



# PHP

Professor Claudio Fico  
E-mail: [cfico@globo.com](mailto:cfico@globo.com)

# Introdução

# PHP

## O que é PHP?

O PHP é uma linguagem que permite criar sites WEB dinâmicos, possibilitando uma interação com o usuário através de formulários, parâmetros da URL e links. A diferença de PHP com relação a linguagens semelhantes a Javascript é que o código PHP é executado no servidor, sendo enviado para o cliente apenas html puro.

Desta maneira é possível interagir com bancos de dados e aplicações existentes no servidor, com a vantagem de não expor o código fonte para o cliente. Isso pode ser útil quando o programa está lidando com senhas ou qualquer tipo de informação confidencial.

O que diferencia PHP de um script CGI escrito em C ou Perl é que o código PHP fica embutido no próprio HTML, enquanto no outro caso é necessário que o script CGI gere todo o código HTML, ou leia de um outro arquivo.

# PHP

## Como surgiu a linguagem PHP?

A linguagem PHP foi concebida durante o outono de 1994 por Rasmus Lerdorf. As primeiras versões não foram disponibilizadas, tendo sido utilizadas em sua home-page apenas para que ele pudesse ter informações sobre as visitas que estavam sendo feitas.

A primeira versão utilizada por outras pessoas foi disponibilizada em 1995, e ficou conhecida como “Personal Home Page Tools” (ferramentas para página pessoal). Era composta por um sistema bastante simples que interpretava algumas macros e alguns utilitários que rodavam “por trás” das home-pages: um livro de visitas, um contador e algumas outras coisas.

# PHP

Em meados de 1995 o interpretador foi reescrito, e ganhou o nome de PHP/FI, o “FI” veio de um outro pacote escrito por Rasmus que interpretava dados de formulários HTML (Form Interpreter).

Ele combinou os scripts do pacote Personal Home Page Tools com o FI e adicionou suporte a mSQL, nascendo assim o PHP/FI, que cresceu bastante, e as pessoas passaram a contribuir com o projeto.

Estima-se que em 1996, PHP/FI estava sendo usado por cerca de 15.000 sites pelo mundo, e em meados de 1997 esse número subiu para mais de 50.000. Nessa época houve uma mudança no desenvolvimento do PHP.

Ele deixou de ser um projeto de Rasmus com contribuições de outras pessoas para ter uma equipe de desenvolvimento mais organizada. O interpretador foi reescrito por Zeev Suraski e Andi Gutmans. Esse novo interpretador foi à base para a versão 3.

# PHP

O lançamento do PHP4, ocorrido em 22/05/2000, trouxe muitas novidades aos programadores de PHP.

Uma das principais foi o suporte a sessões, bastante útil pra identificar o cliente que solicitou determinada informação.

Além das mudanças referentes a sintaxe e novos recursos de programação, o PHP4 trouxe como novidade um otimizador chamado Zend, que permite a execução muito mais rápida de scripts PHP.

A empresa que produz o Zend promete para este ano o lançamento de um compilador de PHP. Códigos compilados serão executados mais rapidamente, além de proteger o fonte da aplicação. Hoje, já se encontra disponível no mercado a versão 7.x do PHP.



# PHP

## Características da Linguagem PHP

- 1 - É uma linguagem de fácil aprendizado;
- 2 - Tem suporte a um grande número de bancos de dados como: dBase, Interbase, mSQL, MySQL, Oracle, Sybase, PostgreSQL e vários outros.
- 3 - Tem suporte a outros serviços através de protocolos como IMAP, SNMP, NNTP, POP3 e, logicamente, HTTP;
- 4 - É multiplataforma, tendo suporte aos sistemas Operacionais mais utilizados no mercado;
- 5 - Seu código é livre, não é preciso pagar por sua utilização e pode ser alterado pelo usuário na medida da necessidade de cada usuário
- 6 - Não precisa ser compilado.

# Sintaxe



# PHP

O código PHP fica embutido no próprio HTML. O interpretador identifica quando um código é PHP pelas seguintes tags:

```
<?php  
    código / instruções.....  
?>
```

```
<script language="php">  
    comandos;  
</script>
```

# PHP

## Locais de uso do PHP

```
<?php
    // Sim também podemos ter código PHP antes do DocType.
?>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>Titulo</title>
        <?php
            // Sim também podemos ter código PHP no <head>.
        ?>
    </head>
    <body>
        <?php
            // Sim também podemos ter código PHP no <body>.
        ?>
    </body>
</html>
<?php
    // Sim também podemos ter código PHP depois do fechamento da tag <html>.
?>
```

# PHP

## Separador de instruções

Entre cada instrução em PHP é preciso utilizar o **ponto-e-vírgula (;)**, assim como em C, Perl e outras linguagens mais conhecidas. Na última instrução do bloco de script não é necessário o uso do ponto-e-vírgula, mas por questões estéticas recomenda-se o uso sempre.

## Comentários

Há dois tipos de comentários em código PHP:

### Comentários de uma linha:

Marca como comentário até o final da linha ou até o final do bloco de código PHP – o que vier antes. Pode ser delimitado pelo caractere “**#**” ou por duas barras (**//**). O delimitador “//”, normalmente, é o mais utilizado.



Enviando informações  
ao browser

# PHP

## Imprimindo código html

Um script php geralmente tem como resultado uma página html, ou algum outro texto. Para gerar esse resultado, deve ser utilizada uma das funções de impressão, **echo e print**.

### Sintaxes:

**echo** (argumento1, argumento2, ... );

**echo** argumento;

ou

**print** (argumento);

# PHP

## Exemplo:

```
<?php
```

```
    echo "teste"; // este teste é similar ao anterior
```

```
    echo "teste"; # este teste é similar ao anterior
```

```
    // sql "teste";
```

```
?>
```



# PHP

## Exemplos:

```
<?php
$a = 10;
$b = 20;
$soma = $a + $b; #Imprimindo a soma na tela com o comando echo.
print ("A soma é: ". $soma); //usando o print para exibir as informações
echo "<br>"; //quebrando uma linha
echo ("A soma é: ". $soma); //usando o print para exibir as
informações
?>
```

# PHP

## Comentários de mais de uma linha:

Tem como delimitadores os caracteres “/\*” **para o início do bloco** e “\*/” **para o final do comentário**. Se o delimitador de final de código PHP ( ?> ) estiver dentro de um comentário, não será reconhecido pelo interpretador.

## Exemplo:

```
<?
    echo “teste”;
    /* este é um comentário com mais de uma linha.
    echo “teste. Este comando echo é ignorado pelo interpretador do
    PHP por ser um comentário.”*/
?>
```

# Variáveis

# PHP

## Nomes de variáveis

Toda variável em PHP tem seu nome composto pelo caracter **\$(dólar)** e uma string, que deve iniciar por uma letra ou o caracter “\_”. O PHP é **case sensitive**, ou seja, as variáveis \$vivas e \$VIVAS são diferentes.

Por isso é preciso ter muito cuidado ao definir os nomes das variáveis. É bom evitar os nomes em maiúsculas, pois como veremos mais adiante, o PHP já possui algumas variáveis pré-definidas cujos nomes são formados por letras maiúsculas.

## Exemplo de variáveis:

- 1 - \$teste = nome correto de variável
- 2 - teste = nome errado de variável.

# PHP

## Tipos Suportados

O PHP suporta os seguintes tipos de dados:

- 1 - Inteiro
- 2 - Ponto flutuante
- 3 - String
- 4 - Array
- 5 - Objeto
- 6 - Boleano;

O PHP utiliza checagem de tipos dinâmica, ou seja, uma variável pode conter valores de diferentes tipos em diferentes momentos da execução do script. Por este motivo não é necessário declarar o tipo de uma variável para usá-la. O interpretador PHP decidirá qual o tipo daquela variável, verificando o conteúdo em tempo de execução. Ainda assim, é permitido converter os valores de um tipo para outro desejado, utilizando o ***typecasting*** ou a função ***settype***

# PHP

## Inteiros (integer ou long)

Uma variável pode conter um valor inteiro com atribuições que sigam as seguintes sintaxes:

1 - `$num = 1234;` # inteiro positivo na base decimal

2 - `$num = -234;` # inteiro negativo na base decimal

3 - `$num = 0234;` # inteiro na base octal-simbolizado pelo 0 equivale a 156 decimal

4 - `$num = 0x34;` # inteiro na base hexadecimal(simbolizado pelo 0x) – equivale a 52 decimal.

A diferença entre **inteiros simples** e **long** está no número de bytes utilizados para armazenar a variável.

Como a escolha é feita pelo interpretador PHP de maneira transparente para o usuário, podemos afirmar que os tipos são iguais.



# PHP

## Ponto Flutuante (double ou float)

Uma variável pode ter um valor em ponto flutuante com atribuições que sigam as seguintes sintaxes:

1 - `$num = 1.234;`

2 - `$num = 23e4;` # equivale a 230.000

# PHP

## Strings

Strings podem ser atribuídas de duas maneiras:

**a) utilizando aspas simples ( ' )** → Desta maneira, o valor da variável será exatamente o texto contido entre as aspas (com exceção de `\\` e `\'` – ver exemplo abaixo)

**b) utilizando aspas duplas ( " )** → Desta maneira, qualquer variável ou caractere de escape será expandido antes de ser atribuído.

# PHP

## Exemplo:

```
<?php
```

```
$texto = "Curso de PHP";
```

```
$teste = "Mauricio";
```

```
$vivas = '---$teste--\n';
```

```
echo "$vivas"; # A saída desse script será "---$teste--\n".
```

```
?>
```

```
<?php
```

```
$texto = "Curso de PHP";
```

```
$teste = "Mauricio";
```

```
$vivas = "---$teste--\n";
```

```
echo "$vivas"; # A saída desse script será "Mauricio".
```

```
?>
```

# PHP

## Exemplo:

```
<?php
    $titulo = 'Título da minha página';
    $css = '<link rel="stylesheet" type="text/css" href="css/estilos.css" />';
    $conteudo = 'Sejam bem-vindos ao mundo do PHP <br> Nele poderemos ter muitas oportunidades <br><br> <strong>Vamos em
frente!</strong>';
?>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title><?php echo $titulo; ?></title>
        <?php
            echo $css;
        ?>
    </head>
    <body>
        <?php
            echo $conteudo;
        ?>
        <p>Eu posso repetir o valor da variável sempre que eu quiser sabia? Veja aqui o nosso título aparecendo "<?php echo $titulo; ?>"</p>
        <p>E não necessariamente deveria imprimir <b>$titulo</b> apenas porque eu a declarei primeiro poderia imprimir <b>$css</b> antes
        e depois <b>$titulo</b> ou qualquer uma variável isso vai de acordo com sua necessidade</p>
        <br>
        <p>Agora irei exibir o <b>$conteudo</b> novamente, olhe ela ai:</p>
        <p><?php echo $conteudo; ?></p>
    </body>
</html>
```

## Exercício:

### Criar um arquivo com extensão php

- 1 - Criar antes do DOCTYPE três variáveis e em cada uma das variáveis colocar um determinado conteúdo. Uma das variáveis deverá ser a do título;
- 2 - Na tag <title> colocar o conteúdo da variável título;
- 3 - Na tag <body> colocar em cada parágrafo a ser criado por você o conteúdo da variável. Lembrando que como são duas variáveis deverão existir dois parágrafos;
- 4 - A variável deverá estar dentro do texto a ser criado por você.

# PHP

## Exemplo: exibindo a idade de uma pessoa

```
<?php
    $idade = 30;
    $nome = "Clara Machado de Assis"
?>
```

```
<!DOCTYPE html>
<html lang="pt-br">
    <head>
        <meta charset="utf-8">
        <title>Teste</title>
    </head>
    <body>
        <strong>Nome: </strong>
        <?php
            echo $nome;
        ?>
        <br>
        <strong>Idade: </strong>
        <?php
            echo $idade;
        ?>
    </body>
</html>
```



# PHP

## Valores com tipos diferentes em variável no PHP

Obtém o valor de retorno da variável.

```
<?php
```

```
$nome = "Elton Fonseca";  
echo gettype($nome); // string  
echo "<br>";  
$nome = 28;  
echo gettype($nome); // integer  
echo "<br>";  
$nome = 78.500;  
echo gettype($nome); // float  
echo "<br>";  
$nome = true;  
echo gettype($nome); //boolean
```

```
?>
```

# PHP

## Constante

Como você já deve ter imaginado as constantes no PHP guardam valores que nunca serão alterados. Diferente das variáveis que possuem valores que podem ser alterados, sendo assim após definida uma constante ela não pode ser alterada ou removida.

Para definirmos uma constante utilizamos o comando **define()**;

### Sintaxe:

```
define( 'NOME_DA_CONSTANTE', 'VALOR DA CONSTANTE' );
```

O nome de uma constante tem a mesma regra de qualquer identificador PHP, ou seja, as mesmas regras de nomes de variáveis exceto pelo fato de constantes **não** iniciarem o nome com **cifrão (\$)**.

# PHP

## Exemplo:

```
<?php
$titulo = 'Utilizando Constantes';
$nome = 'Sandro';
$nascimento = '07/09/1979';
$sobre_nome = 'Nobrega Tavares';
define( 'ESTADO', 'Rio de Janeiro' )
?>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title><?php echo $titulo; ?></title>
  </head>
  <body>
    <p><?php echo $nome; ?> <?php echo $sobre_nome; ?>, nascido em <?php echo $nascimento;
    ?>, nasceu no <?php echo ESTADO; ?></p>
  </body>
</html>
```

# PHP

## Transformação explícita de tipos

A sintaxe do **typecast** de PHP é semelhante ao C: basta escrever o tipo entre parênteses antes do valor.

Os tipos permitidos na **Transformação explícita** são:

- 1 - (int), (integer) → muda para integer;
- 2 - (real), (double), (float) → muda para float;
- 3 - (string) → muda para string;
- 4 - (array) → muda para array;
- 5 - (object) → muda para objeto.

# PHP

## Exemplo:

<?php

```
$vivas = 15; // $vivas é integer (15)
```

```
$vivas = (double) $vivas; // $vivas é double (15.0)
```

```
$vivas = 3.9; // $vivas é double (3.9)
```

```
$vivas = (int) $vivas; // $vivas é integer (3)
```

```
// o valor decimal é truncado
```

```
print ("O valor de vivas é: $vivas");
```

?>

# PHP

## Função settype

A função settype converte uma variável para o tipo especificado, que pode ser "integer", "double", "string", "array" ou "object".

### Sintaxe:

settype(nomedavariável, "novo tipo da variável");  
O novo tipo da variável deve estar entre aspas.

### Exemplo:

```
<?php
    $vivas = 15; // $vivas é integer
    settype($vivas,"double"); // $vivas é double
    $vivas = 15; // $vivas é integer
    settype($vivas,"string"); // $vivas é string
    echo "O valor de vivas é: $vivas";
?>
```



# Arrays e Listas

# PHP

## Arrays

Arrays em PHP podem ser observados como mapeamentos ou como vetores indexados. Mais precisamente, um valor do tipo array é um dicionário onde os índices são as chaves de acesso. O índice terá seu **valor inicial 0**.

Vale ressaltar que os índices podem ser valores de qualquer tipo e não somente inteiros.

Inclusive, se os índices forem todos inteiros, estes não precisam formar um intervalo contínuo como a checagem de tipos em PHP é dinâmica, valores de tipos diferentes podem ser usados como índices de array, assim como os valores mapeados também podem ser de diversos tipos.

# PHP

## Exemplo:

```
<?php
$cor[0] = "amarelo";
$cor[1] = "vermelho";
$cor[2] = "verde";
$cor[3] = "azul";
$cor[4] = "anil";
echo $cor[4];
?>
```

## Equivalentemente, pode-se escrever:

```
<?php
$cor = array(0 => "amarelo", 1 => "vermelho", 2 => "verde", 3 => "azul",
4 => "anil");
echo $cor[2];
?>
```

# PHP

## Listas

As listas são utilizadas em PHP para realizar atribuições múltiplas. Através de listas é possível atribuir valores que estão num array para variáveis.

### Exemplo:

```
<?php
```

```
list($a, $b, $c) = array("a", "b", "c");  
echo "$a<br>";  
echo "$b<br>";  
echo "$c<br>";
```

```
  
$arr = array(0 => "zero", 1=>"um", 2=>"dois", 3=>"tres");  
list($a, $b, $c, $d) = $arr;  
echo "$a<br>";  
echo "$b<br>";  
echo "$c<br>";  
echo "$d<br>";
```

```
?>
```

# PHP

## Listas

As listas são utilizadas em PHP para realizar atribuições múltiplas. Através de listas é possível atribuir valores que estão num array para variáveis.

### Exemplo:

```
list($a, $b, $c) = array(" a ", " b ", " c ");
```

O comando acima atribui valores às três variáveis simultaneamente. É bom notar que só são atribuídos às variáveis da lista os elementos do array que possuem índices inteiros e não negativos. No exemplo acima as três atribuições foram bem sucedidas porque ao inicializar um array sem especificar os índices eles passam a ser inteiros, a partir do zero.

# Operadores

## Operadores Aritméticos

Só podem ser utilizados quando os operandos são números (integer ou float). Se forem de outro tipo, terão seus valores convertidos antes da realização da operação.

São eles:

Operador	Descrição
+	Adição
-	Subtração
/	Divisão
*	Multiplicação
%	Resto da Divisão



# PHP

## Exemplo:

```
<?php
    // Declarando os valores das variáveis
    $a = 4;
    $b = 2;
?>
```

```
<h2>Adição</h2>
<p>
    <?php
        echo $a + $b;
    ?>
</p>
```

```
<h2>Subtração</h2>
<p>
    <?php
        echo $a - $b;
    ?>
</p>
```

# PHP

## Exemplo:

```
<?php
    // Declarando os valores das variáveis
    $a = 4;
    $b = 2;
```

```
?>
```

```
<h2>Multiplicação</h2>
```

```
<p>
```

```
<?php
    echo $a * $b;
```

```
?>
```

```
</p>
```

```
<h2>Divisão</h2>
```

```
<p>
```

```
<?php
    echo $a / $b;
```

```
?>
```

```
</p>
```

# PHP

## Exemplo:

```
<?php
    // Declarando os valores das variáveis
    $a = 4;
    $b = 2;
?>
```

```
<h2>Módulo(resto da divisão)</h2>
```

```
<p>
```

```
<?php
    echo $a % $b;
?>
```

```
</p>
```

# PHP

## Operadores de Strings

Só há um operador exclusivo para strings, que é a **concatenação**

### Exemplo:

```
<?php
    $nome = "Clara";
    $sobrenome = "Nascimento";
    echo $nome. " ". $sobrenome;
?>
```

A saída do script acima será: Clara Nascimento

A colocação das aspas entre as variáveis é para proporcionar o espaço entre as palavras.

## Operadores de atribuição

Existe um operador básico de atribuição e diversos derivados. Sempre retornam o valor atribuído. No caso dos operadores derivados de atribuição, a operação é feita entre os dois operandos, sendo atribuído o resultado para o primeiro. A atribuição é sempre por valor, e não por referência.

Operador	Função	Exemplo	Equivalente a
<code>+=</code>	Atribuição e adição	<code>\$a += \$b</code>	<code>\$a = \$a + \$b</code>
<code>-=</code>	Atribuição e Subtração	<code>\$a -= \$b</code>	<code>\$a = \$a - \$b</code>
<code>*=</code>	Atribuição e Multiplicação	<code>\$a *= \$b</code>	<code>\$a = \$a * \$b</code>
<code>/=</code>	Atribuição e Divisão	<code>\$a /= \$b</code>	<code>\$a = \$a / \$b</code>
<code>%=</code>	Atribuição e Módulo	<code>\$a %= \$b</code>	<code>\$a = \$a % \$b</code>
<code>.=</code>	Atribuição e concatenação	<code>\$a .= \$b</code>	<code>\$a = \$a . \$b</code>

# PHP

## Exemplo com adição:

```
<?php
    $a = 7;
    $a += 2; // $a passa a conter o valor 9
?>
```

## Exemplo com subtração:

```
<?php
    $a = 9;
    $a -= 3; // $a passa a conter o valor 6
?>
```

## Operadores bitwise

Comparam dois números bit a bit.

Operador	Descrição
&	“e” lógico
	“ou” lógico
^	ou exclusivo
~	não (inversão)
<<	shift left
>>	shift right



# PHP

## Operadores bitwise & (and)

Compara dois valores utilizando suas representações binárias, retornando um novo valor, para formar esse valor de retorno cada bit é comparado, retornando 1 ( true ) quando ambos os bits forem iguais a **1 ( true )**, caso contrário retorna 0 ( false ).

Tabela [ASCII](#) para conferir os valores dos bits referentes aos números inteiros.

O **var\_dump()** mostra o conteúdo de uma variável. Se ele for um array é mostrado todo o array estruturado, se for uma variável simples é mostrado o valor dela, se for um objeto então é mostrado todos os campos e assim por diante. O var\_dump é muito útil para depuração de código.

# PHP

## Exemplo:

```
<?php
```

```
$a = 5; // 00000101
```

```
$b = 1; // 00000001
```

```
$c = $a & $b; // 00000101 & 00000001 = 00000001
```

```
var_dump($c); // 00000001 equivale ao valor inteiro 1 na tabela ASC
```

```
?>
```

```
00000101 &
```

```
00000001 =
```

```
00000001 Resultado
```

# PHP

## Exemplo:

```
<?php
```

```
$a = 5; // 00000101
```

```
$b = 3; // 00000011
```

```
$c = $a | $b; // 00000101 | 00000011 = 00000111
```

```
var_dump($c); // 00000111 equivale ao valor inteiro 7 na tabela ASC
```

```
?>
```

```
00000101 |
```

```
00000011 =
```

```
00000111 Resultado
```

## Operadores lógicos

Utilizados para inteiros representando valores. São muito utilizados nas estruturas de controle e repetição.

Operador	Descrição
And	“e” lógico
Or	“ou” lógico
Xor	ou exclusivo
!	não (inversão)
&&	“e” lógico
	“ou” lógico

Existem dois operadores para “e” e para “ou” porque eles têm diferentes posições na ordem de precedência.

## Operadores de comparação

As comparações são feitas entre os valores contidos nas variáveis, e não as referências. Sempre retornam um valor booleano. São muito utilizados nas estruturas de controle e repetição.

Operador	Descrição
==	igual a
!=	diferente de
<	menor que
>	maior que
<=	menor ou igual a
>=	maior ou igual a

# PHP

## Operadores de Expressão condicional

Existe um operador de seleção que é **ternário**. Funciona assim:

**(expressao1)?(expressao2):(expressao3)**

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

Exemplo:

```
<?php
    $a = 4;
    $b = 2;
    echo $a/$b == 2 ? "O resultado da divisão é 2" : "O resultado da
    divisão não é 2";
?>
```

## Operadores de incremento e decremento

++ incremento; e  
-- decremento.

Podem ser utilizados de duas formas: antes ou depois da variável. Quando utilizado antes, retorna o valor da variável antes de incrementá-la ou decrementá-la. Quando utilizado depois, retorna o valor da variável já incrementado ou decrementado.



## Operadores de Pós-incremento e Pós-decremento

Operador	Nome	Função
++\$a	Pré-incremento	Incrementa \$a em um, e então retorna \$a.
\$a++	Pós-incremento	Retorna \$a, e então incrementa \$a em um.
--\$a	Pré-decremento	Decrementa \$a em um, e então retorna \$a.
\$a--	Pós-decremento	Retorna \$a, e então decrementa \$a em um.
-\$a	Inverter o sinal	Inverte o sinal de \$a e retorna \$a.

# PHP

## Exemplos:

<?php

```
$a = $b = 10; // $a e $b recebem o valor 10  
echo "$b <br>";  
$c = $a++; // $c recebe 10 e $a passa a ter 11  
echo "$c <br>";  
echo "$a <br>";  
$d = ++$b; // $d recebe 11, valor de $b já incrementado  
echo $d;
```

?>

<?php

```
$a = $b = 10; // $a e $b recebem o valor 10  
echo "$b <br>";  
$c = $a--; // $c recebe 10 e $a passa a ter 9  
echo "$c <br>";  
echo "$a <br>";  
$d = --$b; // $d recebe 9, valor de $b já incrementado  
echo $d;  
echo -$d; // inverte o sinal e $b fica negativo
```

?>

# PHP

## Estrutura de controle

As estruturas que veremos a seguir são comuns para as linguagens de programação imperativas, bastando, portanto, descrever a sintaxe de cada uma delas, resumindo o funcionamento.

## Blocos

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 case** 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.

# PHP

## Estrutura if / else / elseif

O mais trivial dos comandos condicionais é o **if**. Ele testa a condição e executa o comando indicado se o resultado for true (valor diferente de zero). Ele possui duas sintaxes:

### Sintaxes:

```
if (expressão)  
    comando;
```

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

# PHP

## Exemplo 1:

<p>o Valor da soma é: </p>

<?php

```
$a = 40;
```

```
$b = 20;
```

```
if ($a > $b)
```

```
{
```

```
    echo $a + $b;
```

```
}
```

?>

# PHP

## Exemplo 2:

```
<?php
```

```
$a = 30;
```

```
$b = 20;
```

```
if ($a >= $b) {
```

```
    $media = ($a + $b) / 2;
```

```
    $resto = $a % $b; //calcula o resto da divisão de A por B
```

```
    echo " o Valor de A é: ".$a;
```

```
    echo " o Valor de B é: ".$b;
```

```
    echo " o Valor da média é: ".$media;
```

```
    echo " O resto da divisão de A por B é: ".$resto;
```

```
}
```

```
?>
```

# PHP

## Else

O **else** é um complemento opcional para o **if**. Se utilizado, o comando será executado se a expressão retornar o valor **false** (zero). Suas duas sintaxes são:

### Sintaxes:

```
if (expressão)  
    comando;
```

```
else  
    comando;
```

```
if (expressão) {  
    comando 1;  
    comando n
```

```
else {  
    comando 1;  
    comando n  
}
```



# PHP

## Exemplos:

```
<?php
    $a = 10;
    $b = 20;
    if ($a > $b)
    {
        $maior = $a;
    }
    else
    {
        $maior = $b;
    }
?>
```

O exemplo acima coloca em **\$maior** o maior valor entre \$a e \$b.

# PHP

## Elseif

O comando **elseif** também pode ser utilizado com dois tipos de sintaxe. Em resumo, a sintaxe geral do comando **if** fica das seguintes maneiras:

### Sintaxe:

```
if (expressão 1)
```

```
{  
    comando 1;  
    comando n;  
}
```

```
elseif (expressão 2)
```

```
{  
    comando 1;  
    comando n;  
}
```

```
else  
    comando 1;
```

# PHP

## Exemplo:

```
<?php
$nota1 = 6;
$nota2 = 8;
$media = ($nota1 + $nota2) / 2;
if ($media > 7)
{
    echo "Média: ".$media."<br>";
    echo "Aluno aprovado.";
}
elseif ($media < 7)
{
    echo "Média: ".$media."<br>";
    echo "Aluno reprovado.";
}
else {
    echo "Média: ".$media."<br>";
    echo "Aluno em recuperação.";
}
?>
```

# Switch Case

Frequentemente o programador pode querer comparar uma variável com diversos valores, e executar um código diferente a depender de qual valor é igual ao da variável. Quando isso for necessário, deve-se usar o comando **switch**.

## Sintaxe:

```
switch ($valor)
{
    case "_____":
        comandos;
        break;
    case "_____":
        comandos;
        break;
    default;
        comandos;
        break;
}
```

Diferença do código entre o uso do IF e do SWITCH CASE:

## Exemplos 1: Estrutura IF

```
<?php
if ($i == 0)
    echo "i é igual a zero";
elseif ($i == 1)
    echo "i é igual a um";
elseif ($i == 2)
    echo "i é igual a dois";
?>
```

## Exemplo 2: Estrutura SWITCH

```
<?php
switch ($i)
{
    case 0:
        echo "i é igual a zero";
        break;
    case 1:
        echo "i é igual a um";
        break;
    case 2:
        echo "i é igual a dois";
        break;
}
?>
```

## Código do Exercício

<?php

```
$diasemana = "2";
```

```
switch($diasemana)
```

```
{
```

```
    case"0": $diasemana = "Domingo"; break;
```

```
    case"1": $diasemana = "Segunda Feira"; break;
```

```
    case"2": $diasemana = "Terça Feira"; break;
```

```
    case"3": $diasemana = "Quarta Feira"; break;
```

```
    case"4": $diasemana = "Quinta Feira"; break;
```

```
    case"5": $diasemana = "Sexta Feira"; break;
```

```
    case"6": $diasemana = "Sabado"; break;
```

```
}
```

```
echo $diasemana;
```

?>

# Estrutura de Repetição



# PHP

## Estruturas de Repetição

As estruturas de repetição são utilizadas quando o programador precisa, por exemplo, repetir o mesmo comando várias vezes. Vamos ver as estruturas de repetição existentes.

### While

O **while** é o comando de repetição (laço) mais simples. Ele testa uma condição e executa um comando, ou um bloco de comandos, até que a condição testada seja falsa. Assim como o **if**, o **while** também possui duas sintaxes alternativas:

#### Sintaxes:

##### While (condição)

```
comando;
```

Ou

##### While (condição)

```
{  
    comandos;  
    comandos;  
}
```

# PHP

## Exemplo:

```
<?php
    $a = 0;
    while ($a i<= 10)
    {
        echo $a."<br>";
        $a++;
    }
?>
```

A expressão só é testada a cada vez que o bloco de instruções termina, além do teste inicial. Se o valor da expressão passar a ser **false** no meio do bloco de instruções, a execução segue até o final do bloco. Se no teste inicial a condição for avaliada como **false**, o bloco de comandos não será executado.

# PHP

## For

O tipo de laço mais complexo é o for. Para os que programam em C, C++ ou Java, a assimilação do funcionamento do for é natural. Mas para aqueles que estão acostumados a linguagens como Pascal, há uma grande mudança para o uso do for.

### Sintaxe:

```
for (<inicializacao>;<condicao>;<incremento>) {  
    <comando>;  
    ...  
    <comando>;  
}
```

# PHP

As três expressões que ficam entre parênteses têm as seguintes finalidades:

**1 – Inicialização** → comando ou sequência de comandos a serem realizados antes do início do laço. Serve para inicializar variáveis.

**2 – Condição** → Expressão booleana que define se os comandos que estão dentro do laço serão executados ou não. Enquanto a expressão for verdadeira (valor diferente de zero) os comandos serão executados.

**3 – Incremento** → Comando executado ao final de cada execução do laço.

# PHP

## Exemplo:

```
<?php
    For ($a = 0; $a <10; $a++)
    {
        echo "O valor de A é: ".$a;
        echo "<br>";
    }
?>

<?php
    $nomes = ["Pedro", "Antônia", "Beto", "Julia"];
    for ($contador = 0; $contador < count($nomes); $contador++)
    {
        echo $nomes[$contador];
    }
?>
```

função count() → conta a quantidade de informações no array.

# PHP

## Foreach

É uma estrutura de controle mais simplificada e um pouco diferente das anteriores. Ela é mais simples pois faz a iteração exclusivamente sobre um objeto pré-existente, como uma lista ou array. Como sua iteração se dá sobre um array, **não é necessário definir seu início ou fim**, assim como não é necessário ter um incrementador para ele.

Seu nome significa **para cada**, literalmente. Ou seja, sua estrutura executa o bloco de comando para cada item da lista ou array, automaticamente.

## Sintaxe:

```
<?php
    foreach ($nome_lista as $apelido){
        <comando>;
    }
?>
```

# PHP

## Exemplo:

```
<?php
    $nomes = ["Pedro", "Antônia", "Beto", "Julia"];
    foreach ($nomes as $valor)
    {
        echo "$valor <br>";
    }
?>
```



# PHP

## Exemplo:

```
<?php
    $numeros = array("pi" => 3.14159, "euler" => 2.71828, "omega" =>
    1.61803, "raiz de 2" => 1.41421);
    foreach ($numeros as $chave => $valor)
    {
        echo "$chave. ' é igual a ' . $valor <br>";
    }
?>
```

Neste exemplo, tanto o índice (chave) quanto seu valor foram armazenados em variáveis temporárias chamadas, respectivamente, de \$chave e \$valor, lembrando que estes nomes poderiam ser qualquer outro.



# PHP

## Quebra de fluxo

### Break

O comando **break** pode ser utilizado em laços de **do**, **for** e **while**, além do uso já visto no comando **switch**.

Ao encontrar um **break** dentro de um desses laços, o interpretador PHP pára imediatamente a execução do laço, seguindo normalmente o fluxo do script.

#### Exemplo:

```
<?php
    $x = 20;
    while ($x < 0) {
        if ($x == 5) {
            echo "Número inválido";
            break;
        }
        echo "Número ".$x."<br>";
        $x--;
    }
?>
```

# PHP

## Continue

O comando continue também deve ser utilizado no interior de laços, e funciona de maneira semelhante ao break, com a diferença que o fluxo ao invés de sair do laço volta para o início dele.

### Exemplo:

```
<?php
    for ($i = 0; $i < 100; $i++) {
        if ($i % 2)
            continue;
        echo " $i ";
    }
?>
```

O exemplo acima é uma maneira ineficiente de imprimir os números pares entre 0 e 99. O que o laço faz é testar se o resto da divisão entre o número e 2 é 0. Se for diferente de zero (valor lógico true) o interpretador encontrará um continue, que faz com que os comandos seguintes do interior do laço sejam ignorados, seguindo para a próxima iteração.

# Funções

# PHP

## Funções

### Sintaxe:

```
function nome_da_função([arg1, arg2, arg3])  
{  
    Comandos;  
    return <valor de retorno>;  
}
```

Qualquer código PHP válido pode estar contido no interior de uma função. Como a checagem de tipos em PHP é dinâmica, o tipo de retorno não deve ser declarado, sendo necessário que o programador esteja atento para que a função retorne o tipo desejado. É recomendável que esteja tudo bem documentado para facilitar a leitura e compreensão do código. Para efeito de documentação, utiliza-se o seguinte formato de declaração de função.

# PHP

**tipo function nome\_da\_funcao(tipo arg1, tipo arg2, ...);**

Este formato só deve ser utilizado na documentação do script, pois o PHP não aceita a declaração de tipos.

Isso significa que em muitos casos o programador deve estar atento aos tipos dos valores passados como parâmetros, pois se não for passado o tipo esperado não é emitido nenhum alerta pelo interpretador PHP, já que este não testa os tipos.

## Valor de retorno

Toda função pode opcionalmente retornar um valor, ou simplesmente executar os comandos e não retornar valor algum.

Não é possível que uma função retorne mais de um valor, mas é permitido fazer com que uma função retorne um valor composto, como listas ou arrays.

# PHP

## Argumentos

É possível passar argumentos para uma função. Eles devem ser declarados logo após o nome da função, entre parênteses, e tornam-se variáveis pertencentes ao escopo local da função. A declaração do tipo de cada argumento também é utilizada apenas para efeito de documentação.

## Exemplo:

```
<?php
    function imprime($texto)
    {
        echo $texto;
    }

    imprime("teste de funções"); //chama a função
?>
```

# PHP

## Passagem de parâmetros por referência

Normalmente, a passagem de parâmetros em PHP é feita por valor, ou seja, se o conteúdo da variável for alterado, essa alteração não afeta a variável original.

### Exemplo:

```
<?php
function mais8($numero)
{
    $numero += 8;
    echo $numero. "<br>";
}
$a = 3;
mais8($a); //chama a função e $a continua valendo 3
echo $a;
?>
```



# PHP

## Parâmetros da função por valor

Consiste na passagem de valores para os parâmetros definidos na função.

### Exemplo:

```
<?php
function soma($v1, $v2)
{
    $somaval = $v1+$v2;
    return $somaval;
}

$num1 = 10;
$num2 = 7;
echo soma($num1, $num2); //chama a função
?>
```



# Formulários

# PHP

Por ser uma linguagem de marcação, a sintaxe do HTML na maioria dos casos exige uma “tag” de início e uma de final daquele bloco. É exatamente isso que ocorre com a definição de um formulário: uma tag no início e outra no final, sendo que todos os elementos do formulário devem estar entre as duas tags. Isto torna possível a inclusão de mais de um formulário num mesmo html. As tags citadas são:

```
<form name="..." action="..." method="..." enctype="...">
```

## Onde temos:

- 1 – **name** → o identificador do formulário. Utilizado principalmente em Scripts client-side (JavaScript);
- 2 – **action** → nome do script que receberá os dados do formulário ao ser submetido;
- 3 – **method** → método de envio dos dados: get ou post;
- 4 – **enctype** → formato em que os dados serão enviados. O default é urlencoded. Se for utilizado um elemento do tipo upload de arquivo (file) é preciso utilizar o tipo multipart/form-data.

# PHP

## Exemplo:

```
<form action="exemplo.php" method="post">  
  (textos e elementos do form)  
</form>
```

Cada elemento do formulário deve possuir um nome que irá identificá-lo no momento em que o script indicado no **ACTION** for tratar os dados.

O **exemplo.php** deverá ser um outro arquivo criado para receber as informações enviadas pelo formulário atual.

# PHP

## A tag <input>

Muitos elementos de um formulário html são definidos pela tag <input>. Cada tipo de elemento possui parâmetros próprios, mas todos possuem pelo menos dois parâmetros em comum: **type**, que define o tipo de elemento, e **name**, que como já foi dito define o nome daquele elemento. Existem, ainda, outros parâmetros, também chamados atributos que são comuns a maioria dos campos de um formulário.

São eles:

- 1 – Value** → o valor pré-definido do elemento, que aparecerá quando a página for carregada;
- 2 – Size** → o tamanho do elemento na tela, em caracteres;
- 3 – Maxlength** → o tamanho máximo do texto contido no elemento, em caracteres;

# PHP

## Campo de Texto (Text)

Exibe na tela um campo para entrada de texto com apenas uma linha.

### Sintaxe:

```
<input type="text" name="..." value="..." size="..." maxlength="...">
```

## Campo de Texto com Máscara (Password)

Tipo de campo semelhante ao anterior, com a diferença que neste caso os dados digitados são substituídos por asteriscos, e por isso são os mais recomendados para campos que devam conter senhas. É importante salientar que nenhuma criptografia é utilizada. Apenas não aparece na tela o que está sendo digitado.

### Sintaxe:

```
<input type="password" name=".." value=".." size=".." maxlength="..">
```

# PHP

## Checkbox

Utilizado para campos de múltipla escolha, onde o usuário pode marcar mais de uma opção.

### Sintaxe:

```
<input type="checkbox" name="..." value="..." checked>
```

Onde:

- 1 – Value** → o valor que será enviado ao servidor quando o formulário for submetido, no caso do campo estar marcado;
- 2 – Checked** → o estado inicial do elemento. Quando presente, o elemento já aparece marcado;



# PHP

## Radio Button

Utilizado para campos de múltipla escolha, onde o usuário pode marcar apenas uma opção. Para agrupar vários elementos deste tipo, fazendo com que eles sejam exclusivos, basta atribuir o mesmo nome a todos do grupo.

### Sintaxe:

```
<input type="radio" name=" ..." value=" ..." checked>
```

Onde:

- 1 – Value** → o valor que será enviado ao servidor quando o formulário for submetido, no caso do campo estar marcado;
- 2 – Checked** → o estado inicial do elemento. Quando presente, o elemento já aparece marcado;

# PHP

## Upload de arquivos (file)

Exibe na tela do browser um campo de texto e um botão, que ao clicado abre uma janela para localizar um arquivo no disco. Para utilizar este tipo de componente, o formulário deverá utilizar o método “**POST**” e ter o parâmetro “**enctype**” com o valor “**multipart/form-data**”.

### Sintaxe:

```
<form name="form1" enctype="multipart/form-data" method="post"
action="">
    <input type="file" name="" size="30">
</form>
```

Onde:

**1 – Size** → o tamanho do campo de texto exibido.



# PHP

## TextArea

Exibe na tela uma caixa de texto, com o tamanho definido pelos parâmetros “cols” e “rows”.

### Sintaxe:

```
<textarea cols=".." rows=".." name=".." wrap="..">texto</textarea>
```

Onde:

- 1 – Cols** → número de colunas do campo, em caracteres;
- 2 – Rows** → número de linhas do campo, em caracteres;
- 3 – Wrap** → maneira como são tratadas as quebras de linha automáticas.

# PHP

**1 - O valor `soft`** → faz com que o texto “quebre” somente na tela, sendo enviado para o servidor o texto da maneira como foi digitado;

**2 - O valor `“hard”`** → faz com que seja enviado para o servidor da maneira como o texto aparece na tela, com todas as quebras de linhas inseridas automaticamente;

**3 - O valor `“off”`** → faz com que o texto não quebre na tela e nem quando enviado ao servidor.

**Obs.:** O elemento do tipo `textarea` não possui o parâmetro `“value”`. O valor pré-definido do campo é o texto que fica entre as tags `<textarea>` e `</textarea>`.

# Select

Campo utilizado para que o usuário faça a seleção a partir de uma lista de opções.

## Sintaxe:

[illegible]

# PHP

Onde:

- 1 – Size** → número de linhas exibidas. Default: 1;
- 2 - Multiple** → parâmetro que, se presente, permite que sejam selecionadas duas ou mais linhas, através das teclas Control ou Shift;
- 3 – Option** → cada item do tipo “option” acrescenta uma linha ao select;
- 4 – Value** → valor a ser enviado ao servidor se aquele elemento for selecionado. Default: o texto do item;
- 5 – Texto** → valor a ser exibido para aquele item. Não é definido por um parâmetro, mas pelo texto que fica entre as tags `<option>` e `</option>`

Se o parâmetro “size” tiver o valor 1 e não houver o parâmetro multiple”, exibe na tela uma “combo box”.

Caso contrário, exibe na tela uma “select list”.

# PHP

## Hidden

Campo oculto que é utilizado para se passar parâmetros para o servidor. Este campo não é visível para o usuário.

### Sintaxe:

```
<input type="hidden" name=".." value="..">
```

Onde:

**1 – Value** → o parâmetro que será passado para o servidor.

# PHP

## Submit Button

Utilizado para enviar os dados do formulário para o script descrito na seção “**action**” da definição do formulário

### Sintaxe:

```
<input type="submit" name=".." value="..">
```

Onde:

**1 – Value** → o texto que aparecerá no corpo do botão.

# PHP

## Reset Button

Utilizado para fazer todos os campos do formulário retornem ao valor original, quando a página foi carregada. Bastante utilizado como botão “limpar”, mas na realidade só limpa os campos se todos eles têm como valor uma string vazia.

### Sintaxe:

```
<input type="reset" name=".." value="..">
```

Onde:

**1 – Value** → o texto que aparecerá no corpo do botão.



# PHP

## Exemplo: arquivo botaoreset.php

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Botao de Reset</h1>
    <p>Clique no botão RESET para limpar o formulário</p>
    <form action="/action_page.php">
      <label for="email">Digite seu e-mail:</label>
      <input type="email" id="email" name="email"><br><br>
      <label for="pin">Digite o PIN de validação:</label>
      <input type="text" id="pin" name="pin" maxlength="4"><br><br>
      <input type="reset" value="Reset">
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```



# PHP

## Interagindo o PHP com os Formulários HTML

O que você deverá observar quando criar seus formulários para manipular dados no PHP:

- 1 - Seu formulário deve conter um botão "**SUBMIT**" para poder enviar as informações;
- 2 - Todos os campos do formulário que serão tratados no script PHP devem conter o parâmetro "**NAME**", caso contrário, os dados não serão passados para o script PHP;

Como as informações do formulário são passadas para esse script PHP e como as informações do formulário enviado são tratadas, dependem de você.

# PHP

Existem 2 métodos como as informações podem ser passadas:

- 1 – GET; e
- 2 – POST.

O recomendável sempre, para todos os formulários é usar o método **POST**, onde os dados enviados **não são visíveis** nas URLs, ocultando possíveis importantes informações e permitindo o envio de longas informações.

O **GET** é totalmente o contrário disso.

# PHP

## Variável \$\_POST

Esta variável é utilizada para receber as variáveis vindas do formulário pelo método post.

### Sintaxe:

```
$_POST[campo_do_formulário]
```

onde:

**campo\_do\_formulário** → é o campo que foi criado no formulário que se deseja recuperar.

### Exemplo:

```
<?php  
    $_POST[campo1];  
?>
```

# PHP

Vamos ver um outro exemplo utilizando um formulário HTML e um script PHP que recebe os dados do formulário.

O **script.php** é um arquivo que preciso ser criado para receber as informações do formulário abaixo.

## Exemplo:

```
<form action="script.php" method="post">  
  Campo 1: <input type="text" name="campo1"><br>  
  Campo 2: <input type="text" name="campo2"><br>  
  <input type="submit" value="OK">  
</form>
```

arquivo script.php (vai receber as informações do formulário acima)

```
<?php  
  echo "O valor de CAMPO 1 é: " . $_POST["campo1"];  
  echo "<br>O valor de CAMPO 2 é: " . $_POST["campo2"];  
?>
```

# PHP

## Exercício: arquivo formularioenvio.php / formularioresposta.php

Criar um arquivo html de nome formularioenvio.html

Criar uma tag <form> para que o usuário informe os seguintes dados:

- nome (texto)

- senha (texto)

- gênero (radio button)

- turno (checkbox)

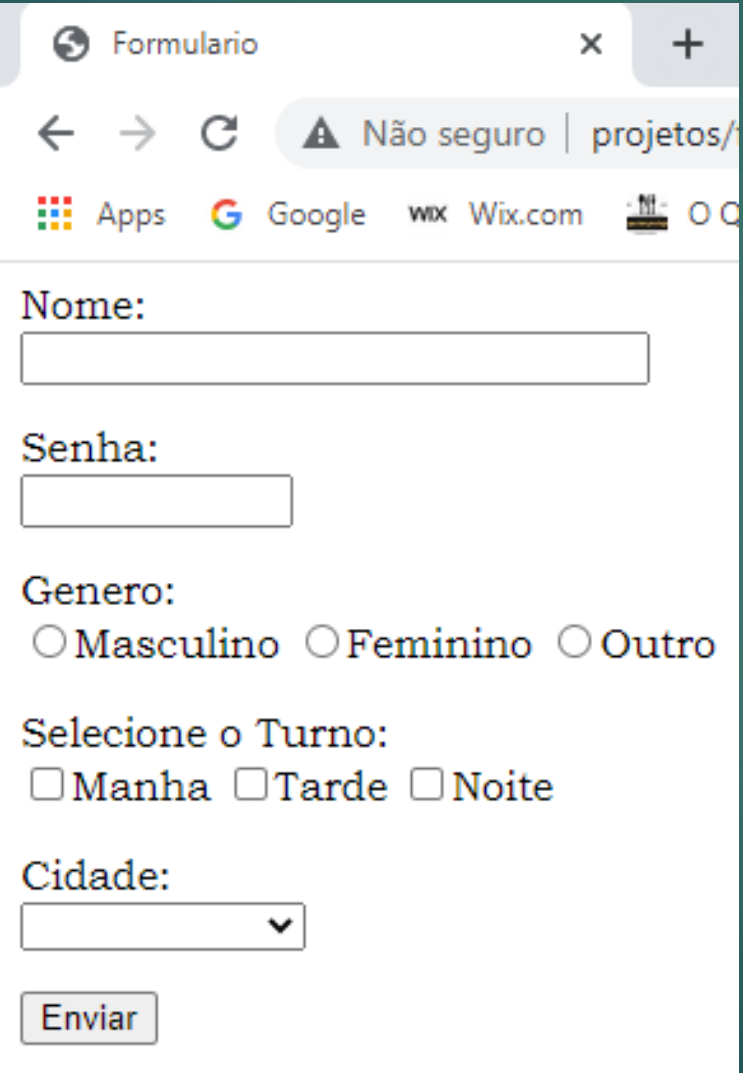
- cidade (select)

- botão enviar (botão)

Após a digitação das informações, as mesmas devem ser enviadas via método POST para o arquivo php de nome formularioresposta.php possa receber as informações e fazer a exibição das mesmas.

# PHP

Como a tela deverá ficar com o primeiro arquivo html:



A screenshot of a web browser window displaying a form titled "Formulario". The browser's address bar shows a warning "Não seguro" (Not secure) and the URL "projetos/". The form contains the following fields and options:

- Nome:** A text input field.
- Senha:** A password input field.
- Genero:** Radio buttons for "Masculino", "Feminino", and "Outro".
- Selecione o Turno:** Checkboxes for "Manha", "Tarde", and "Noite".
- Cidade:** A dropdown menu.
- Enviar** button.

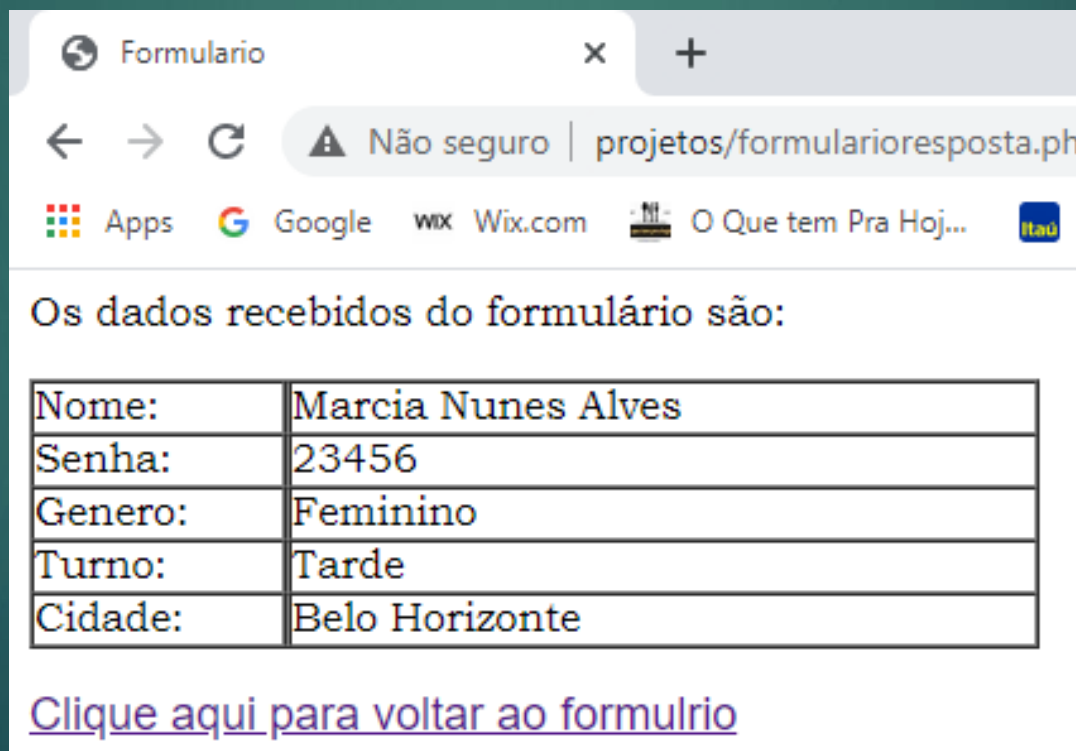
# PHP

## Arquivo que recebe as informações:

No arquivo formularioesposta.php é preciso fazer o código php para receber as informações do arquivo formularioenvio.html.

As informações foram colocadas dentro de uma tabela. Trabalhar com a tag <tr> e <td> para montar a tabela. Fazer um link para que haja um retorno para o formulário anterior.

**O arquivo formularioenvio.html para ser testado deverá ser executado pelo localhost. Se for executado com um duplo click no nome do arquivo irá dar erro no formularioesposta.php**



The screenshot shows a web browser window with the title 'Formulario'. The address bar indicates a non-secure connection to 'projetos/formularioesposta.ph'. Below the browser interface, the text 'Os dados recebidos do formulário são:' is displayed above a table. The table contains five rows of form data: Name (Marcia Nunes Alves), Password (23456), Gender (Feminino), Shift (Tarde), and City (Belo Horizonte). At the bottom, there is a link that says 'Clique aqui para voltar ao formulrio'.

Nome:	Marcia Nunes Alves
Senha:	23456
Genero:	Feminino
Turno:	Tarde
Cidade:	Belo Horizonte

[Clique aqui para voltar ao formulrio](#)



# PHP

## Variável \$\_GET

Esta variável é utilizada para receber as variáveis vindas do formulário pelo método post.

### Sintaxe:

`$_GET[campo_do_formulário]`

onde:

**campo\_do\_formulário** → é o campo que foi criado no formulário que se deseja recuperar.

### Exemplo:

```
<?php
    $_GET[campo1];
?>
```



# PHP

Via GET a informação é passada para o formulário seguinte pela URL.

O **script.php** é um arquivo que preciso ser criado para receber as informações do formulário abaixo.

## Exemplo:

```
<form action="script.php" method="get">  
  Campo 1: <input type="text" name="campo1"><br>  
  Campo 2: <input type="text" name="campo2"><br>  
  <input type="submit" value="OK">  
</form>
```

arquivo script.php (vai receber as informações do formulário acima)

```
<?php  
  echo "O valor de CAMPO 1 é: " . $_GET["campo1"];  
  echo "<br>O valor de CAMPO 2 é: " . $_GET["campo2"];  
?>
```

# PHP

**Exercício:** arquivo `formularioenvioget.php` / `formulariorespostaget.php`

Criar um arquivo html de nome `formularioenvioget.html`

Criar uma tag `<form>` para que o usuário informe os seguintes dados:

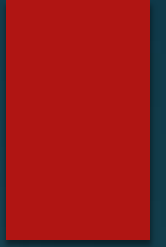
- nome (texto)

- nota1 (float)

- nota2 (float)

- botão enviar (botão)

Após a digitação das informações, as mesmas devem ser enviadas via método GET para o arquivo php de nome `formulariorespostaget.php` possa receber as informações e fazer o cálculo da média e a exibição das informações.



# Conexão com Banco de Dados MySQL

# PHP

## Abrindo e fechando conexão com o MySQL

### **mysqli\_connect( )**

Comando utilizado para criar a conexão com a base de dados num servidor MySQL.

Função na versão 7 do PHP.

### **Sintaxe:**

```
mysqli_connect(string [nome do host], string [login], string [senha], base  
de dados [nome da base de dados]);
```

# PHP

onde:

- 1 - **string [host]** → é o endereço do servidor, onde o banco está armazenado;
- 2 - **string [login]** → é o usuário do banco de dados;
- 3 - **string [senha]** → é a senha do usuário do banco de dados MySQL.
- 4 - **string [base de dados]** → é o nome da base de dados que será acessada banco de dados MySQL

# PHP

## mysql\_close( )

Comando utilizado para encerrar uma conexão estabelecida com o comando mysql\_connect antes de chegar ao final do script. Caso esse comando não seja utilizado a conexão é encerrada no final do script.

### Sintaxe:

mysql\_close (string da conexão);

### Exemplo:

```
<?php
    $conexao = mysql_connect("localhost", "root", "", "recodepro");
    mysql_close ($conexao);
?>
```

# PHP

## Exemplo:

```
<?php
    $conexao = mysqli_connect ("localhost", "root", "", "recodepro");
?>
```

O identificador da conexão fica armazenado na variável **\$conexao**, para que uma verificação de conexão possa ser feita.

# PHP

## Exemplo:

```
<?php
    $link = mysqli_connect("localhost", "root", "", "recodepro");

    if (!$link)
        die ("Falha na conexão com o BD " . mysqli_connect_errno());
    else
        echo "Sucesso: Sucesso ao conectar-se com a base de dados MySQL;

    mysqli_close($link);
?>
```



# PHP

## Exemplo:

```
<?php
    $servidor = "localhost";
    $login = "root";
    $senha = "";
    $bancodados = "recodepro";
    $link = mysqli_connect($servidor, $login, $senha, $bancodados);

    if (!$link)
        die ("Falha na conexão com o BD " . mysqli_connect_errno());
    else
        echo "Sucesso: Sucesso ao conectar-se com a base de dados MySQL.";

    mysqli_close($link);
?>
```

# PHP

## mysqli\_query( )

Este comando é utilizado para realizar uma consulta SQL no MySQL.

### Sintaxe:

```
mysqli_query(código sql a ser executado);
```

Onde:

**1 – Código SQL a ser executado** → é um dos comandos utilizados do SQL para efetuar uma consulta, uma inclusão, uma alteração ou uma exclusão no banco de dados.

# PHP

O comando **mysqli\_query()** envia uma query (consulta) para o banco de dados ativo no servidor da conexão informada em **string da consulta**. Se o parâmetro **string da consulta** não é especificado, a ultima conexão aberta é usada. Se nenhuma conexão esta aberta, a função tenta estabelecer uma conexão como **mysqli\_connect()** seja chamada sem argumentos e usá-la. O resultado é guardado em buffer.

## Exemplo:

```
<?php
    $sql = "select * from alunos where id_aluno = 10";
    mysqli_query ($sql);
?>
```

# PHP

Exemplo (inserindo informações na tabela): arquivo sqlinsert.sql

```
<?php
$link = mysqli_connect("localhost", "root", "", "recodepro");
if (!$link)
    die ("Falha na conexão com o BD " . mysqli_connect_errno());
else
    echo "Sucesso: Sucesso ao conectar-se com a base de dados MySQL.";

$inserte = "Insert into curso (cod_cur, nome, tipo, duracao)
values ('C9', 'Letras','Licenciatura', 3)";

mysqli_query ($link, $inserte);

mysqli_close($link);
?>
```

# PHP

Exemplo (selecionando informações na tabela curso): arquivo sqlselect.sql

<?php

```
$con = mysqli_connect("localhost", "root", "", "recodepro");  
if (!$con)  
{  
    die ("Falha na conexão com o BD " .mysqli_connect_errno());  
    exit();  
}
```

```
$sel = "select cod_cur, nome from curso order by cod_cur";  
$result = mysqli_query($con, $sel);
```

```
while($row = mysqli_fetch_assoc($result))  
{  
    printf ("%s (%s)", $row["cod_cur"], $row["nome"]);  
    echo "<hr>";  
}
```

```
mysqli_close($con);
```

?>

# PHP

Exemplo (selecione informações na tabela com print\_r no array): arquivo sqlselect.sql

```
<?php
```

```
$con = mysqli_connect("localhost", "root", "", "recodepro");  
if (!$con)  
{  
    die ("Falha na conexão com o BD " .mysqli_connect_errno());  
    exit();  
}
```

```
$sel = "select cod_cur, nome, tipo, duracao from curso";
```

```
$result = mysqli_query($con, $sel);
```

```
while($row = mysqli_fetch_assoc($result))
```

```
{  
    print_r ($row); // descarrega o array  
    echo "<hr>";  
}
```

```
mysqli_close($con);
```

```
?>
```

# PHP

Exemplo (excluindo uma informação da tabela curso): arquivo sqldelete.sql

<?php

```
$link = mysqli_connect("localhost", "root", "", "recodepro");
```

```
if (!$link)
```

```
{
```

```
    die ("Falha na conexão com o BD " . mysqli_connect_errno());
```

```
    exit();
```

```
}
```

```
$del = "delete from curso where cod_cur = 'C9'";
```

```
mysqli_query ($link, $del);
```

```
mysqli_close($link);
```

?>



# PHP

Exemplo (alterando uma informação da tabela curso): arquivo sqlupdate.sql

```
<?php
$link = mysqli_connect("localhost", "root", "", "recodepro");

if (!$link)
{
    die ("Falha na conexão com o BD " . mysqli_connect_errno());
    exit();
}

$del = "update curso set nome = 'Português Inglês' where cod_cur = 'C1'";
mysqli_query ($link, $del);

mysqli_close($link);
?>
```

# PHP

## Exemplo (inserindo informações em um array): arquivo sqlarray.sql

```
<?php
$con = mysqli_connect("localhost", "root", "", "recodepro");

if (!$con)
{
    die ("Falha na conexão com o BD " . mysqli_connect_errno());
    exit();
}

$cursos = array(); // aqui criamos um array vazio
$resultado = mysqli_query($con, "select * from curso"); // selecionamos todos os cursos do nosso
banco

while($cur = mysqli_fetch_assoc($resultado)) // aqui dizemos que enquanto houver cursos no nosso
resultado realizaremos a logica a seguir
{
    array_push($cursos, $cur); // colocamos nosso curso dentro do array de cursos
}
print_r($cursos); // imprimimos o array
mysqli_close($con);
?>
```

# PHP

## Exemplo (colocando informações de um array na tabela): arquivo sqlarraytabela.sql

```
<?php
$con = mysqli_connect("localhost", "root", "", "recodepro");

if (!$con)
{
    die ("Falha na conexão com o BD " . mysqli_connect_errno());
    exit();
}

$cursos = array(); // aqui criamos um array vazio
$resultado = mysqli_query($con, "select cod_cur, nome from curso"); // selecionamos todos os cursos do nosso banco

while($cur = mysqli_fetch_assoc($resultado)) // aqui dizemos que enquanto houver cursos no nosso resultado realizaremos a logica a seguir
{
    array_push($cursos, $cur); // colocamos nosso curso dentro do array de cursos
}

foreach($cursos as $curso): // para cada curso dentro do nosso array de cursos geramos o código abaixo
?>
<table border="1">
    <tr>
        <td><?php echo $curso["cod_cur"] ?></td>
        <td><?php echo $curso["nome"] ?></td>
    </tr>
</table>

<?php
endforeach; // acaba o nosso laço de repetição

mysqli_close($con);
?>
```

# PHP

## mysqli\_num\_rows()

### Exemplo: arquivo sqlnumrows.sql

<?php

```
$con = mysqli_connect("localhost", "root", "", "recodepro");
```

```
if (!$con)
```

```
{
```

```
    die ("Falha na conexão com o BD " . mysqli_connect_errno());
```

```
    exit();
```

```
}
```

```
$resultado = mysqli_query($con, "select cod_cur from curso");
```

```
printf("Resultado foi %d linhas", mysqli_num_rows($resultado));
```

```
mysqli_close($con);
```

?>