link informação** < Estamos enviando dois valores via método GET, um contendo 3, e o outro 4, para o arquivo gravar.php .

**Link Entrar** < Estamos enviando dois valores via método GET, um contendo 24, e o outro anonimo, para o arquivo gravar.php .

### Formulário(<form>)

< Envias as informações preenchidas no **input** nome via GET para o arquivo gravar.php.

## RECEBENDO DADOS VIA MÉTODO GET

Agora trabalharemos com o arquivo PHP, onde podemos resgatar os valores enviados pelo método \$\_GET, sua sintaxe é a seguinte:

`$_GET['nome_da_campo']` < retorna o valor passado pelo campo.  
`$_GET` < retorna um **array** com todos os valores enviados e seus supostos índices.

Quando queremos um valor específico, colocamos o nome da “variável” da URL ou o nome atribuído na propriedade name do input do formulário.

Exemplo com links:

código – HTML:

```
<a href="gravar.php?pagina=3&conteudo=4">Informação</a>
```

código – PHP (nome do arquivo: gravar.php):

```

1 <?php
2
3 // Imprime valores específico
4 echo $_GET['pagina']."<br>";
5 echo $_GET['conteudo']."<br>";
6 // monta um array com todos valores enviados.
7 $a = $_GET;
8 foreach($a as $nome => $valor)
9     echo $nome." = ".$valor."<br>";
10
11 ?>

```

Resultado:

```

3
4
pagina = 3
conteudo = 4

```

Exemplo com Formulário:

código – HTML:

```

1 <form method="get" action="gravar.php">
2     Login: <input name="login" type="text">
3     Senha: <input name="senha" type="password">
4     <input type="submit" value="Logar">
5 </form>

```

Resultado no Browser:

Login:  Senha:

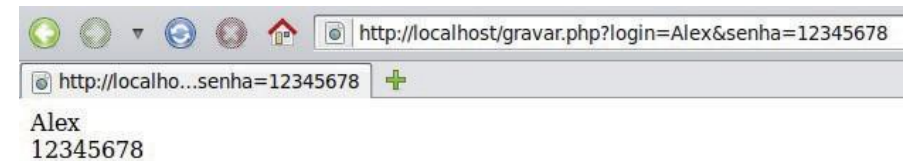
código – PHP (nome do arquivo: gravar.php):

```

1 <?php
2
3 echo $_GET['login']."<br>";
4 echo $_GET['senha']."<br>";
5
6 ?>

```

Resultado:



Em casos que precisamos de mais segurança, onde o ocultamento de informação é necessário, o método GET não é uma boa opção. Observe no exemplo anterior que qualquer valor passado pela URL fica visível, e o usuário pode ver informações pessoais como um login e uma senha.



Dica: Em casos semelhante, utilize sempre o método POST.

## 5.5 Método Post

Muito semelhante ao método GET, porém a principal diferença está em enviar os dados encapsulado dentro do corpo da mensagem. Sua utilização é mais viável quando trabalhamos com informações seguras ou que podem ser alteradas somente por eventos do Browser. Sintaxe:

`$_POST['nome_da_campo']` < retorna o valor passado pelo campo.  
`$_POST` < retorna um **array** com todos os valores enviados e seus supostos índices.



Veja o mesmo exemplo anterior, porém com o uso de POST:

Exemplo com Formulário:

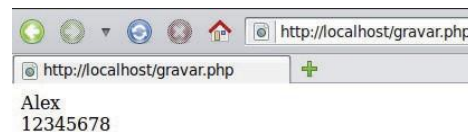
código – HTML:

```
1 <form method="post" action="gravar.php">
2     Login: <input name="login" type="text">
3     Senha: <input name="senha" type="password">
4           <input type="submit" value="Logar">
5 </form>
```

código – PHP (nome do arquivo: gravar.php):

```
1 <?php
2
3 echo $_POST['login']. "<br>";
4 echo $_POST['senha']. "<br>";
5
6 ?>
```

Resultado após o envio dos dados preenchidos:



Mudança no método

Com o uso do método post, as informações ficam invisíveis ao usuário, isso evita que alguma causa mal-intencionada venha tornar os dados inconsistentes( dados sem fundamentos ou falsos).

## 6.0 Estruturas de controle e repetição

### Objetivos

*Mostra estruturas de controle e sua aplicação prática em PHP; Definir qual a principal finalidade dessas estruturas; Mostrar exemplos em sua sintaxe; Mostrar aplicação e uso de foreach.*

As estruturas que veremos a seguir são comuns para as linguagens de programação imperativas, bastando descrever a sintaxe de cada uma delas resumindo o funcionamento. Independente do PHP, boa parte das outras linguagens de programação tem estruturas iguais, mudando apenas algumas sintaxes.

### 6.1 Blocos de controle

Um bloco consiste de vários comandos agrupados com o objetivo de relacioná-los com determinado comando ou função. Em comandos como if, for, while, switch e em declarações de funções blocos podem ser utilizados para permitir que um comando faça parte do contexto desejado. Blocos em PHP são delimitados pelos caracteres “{” e “}”. A utilização dos delimitadores de bloco em uma parte qualquer do código não relacionada com os comandos citados ou funções não produzirá efeito algum, e será tratada normalmente pelo interpretador. Outro detalhe importante: usar as estruturas de controle sem blocos delimitadores faz com que somente o próximo comando venha ter ligação com a estrutura. Observe os exemplos:

```
1 <?php
2
3 $a = 0;
4 if($a>2)
5     echo "comando1";
6     echo "comando2";
7
8 ?>
```

Observe que temos um comando IF, onde é passado a ele uma expressão booleana que retorna verdadeiro ou falso.

O resultado da expressão é FALSE(falso), pois 0 não é maior que 2, fazendo com que o IF não execute o **echo** com “comando1”. Somente o segundo **echo** é executado, pois não pertence ao IF declarado.

Mas se quisermos que mais de um comando pertença a estrutura de controle, será usado blocos de comandos

( { comando; } ), onde através deles podemos delimitar e organizar os códigos.

```
1 <?php
2
3 $a = 0;
4 if($a>2){ // início do bloco de código.
5     echo "comando1";
6     echo "comando2";
7 } // fim do bloco de código.
8 ?>
```

No código ao acima, temos um bloco onde inserimos dois comandos. Observe que eles não serão

executados, pois a expressão booleana passada para o IF é falsa.

## 6.2 If e else

Essa estrutura condicional está entre as mais usadas na programação. Sua finalidade é induzir um desvio condicional, ou seja, um desvio na execução natural do programa. Caso a condição dada pela expressão seja satisfeita, então serão executadas as instruções do bloco de comando. Caso a condição não seja satisfeita, o bloco de comando será simplesmente ignorado. Em lógica de programação é o que usamos como “SE (expressão) ENTÃO

{comando;}”.

Sintaxe:

```
if (expressão)
```

```
comando; if (expressão){
comando1;
```

```
comando2; comando3;
```

```
}
```

exemplo:

```
1 <?php
2
3 $idade = 16;
4 if($idade > 18 ){
5     $texto = "maior idade";
6     echo $texto;
7 }
8 ?>
```

Caso a condição não seja satisfatória (FALSE), podemos atribuir outro comando pertencente ao IF chamado ELSE, como se fosse a estrutura SENÃO em lógica de programação.

Sintaxe:

```
if (expressão)
comando;
eles

comando;

if (expressão){
comando1;
comando2;
comando3;
} else {
comando1;
comando2;
comando3;
}
```

Exemplo:

```
1 <?php
2
3 $idade = 16;
4 if($idade > 18 ){
5     $texto = "maior idade";
6 }
7 else {
8     $texto = "menor idade";
9     echo $texto;
10 }
11 ?>
```

Nesse exemplo temos uma expressão booleana onde retorna falso, com isso o IF não executa, passando a execução para o else, que por sua vez executa e atribui o valor “menor idade” a variável \$texto.

Em determinadas situações é necessário fazer mais de um teste, e executar condicionalmente diversos comandos ou blocos de comandos. Isso é o que podemos chamar de “Ifs encadeados”, onde usamos a estrutura **IFELSE**. Para facilitar o entendimento de uma estrutura do tipo:

Sintaxe:

```
if (expressao1)
comando1;

else
if (expressao2)

comando2;
else
if (expressao3)
comando3;
else
```

Exemplo:

```
1 <?php
2
3 $idade = 16;
4 if($idade > 18 ){
5     $texto = "maior de 18 anos";
6 }elseif($idade < 12){
7     $texto = "Idade de Criança";
8 }else{
9     $texto = "Idade de Adolescente";
10 }
11 echo $texto;
12
13 ?>
```

comando4;

### Exercício rápido:

1º) Faça uma script em PHP que possua 4 notas de um aluno (cada uma em uma variável). Depois calcule e imprima a média aritmética das notas e a mensagem de aprovado para média superior ou igual a 7.0 ou a mensagem de reprovado para média inferior a 7.0.

2º) Faça uma script em PHP que receba a idade de um nadador (representada por uma variável chamada “\$idade”) e imprima a sua categoria seguindo as regras:

Categoria	Idade
Infantil A	5 - 7 anos
Infantil B	8 - 10 anos
Juvenil A	11- 13 anos
Juvenil B	14- 17 anos
Sênior	maiores de 18 anos

## 6.3 Atribuição condicional (ternário)

Como já vimos exemplos de atribuição condicionais (ternários), podemos defini-los usando a sintaxe:

*(expressão booleana) ? (executa caso verdadeiro) : (executa caso falso);*

Isso se aplica quando queremos uma estrutura resumida, onde podemos ter um resultado mais direto, como por exemplo, atribuir um valor a uma variável dependendo de uma expressão. Observe o exemplo abaixo onde envolvemos uma variável do tipo string, porém o valor

```
1 <?php
2
3 $idade =16;
4 $texto = $idade > 18 ? "maior idade" : "menor idade";
5 echo $texto;
6
7 ?>
```

atribuído a essa variável deverá ser de acordo com o valor da idade:

É uma estrutura parecida com IF e ELSE, onde dependendo da expressão booleana podemos executar um bloco ou não.

#### Exercício rápido:

1º) Faça uma script em PHP que receba um número representado por uma variável. Verifique se este número é par ou ímpar e imprima a mensagem.

2º) Crie outro script baseando em um seguro de vida com as seguintes regras:

**Idade:**      **Grupo de Risco :**

18 a 24      - Baixo

25 a 40      - Médio

41 a 70      - Alto

## 6.4 Switch

Observe que quando temos muitos “if’s encadeados” estamos criando uma estrutura que não é considerada uma boa prática de programação. Para resolver esse problema temos uma estrutura onde sua funcionalidade é semelhante ao **IFELSE**. O comando **SWITCH** é uma estrutura que simula uma bateria de teste sobre uma variável. Frequentemente é necessário comparar a mesma variável com valores diferentes e executar uma ação específica em cada um desses valores.

Sintaxe:

```
switch(expressão)
```

```
{
    case "valor 1":
        comandos;
    case "valor 1":
        comandos;
    case "valor 1":
        comandos;
    case "valor 1":
        comandos;
    ...
}
```

Exemplo:

```
1 <?php
2 $numero = 2;
3 switch ($numero){
4     case 1:
5         echo "opção 1:";
6     case 2:
7         echo "opção 2:";
8     case 3:
9         echo "opção 3:";
10    case 4:
11        echo "opção 4:";
12    case 5:
13        echo "opção 5:";
14    }
15 }
16 ?>
```

Resultado: opção 2:opção 3:opção 4:opção 5:

Nesse exemplo temos o número = 2, onde o switch compara com os case's o valor recebido, o bloco que é executado é do segundo case, porém os demais também são executados para que tenhamos um resultado satisfatório temos que usar em cada case um comando chamado **break**. No qual tem a função de para o bloco de execução.

### 6.4.1 Switch com break

Break é uma instrução (comando) passada quando queremos parar o fluxo da execução de um programa. Em PHP, ele tem a mesma função que é “abortar” o bloco de código correspondente.

Observe o mesmo exemplo com o uso de break:

Obs.: Além de números podemos também comparar outros tipos como string, pontos flutuantes e inteiros, veja um exemplo abaixo:

```
1 <?php
2 $numero = 2;
3 switch ($numero){
4     case 1:
5         echo "opção 1:";
6         break;
7     case 2:
8         echo "opção 2:";
9         break;
10    case 3:
11        echo "opção 3:";
12        break;
13    case 4:
14        echo "opção 4:";
15        break;
16    case 5:
17        echo "opção 5:";
18        break;
19 }
20 ?>
```

```
1 <?php
2 $numero = "opc 2";
3 switch ($numero){
4     case "opc 1":
5         echo "opção 1:";
6         break;
7     case "opc 2":
8         echo "opção 2:";
9         break;
10 }
11 }
12 ?>
```

Temos agora como resultado “opção 2:”. O comando break fez com que os demais case's abaixo do 'case 2' não sejam executados.

Mas o que acontece se não tivermos um valor que seja satisfatório aos casos existentes no switch? A resposta é bem simples, nenhum dos blocos seria executados, porém temos um comando onde determinamos uma opção padrão caso nenhuma das outras venha ter resultado que satisfaça a expressão passada para o switch chamada default (padrão).

Veja um exemplo:

```

1 <?php
2 $numero = "opc 3";
3 switch ($numero){
4     case "opc 1":
5         echo "opção 1:";
6         break;
7     case "opc 2":
8         echo "opção 2:";
9         break;
10    default:
11        echo "opção inválida";
12    }
13    ?>
14

```

Resultado: opção inválida

A instrução passada não condiz com nenhum dos caso existentes. Por esse motivo o bloco

O comando default pode ser inserido em qualquer lugar dentro do switch, porém caso isso aconteça, o uso do comando break deve ser adicionado para evitar que os case's abaixo sejam executados.

pertencente ao comando default será executado.

A partir de agora trabalharemos as estruturas de repetição. Elas muito utilizadas nas linguagens de programação.

#### Exercício rápido:

- 1º) Faça um script em PHP usando switch, onde receba uma variável e mostre as seguintes opções: 1 - módulo.  
 2 - somar.  
 3 - subtrair.  
 4 - multiplicar.  
 5 - dividir.

## 6.5 While

O WHILE é uma estrutura de controle similar ao IF, onde possui uma condição para executar um bloco de comandos. A diferença primordial é que o WHILE estabelece um laço de repetição, ou seja, o bloco de comandos será executado repetitivamente enquanto a condição passada for verdadeira. Esse comando pode ser interpretado como “ENQUANTO (expressão) FAÇA { comandos...}”.

Sintaxe:

```

while (expressão){
    comandos;
}

```

Quando estamos usando um laço de repetição, podemos determinar quantas vezes ele deve ou não se repetir. Isso pode ser feito de forma manual, onde o programador determina, ou automaticamente, onde quem vai determinar é fluxo de execução o código- fonte através de funções do PHP, funções estas já existentes. Por exemplo a função “sizeof”.

Para trabalharmos com essa contagem de quantas vezes o laço deve se repetir, usaremos incremento ou decremento de uma variável conforme vimos no capítulo de operadores em PHP.

```

1 <?php
2 $a = 1; // iniciamos uma variável para contagem.
3 while($a < 10){
4     print $a;
5     $a++; // incrementamos a variável.
6 }
7 ?>
8

```

Observe o exemplo abaixo:

Resultado: 123456789

Nesse exemplo criamos um laço de repetição que tem como condição \$a < 10, a cada laço é executado um incremento na variável \$a, fazendo com que o seu valor aumente até a condição não ser mais satisfatória.



Dicas: Tenha cuidado quando estiver trabalhando com loop's (laço de repetição), pois caso a expressão passada esteja errada, pode ocasionar em um loop infinito fazendo com que o bloco de código se repita infinitamente. Isso pode ocasionar um travamento do navegador ou até mesmo do próprio servidor WEB.

Vamos ver agora um exemplo em que o laço se repete de forma automática, onde quem determina o loop é uma função do PHP e não um numero determinado pelo programador.

A função **strlen()** recebe uma string e retorna a quantidade de caracteres incluindo também os espaços em branco. Ele poderia ser aplicado diretamente no **echo**, mas no exemplo, ele determina a quantidade de loop's.

```

1 <?php
2 $a = 1;
3 $b = "Projeto ejovem";
4 while($a < strlen($b)) // conta o tamanho da string.
5     $a++;
6
7 echo "a frase possui ".$a." caracteres";
8 ?>
9

```

Resultado: a frase possui 14 caracteres



**Exercício rápido:**

- 1º) Faça um script que conte de 1 até 100.
- 2º) Faça um script que imprima na tela números de 3 em 3 iniciando com 0 até 90, ex: 0,3,6,9...

**6.6 Do...while**

O laço do...while funciona de maneira bastante semelhante ao while, com a simples diferença que a expressão é testada ao final do bloco de comandos. O laço do...while possui apenas uma sintaxe que é a seguinte:

**Sintaxe:**

```
do {
    comando;
    . . .
    comando;
} while (expressão);
```

**Exemplo:**

```
1 <?php
2
3 $a = 1;
4 do{
5
6     echo $a;
7     $a++; // incrementa +1 a cada laço.
8
9 }while($a < 10)
10
11 ?>
```

Resultado: 123456789



Dicas: Talvez na criação de alguma página ou sistema web, seja necessário executar um bloco de código existente em um laço de repetição pelo menos uma vez, nesse caso podemos usar o do...while.

**Exercício rápido:**

- 1º) Faça um script que conte de -1 até -100 usando “do while”.
- 2º) Faça um script que imprima na tela somente números pares de 2 até 20 com do while.

**6.7 For**

Outra estrutura semelhante ao while é o **for**, onde tem a finalidade de estabelecer um laço de repetição em um contador. Sua estrutura é controlada por um bloco de três comandos que estabelecem uma contagem, ou seja, o bloco de comandos será executado determinado número de vezes.

**Sintaxe:**

```
for( inicialização; condição; incremento ){
    comandos;
}
```

Parâmetros	Descrição
<b>inicialização</b>	Parte do for que é executado somente uma vez, usado para inicializar uma variável.
<b>condição</b>	Parte do for onde é declarada uma expressão booleana.
<b>incremento</b>	Parte do for que é executado a cada interação do laço.

Lembrando que o loop do **for** é executado enquanto a condição retornar expressão booleana verdadeira. Outro detalhe importante é que poderemos executar o incremento a cada laço, onde possibilitamos adicionar uma variável ou mais. Mostraremos agora um exemplo fazendo um comparativo entre a estrutura de repetição do while e também a do **for** de forma prática.

**Exemplo:****while**

```
1 <?php
2
3 $a = 1; //inicia a variável.
4 while($a < 10){ //condição.
5     echo $a; //comando.
6     $a++; //incremento.
7 }
8
9 ?>
```

**for**

```
1 <?php
2
3 //inicia a variável.
4 //condição
5 //incremento
6
7 for($a = 1 ; $a < 10 ; $a++){
8     echo $a;
9
10 }
11
12 ?>
```

Ambos exemplo geram o mesmo resultado: 123456789

O **for** não precisa ter necessariamente todas as expressões na sua estrutura, com isso podemos

```
1 <?php
2
3 $a = 1; //inicia a variável.
4 for( ; $a < 10 ; ){ //condição.
5     echo $a; //comando.
6     $a++; //incremento.
7 }
8
9 ?>
```

criar um exemplo de **for** onde suas expressões são declaradas externamente.

Observe nesse exemplo uma proximidade muito grande do comando while. Apesar de ser funcional, não é uma boa prática de programação utilizar desta forma.

#### Exercício rápido:

1º) Faça um script que receba duas variáveis \$a e \$b, logo após imprima os números de intervalos entre eles com o uso de “for”.ex: a=5 ,b = 11, imprime : 5,6,7,8,9,10,11.

## 6.8 Foreach

O **foreach** é um laço de repetição para interação em array's ou matrizes, o qual estudaremos com mais detalhes no próximo capítulo. Trata-se de um **for** mais simplificado que compõe um vetor ou matriz em cada um de seus elementos por meio de sua cláusula AS. **Exemplo:**

Sintaxe:

```
foreach( expressão_array as $valor){
    comandos;
}
```

```
1 <?php
2
3 $nomes = array("ana","maria","joão","alex");
4 foreach($nomes as $nome){
5     echo $nome."<br>";
6 }
7
8 ?>
```

**Resultado:** ana maria joão alex

Veremos adiante que um array é uma variável composta por vários elementos. No caso do exemplo anterior, esses elementos são nomes de pessoas. A finalidade do **foreach** é justamente a cada laço, pegar um desses valores e atribuir a uma variável \$nome até que tenha percorrido todo array e assim, finalizar o laço. Também podemos saber em qual posição o elemento se encontra no array, para isso basta adicionar uma nova variável logo após o AS seguido de =>.

Observe o exemplo:

```
1 <?php
2
3 $nomes = array("ana","maria","joão","alex");
4 foreach($nomes as $chave => $nome){
5     echo $chave." - ".$nome."<br>";
6 }
7
8 ?>
```

**Resultado:**

1. ana
2. maria 2-joão 3-alex

Nesse exemplo observamos que cada elemento do array possui um índice (chave), imprimindo na tela o numero da posição e o valor guardado.

## 6.9 Break

Outro comando importante é o **break**, usado para abortar (parar) qualquer execução de comandos como SWITCH, WHILE, FOR, FOREACH, ou qualquer outra estrutura de controle. Ao encontrar um **break** dentro de um desses laços, o interpretador PHP interrompe imediatamente a execução do laço, seguindo normalmente o fluxo do script.

Sintaxe:

```
while....
for....
    break <quantidades de níveis>;
```

Vamos ver um exemplo com o uso de **break** dentro de um laço de repetição (no caso o **for**), onde criamos um laço infinito, porém colocamos um if com a condição de parar o laço através do **break**. Observe:

```
1 <?php
2
3 $a = 1; // inicia a variável.
4 for( ; ; ){ // loop infinito.
5     echo "- ".$a; // imprime na tela.
6     if($a == 10) // condição.
7         break; // break, para o laço.
8     $a++; // incremento
9 }
10
11 ?>
```

Podemos notar nesse exemplo a criação de um laço(loop) infinito, que ocorre quando tiramos a condição do **for**, ou atribuímos “**for( ; true ; )**”, porém a condição fica na responsabilidade do if, quando o valor de \$a é igual a 10, faz com que o if execute o **break**,

fazendo com que o laço pare de funcionar.

Mas se tivéssemos mais de um laço, como poderíamos definir qual deles deixaria de funcionar? Para responder essa pergunta usamos a quantidades de níveis que pode existir em um **break**, observe o exemplo abaixo:

```
1 <?php
2
3 for($a = 1;$a < 5;$a++){ // primeiro laço
4     echo $a." - ---- laço<br>";
5
6     for($b = 1;$b < 5;$b++){ // segundo laço
7         echo $b." - segundo for<br>";
8         if($a == 2) // condição
9             break 2; // break
10     }
11 }
12
13 ?>
```

**Resultado:**

1 ° ---- laço  
1 - segundo for  
2 - segundo for  
3 - segundo for  
4 - segundo for  
2 ° ---- laço  
1 - segundo for

Observe que para definir qual nível podemos parar utilizamos o break, ou seja, o primeiro nível é onde o break está localizado, no exemplo citado temos dois níveis, e determinamos pelo “break 2;” que o segundo for( que é o de fora! ) deixaria de funcionar.

## 6.10 Continue

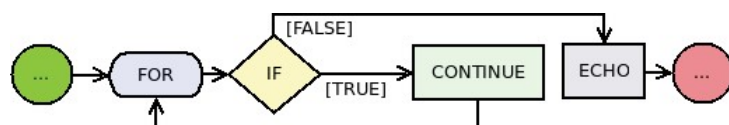
A instrução **continue**, quando executada em um bloco de comandos for/while, ignora as instruções restantes até o fechamento em “}”. Dessa forma, o programa segue para a próxima verificação da condição de entrada do laço de repetição, funciona de maneira semelhante ao break, com a diferença que o fluxo ao invés de sair do laço volta para o início dele. Veja um exemplo:

```
1 <?php
2
3 for ($i = 0; $i < 20; $i++) { //laço de repetição.
4     if ($i % 2) continue; //continue.
5     echo $i.", "; //imprime na tela.
6 }
7
8 ?>
```

Resultado: 0,2,4,6,8,10,12,14,16,18,

Podemos observar a seguinte lógica no exemplo acima:

Criamos um laço que tem 20 interações de repetição. Logo após temos um if, onde, quando o resto da divisão por 2 for igual a 0 (numero par), o valor booleano será “false”. Quando não for igual a 0, significa que a variável \$i é um numero impar( ex:  $5\%2 = 1$  ), então temos um valor booleano “true”. Isso significa que o if executa somente quando os números forem ímpares. Adicionamos um **continue**, que ao executar o if, faz com que volte novamente para o início do for, impedindo de alcançar o **echo** em seguida. Com isso, em vez de mostrarmos os números ímpares, imprimimos somente os números pares incluindo o 0. Resumimos que o código só passa adiante quando o if não executa o continue. Fluxograma:



Assim como o break, também podemos definir em qual nível queremos que a execução continue. Veja o exemplo abaixo:

```
1 <?php
2
3 $i = 0;
4 while ($i++ < 5) { //laço de 5 interações, 3º nível.
5     echo "---primeiro<br>"; //laço infinito, 2º nível.
6     while (1) { //laço infinito, 1º nível.
7         echo "-segundo<br>";
8         while (1) {
9             echo "-terceiro<br>";
10            continue 3; //continua do 3º nível.
11        }
12        echo "Esta saída não é Alcançada.<br>";
13    }
14    echo "Esta saída não é Alcançada.<br>";
15 }
16
17 ?>
```

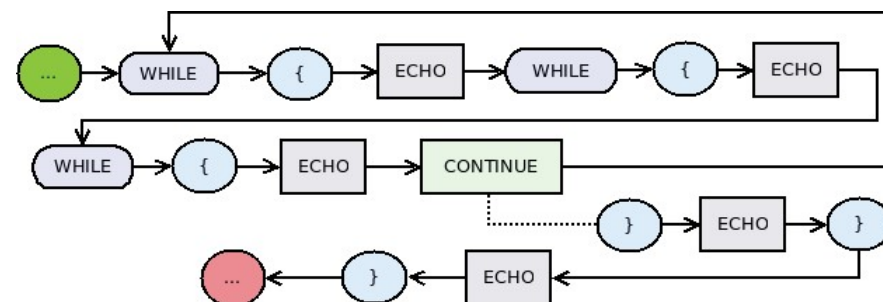
Resultado:

Podemos observar então o uso de **continue** dentro de um laço infinito. Ele faz com que o laço de nível 3 seja executado toda vez que a execução passe pela linha 10 do código, logo, impede que o programa fique sempre executando dentro do while de nível 1. Com isso, o while da linha 4 do código tem um ciclo de vida de 5 laços.

Observe também que os dois últimos echo's nunca serão alcançados, pois o comando continue impede que o fluxo do código passe adiante, fazendo voltar ao nível determinado.

**Resumindo:** O **continue** é usado dentro de estruturas de loops para saltar o resto da execução do loop atual e continuar a execução na avaliação do estado, em seguida, o início da próxima execução.

Fluxograma:



**Exercício Resolvido com switch**

Qual a mensagem que será Gerada para o usuário no final desse código?

```
<?php

$texto = "Olá";
$op = 2;

switch($op){
    case 1:
        $texto .=", crianças";
        break;
    case 2:
        $texto .=", senhores";
    case 3:
        $texto .=", senhoras";
        break;
    default:
        $texto .= ", idosos";
        break;
}

$texto .= " e Jovens de todo o Ceará";

echo $texto;

?>
```

**Entendendo o algoritmo:**

- No escopo temos uma variável com conteúdo de \$texto = “Olá”
- No escopo temos uma variável \$op = 2.
- O valor da variável \$op será avaliado no switch.
- O switch redireciona para o caso 2, pois o valor da variável \$op = 2.
- Uma operação de concatenação é realizada, a variável \$texto agora apresenta : “Olá, senhores”.
- Abaixo da concatenação não existe o comando break, podendo prosseguir com o caso 3 logo abaixo.
- Uma nova operação de concatenação é feita no caso 3 com a variável \$texto, tornando-se : “Olá, senhores, senhoras”.
- Ao encontrar o break, a estrutura de seleção multipla será finalizada. realizando abaixo da estrutura uma ultima operação de concatenação : “Olá, senhores, senhoras e Jovens de todo o Ceará”
- Por fim, temos a variável \$texto sendo impressa com : **“Olá, senhores, senhoras e**

**Jovens de todo o Ceará”****Exercício Resolvido com “do while”.**

Existe algum erro nesse código? Se existe, onde estaria o erro, caso contrário qual mensagem

```
<?php

$vidas = 0;
$pontos = 0;

do{
    if($pontos == 0 ){
        $pontos++;
    }
}while($vidas>0);

echo "Você acumulou $pontos Pontos";

?>
```

será gerada para o usuário?

**Entendendo o algoritmo:**

- No escopo temos uma variável \$vidas = 0;
- No escopo temos uma variável \$pontos = 0;
- A estrutura do {} while(\$vidas>0) realiza pelo menos 1 vez o bloco de código independente da condição.
- Na estrutura de condição if(\$pontos==0) podemos identificar um condição “true” pois o valor da variável pontos é igual a 0. Assim pontos será incrementado em 1.
- A estrutura de repetição “do while” é verificada, como o valor da variável \$vidas é igual a 1, essa estrutura irá retornar falsa encerrando a estrutura de repetição.
- Por fim, a impressão de um texto que será de :

**Você acumulou 1 Pontos.**

Obs.: Perceba que nesse algoritmo não diferencia a palavra “PONTOS” e “PONTO” de acordo com o número de vidas..

Pergunta: Como melhora ríamos esse código para caso eu tenha apenas 1 vida, eu possa imprimir:

**Você acumulou 1 Ponto.**



**EXERCÍCIOS PROPOSTOS**

Qual a principal finalidade de uma estrutura de controle?

Qual a principal finalidade de uma estrutura de repetição?

Crie um código com a um condição ternária onde receba um valor booleano e de acordo com o valor passado na expressão, deve imprimir “sim” ou “não”.

Com o comando IF e ELSE crie um código que determine se uma expressão é verdadeira ou falsa.

Qual a finalidade da estrutura de controle SWITCH e cite um exemplo onde comparamos uma opção com 4 casos diferente?

Crie um contador de 1 até 20 usando a estrutura de repetição WHILE.

Crie um contador de 1 até 100 usando DO WHILE. **EP06.8:** Crie um contador de 100 até 1 usando FOR. **EP06.9:** Qual a finalidade de um FOREACH?

Crie um código onde podemos para a execução de um laço infinito com o uso de BREAK.

Como podemos determinar o uso de CONTINUE e qual a sua aplicação prática em PHP.

Crie um código com as seguintes características:

- a) Deverá receber um valor inicial e outro final (crie duas variáveis para esse fim).
- b) Como o comando FOR crie um laço onde a contagem é determinada pelo valor inicial e final?
- c) Dentro do for deverá conter um IF e ELSE responsável por compara os valores passado a ele e imprimir os pares e ímpares. Exemplo:  
IF(\$valor%2==0)  
  
echo \$valor. “é um numero par”; ELSE  
  
echo \$valor. “é um numero impar”;
- d) Exemplo prático: foi passado o numero inicial 8 e o final 15, então o script PHP deverá imprimir o intervalo entre esse numero ou seja 8,9,10,11,12,13,14,15, mostrando quais deles são pares e quais são ímpares.