APOSTILA DE APOIO

Programação para Internet Utilizando PHP

professor Bruno Vinícius Marton

Sumário

Introdução

A sigla PHP é um acrônimo recursivo que significa **PHP: Hypertext Preprocessor** (isso mesmo! O primeiro P é de PHP!). A linguagem foi criada por Rasmus Lerdorf em 1994 para ser usado em seu site. O que nosso amigo não imaginava é que pouco mais de dez anos depois, sua criação ficaria tão famosa!

O PHP é uma linguagem denominada **server-side**, enquanto outras como JavaScript são chamadas **client-side**. Vamos começar nossos estudos entendendo esses dois conceitos.

Como funciona o PHP?

Quando acessamos outros sites via Web, dizemos que o nosso computador é o **Cliente**.

Exemplo:

1) Digitamos a URL *www.php.net*, e o nosso navegador envia pela rede uma solicitação de acesso ao site;
2) O servidor que contém o documento HTML requisitado receberá a solicitação feita pelo nosso navegador;
3) Imediatamente uma cópia do documento HTML é enviada para o cliente. Entra em ação o **navegador**, que será o responsável por interpretar as marcas (tags) do documento HTML;
4) Analisando o código HTML, o navegador descobrirá quais são os anexos adicionais que são necessários (fotos, imagens, sons) e solicitará ao servidor.

Note que nessa tecnologia, quem desempenha a função de **processamento** é o **cliente**, por isso ela é chamada **client-side**. A função do servidor é somente fornecer os arquivos solicitados.

Como exemplos de tecnologias client-side podemos citar: **HTML**, **Linguagens script** (JavaScript, JScript, VBScript), **Java Applet**, **VRML** e **Flash**.

Em uma tecnologia desse tipo, normalmente precisaremos instalar programas adicionais ou plug-ins em nosso navegador, já que o cliente é quem fará o processamento.

Vejamos agora uma outra situação. Queremos acessar um site desenvolvido em linguagem PHP. Como será que o cliente reagirá?

Exemplo:

1) A solicitação é exatamente igual à outra;
2) O servidor agora contém um arquivo PHP com um programa escrito na linguagem. Se uma cópia desse arquivo for enviada ao cliente, nada poderá ser feito;
3) No servidor, existe um programa chamado **Apache**, que consegue interpretar as instruções em PHP. Esta não é a única função do Apache, mas para nós é a que mais interessa no momento;
4) Todos os códigos em PHP são interpretadas pelo **Apache** e a saída é totalmente convertida em código HTML, que pode ser perfeitamente interpretado pelo cliente.

Dessa vez ficou claro que o responsável pelo processamento é o servidor. Tecnologias que exigem processamento de código por parte do servidor são chamadas **server-side**.

Como exemplos de tecnologias server-side podemos citar: **PHP**, **ASP** e **JSP**.

Em uma tecnologia desse tipo, o cliente não possui nenhum requisito de software adicional, já que ele receberá o resultado em HTML e precisará apenas do navegador para interpretá-lo.

Ao construir um site utilizando uma linguagem server-side, podemos criar conteúdos dinâmicos e personalizados. Nenhuma tecnologia client-side teria a capacidade para tanto!

Client-side ou **Server-side**? Qual escolher para construir meu site?

A resposta é simples: **ESCOLHA OS DOIS!** Hoje em dia, os sites aplicam vários tipos de técnicas. Para cada situação, um deles se aplica melhor que o outro. Sendo assim, não descarte a possibilidade de aprender Java, JavaScript ou qualquer outra tecnologia do mercado de desenvolvimento de sites.

Instalando o EasyPHP

Como instalar o pacote **EasyPHP** para que possamos programar em PHP?

Você deve estar se perguntando: "-Não é o **Apache** que tem que ser instalado?"

O EasyPHP nada mais é do que um pacote contendo as principais ferramentas para desenvolvimento PHP. Com este pacote são instalados o **Apache**, o **MySQL** e o **MyPHPAdmin**. Mais tarde veremos a função de cada software.

1) Digite no seu navegador a URL *www.easyphp.org* e dê ENTER;
2) Para baixar o pacote, devemos clicar na seção Downloads;
3) Fala o download da última versão do EasyPHP, que estará no topo da lista;
4) Salve o arquivo no Desktop;
5) Execute o arquivo;
6) Escolha o idioma;
7) Avance, aceite o contrato de licença, avance novamente e deixe o caminho de instalação sugerido (C:\Arquivos de programas\EasyPHP-5.3.5.0);
8) Avance. Mantenha o nome do menu que será criado em Iniciar e avance novamente;
9) Tudo configurado, agora é só clicar em Instalar.
10) Após a Instalação clique em Concluir e aguarde o EasyPHP ser executado automaticamente.
11) Para abrir o documento PHP pelo host local criado pelo Apache, você pode utilizar o endereço: http://localhost:8888 ou http://127.0.0.1:8888 (a porta 8888 está configurada automaticamente no Apache).
12) Se o Apache estiver funcionando corretamente, será exibida uma tela do EasyPHP.

Primeiros Passos

Para não fugir a tradição, faremos o programa "Olá Mundo!"

Para programar em PHP, precisaremos apenas de um editor de códigos e um navegador web.

O editor de códigos pode ser o bloco de notas do Windows, outra boa opção seria o **NuSphere PhpED**, pois já vem com colorização de código específica para instruções PHP.

Um documento PHP pode ser composto por tags HTML misturadas com os blocos programados na linguagem. Como diferenciar instruções PHP de tags HTML? Entram as SUPERTAGS:

```
<HTML>
<head>
        <title>Aulas de PHP</title>
</head>
<body>
        <?php
                echo "Olá Mundo!";
        ?>
</body>
</HTML>
```

Para o Apache, tudo aquilo que estiver escrito entre as supertags **<?php** e **?>** será considerado instruções PHP.

A instrução **echo** informa ao Apache que deve ser exibida uma mensagem. As mensagens devem estar entre aspas, e podem conter qualquer caractere alfanumérico. Podemos também colocar tags HTML para compor a mensagem que será exibida. Exemplo: **echo** "<h1>Olá Mundo!</h1";

Para indicar ao PHP que a instrução acabou, usamos um ponto-e-vírgula (;).

Crie o documento ilustrado acima e salve-o em:

C:\Arquivos de Programas\EasyPHP-5.3.5.0\www\Aulas PHP

Crie uma nova pasta dentro de www para gravarmos os exercícios

Nome pode variar de acordo com a versão do programa

A pasta www abrigará os nossos documentos PHP

Salve este primeiro exemplo com o nome **ex01.php**, dentro da pasta **Aulas PHP** que acabou de criar.

Com o documento salvo agora é só abrir o navegador e acessar o servidor web (http://localhost:8888). Perceba que a pasta **Aulas PHP** aparecerá na pasta inicial do EasyPHP. Ao abri-la o documento **ex01.php** aparecerá na lista. Clique nele e confirme que irá aparecer a mensagem "Olá Mundo!" no navegador.

Se você substituir no arquivo o "Olá Mundo!" por "<h1>Olá Mundo!</h1>" ele realmente aparecerá na formatação do H1, pois tudo aquilo que estava escrito em PHP foi completamente interpretado pelo Apache, gerando apenas texto e tags HTML e o resultado foi enviado ao navegador. Faça o teste.

Variáveis e Constantes

Variáveis

Os identificadores de variáveis em PHP devem seguir algumas regras básicas:

- Devem começar pelo símbolo de **$**
- O segundo caractere deve ser uma **letra** ou o caractere **underline** "_"
- Os demais caracteres podem ser letras, números ou underline;
- Não são aceitos símbolos como **!**, **@**, **&**, **)** e outros
- Não utilize caracteres **acentuados**, nem mesmo **ç**

Obs1: É muito comum o programador iniciante esquecer de colocar o **$** no início do identificador de variável. Com o tempo você se acostuma!

Obs2: O PHP é case sensitive, por isso, **$nome**, **$Nome**, **$NOME** e **$NoMe** são variáveis totalmente distintas para a linguagem!

Cientes destas regras vão fazer alguns exercícios:

1) O identificador $_nome é:
   a. Válido
   b. Inválido
2) O identificador $12anos é:
   a. Válido
   b. Inválido
3) O identificador $Sal_Líq é:
   a. Válido
   b. Inválido
4) O identificador $Aumento% é:
   a. Válido
   b. Inválido
5) O identificador $Casal20 é:
   a. Válido
   b. Inválido
6) O identificador Cod_Produto é:
   a. Válido
   b. Inválido

Atribuições

Em programas desenvolvidos em PHP, não precisamos declarar as variáveis, pois elas serão automaticamente criadas conforme elas forem utilizadas. O tipo de cada variável pode ser definido de algumas maneiras, que veremos mais adiante.

Para criarmos uma variável para armazenar um nome, basta que façamos uma atribuição inicial.
Por exemplo: **$nome = "Bruno Marton";**
Ao executar a linha acima, o PHP criaria uma variável **String** $nome, já que o conteúdo atribuído está entre aspas.

De maneira análoga, consideramos as atribuições:

```
$nota = 3.5;
$idade = 15;
```

A variável $nota seria do tipo **Inteiro** e a $idade seria **Real**.

A definição automática de tipos do PHP pode parecer um milagre, mas ela pode confundir o programador desavisado. Por exemplo, veja as linhas a seguir:

```
$a = 4;
$b = "101 Dálmatas";
$s = $a + $b;
echo $s;
```

Pergunto: Qual seria o resultado exibido na tela?

A resposta certa é **105**! Mas como?

O PHP interpretaria $s como a soma numérica entre $a e $b. Como $b é uma string que começa por um número, os caracteres serão ignorados e o valor somará com o valor de $a. Assim, o PHP calcularia **4 + 101 = 105**. Bem intuitivo, não acha?

O principal deslize foi considerar que o operador "**+**" faria a concatenação de strings, mas essa tarefa é realizada pelo operador "**.**" (ponto). Assim, se substituirmos por:

```
$s = $a.$b;
```

Teremos o resultado **4101 Dálmatas**, concatenando as duas variáveis. As conversões necessárias seriam feitas automaticamente pelo PHP.

Tipos primitivos

O PHP suporta alguns tipos primitivos de dados:

- Inteiro simples (**integer** ou **int**)
- Inteiro long (**long**)
- Real precisão simples (**float** ou **real**)
- Real precisão dupla (**double**)
- String (**string**)
- Matriz (**array**)
- Objeto (**object**)

Obs.: O tipo lógico (boolean), apesar de presente nas versões mais recentes, é tratado como valores inteiros (1 – True ou vazio – False).

Conversões Explícitas

Podemos também forçar a conversão de um valor para um determinado tipo primitivo, indicando-o antes da expressão, entre parênteses.

Vejamos um exemplo:

```
$x = 3.5;
$y = 4.3;
$z = “9.9”;
$n1 = (int) $x + $y;
$n2 = (int) ($x + $y);
$n3 = (real) ($y + $z);
echo “Resultados: $n1, $n2, $n3”;
```

A mensagem exibida na tela seria:

Resultados: 7.3, 7, 14.2

Alguma dúvida? Vejamos...

Na variável **$n1**, apenas o valor de $x será convertido para inteiro e somado com $y, que continua do tipo real. Assim: **$n1 = 3 + 4.3 = 7.3**

Na variável **$n2**, o valor da soma entre $x e $y (note os parênteses) será convertido para inteiro.
Assim: **$n2 = 3 + 4 = 7**

Na variável **$n3**, a soma entre $y e $z será mantida como real. Note que $z será convertido automaticamente de string para real. Assim: **$n3 = 4.3 + 9.9 = 14.2**

Constantes

Diferente da maioria das demais linguagens de programação, as regras para nomear constantes em PHP são ligeiramente diferentes das utilizadas nos identificadores de variáveis.

As constantes PHP não começam por $. O restante é totalmente baseado nas regras para variáveis. Para declarar uma constante, usamos a instrução **define**:

```
define (“pi", 3.1415926536);
```

Para utilizar as constantes em nosso código:

```
$circ = (real) 2 * pi * $raio;
```

Importante: Na hora de exibir o conteúdo de uma constante, devemos lembrar que o PHP não terá como diferenciar o identificador da constante do restante do código, portanto devemos usar o operador de concatenação (.)

```
echo “O valor de pi é”.pi;
```

A instrução acima mostrará: **O valor de pi é 3.1415926536**

Cuidado com exibição de variáveis

Considere o seguinte código:

```
$n = “penta”;
echo “O Brasil é $ncampeão!”;
```

O que será exibido?

Acertou quem respondeu **ERRO!**

Na verdade, o PHP vai procurar por uma variável chamada **$ncampeão**. Este problema pode ser resolvido de duas maneiras:

```
echo “O Brasil é “.$n.”campeão!”;
```

ou ainda

```
echo “O Brasil é ${n}campeão!”;
```

Trabalhar com variáveis no PHP pode ser uma grande novidade para programadores experientes em linguagens tipicamente “tipadas” como o Delphi. Em caso de dúvida, lembre-se de alguns detalhes:

- O nome da variável sempre começa por $
- O PHP é case sensitive
- As conversões serão automáticas caso o programador não faça uma conversão explícita
- O operador de concatenação é “.” e não “+”
- Constantes não iniciam com $

Para fixar alguns conceitos importantes, vamos fazer mais alguns exercícios.

7) Considerando as instruções abaixo:

```
$a = “Casal 20”;
$b = 10;
$c = $a + $b;
echo $c;
```

Qual será o resultado exibido na tela?

a. Casal 2010
b. 30
c. 10
d. ERRO!

8) Com a atribuição **$pre = “hiper”**;
Qual das linhas abaixo mostrará a mensagem “hipertexto”?

a. echo “$pre.texto”;
b. echo $pre”texto”;
c. echo “$pretexto”;
d. echo “${pre}texto”;

9) Considerando as instruções abaixo:

```
$x = “P”;
$y = “H”;
$z = “$x$y$x”;
echo $z;
```

Qual será o resultado na tela?

a. XYX
b. PHP
c. NADA
d. ERRO!

10) Qual das opções abaixo faria a declaração de uma constante?

a. define inc=2;
b. define(“inc”,2);
c. define(“$inc”,2);
d. $inc = 2;

Operadores

Os Operadores do PHP são muito parecidos com os utilizados em linguagem C. É muito importante que o programador iniciante esteja bastante familiarizado com o uso de operadores.

Operadores Aritméticos

Utilizados para cálculos com variáveis do tipo numérico. São os operadores mais semelhantes com as demais linguagens.

Considerando:
$a = 5; $b = 2;

Op	Função	Exemplo	Resultado
+	Adição	$a+$b	7
-	Subtração	$a-$b	3
*	Multiplicação	$a*$b	10
/	Divisão	$a/$b	2.5
%	Resto da divisão	$a%$b	1 (5/2=2, resta 1)

Operadores de incremento/decremento

Uma das operações mais utilizadas em programas é o incremento ou decremento de uma variável. Não é difícil encontrar em nossos códigos expressões do tipo:

$cont = $cont + 1;

No PHP, podemos realizar a mesma tarefa usando o operador de incremento:

$cont++;

De forma similar, podemos decrementar uma variável:

$x = $x – 1;

Podemos utilizar o operador de decremento:

$x--;

A posição do operador pode gerar um pré-incremento ou um pós-incremento.

Pré-incremento e pós-incremento

Dependendo da posição do operador de incremento em uma expressão, o resultado final pode ser diferente. Vejamos:

Exemplo 1:
```
$a = 3;
$b = 2;
$s = (++$a) + $b;
echo “$a, $b, $s”;
```
Resultado: **4, 2, 6**

Exemplo 2:
```
$a = 3;
$b = 2;
$s = ($a++) + $b;
echo “$a, $b, $s”;
```
Resultado: **4, 2, 5**

Veja que o resultado final de $s foi diferente nos dois exemplos. No exemplo 1, como foi utilizado o pré-incremento (**++$a**), o valor inicial de $a foi incrementado antes da atribuição em $s. Já no exemplo 2, o uso do pós-incremento (**$a++**) fará com que $s receba o valor inicial de $a somado com $b e só depois será feito o incremento em $a.

Os conceitos acima também podem ser usados com o operador de decremento. Façamos um exercício:

11) Considerando as instruções abaixo:

```
$x = 5;
$y = 4;
$n1 = (--$x)+$y;
echo $n1;
```

Qual será o resultado exibido na tela?

a. 8
b. 9
c. 4
d. 5

Operadores Binários

Operadores que tratam os valores das variáveis de forma binária. Estamos assumindo que você sabe converter um valor na base decimal para binário.

Op	Função
~	NÃO binário
|	OU binário
&	E binário
^	XOR binário
>>	Deslocamento à direita
<<	Deslocamento à esquerda

Para realizar as operações, os valores são convertidos para a base binária e tratados bit a bit. Vamos ver um exemplo para entender melhor os operadores binários.

Considere as seguintes atribuições. Em verde são mostradas as devidas conversões para a base binária.

```
$x = 14;    //1110
$y = 12;    //1100
```

Resultados		
Expressão	Base 2	Base 10
~$x	0001	1
$x & $y	1100	12
$x | $y	1110	14
$x ^ $y	0010	2
$x>>2	0011	3
$x<<2	111000	56

Operadores de Atribuição

Assim com na linguagem C, podemos utilizar operadores que substituem algumas expressões bastantes utilizadas:

Op	Exemplo	Corresponde a
+=	$a += $b;	$a = $a+$b;
-=	$a -= $b;	$a = $a-$b;
*=	$a *= $b;	$a = $a*$b;
/=	$a /= $b;	$a = $a/$b;
%=	$a %= $b;	$a = $a%$b;
.=	$a .= $b;	$a = $a.$b;
&=	$a &= $b;	$a = $a&$b;
|=	$a |= $b;	$a = $a\|$b;
^=	$a ^= $b;	$a = $a^$b;
>>=	$a >>= $b;	$a = $a>>$b;
<<=	$a <<= $b;	$a = $a<<$b;

Mostre que você entendeu os conceitos acima, fazendo um exercício.

12) Considerando as instruções abaixo:

```
$x = 5;
$y = 4;
$x += $y;
echo $x;
echo $y;
```

Quais serão os valores exibidos?

a. 5 e 4
b. 4 e 5
c. 9 e 9
d. 9 e 4

Operadores Relacionais

Parecidos na maioria das linguagens, os operadores relacionais permitem a comparação entre valores.

Op	Função
==	Igual a
===	Idêntico (igual E do mesmo tipo)
>=	Maior ou igual a
<=	Menor ou igual a
>	Maior que
<	Menor que
!=	Diferente de
<>	Diferente de

A grande peculiaridade da linguagem PHP é a presença do operador “idêntico a”. Vejamos a diferença entre ele e o operador “igual a”...

Igual ou Idêntico?

Vejamos o exemplo a seguir:

Exemplo:
```
$a = 3;
$b = "3";
if ($a ==b)
        echo "A é igual a B";
else
        echo "A não é igual a B";
if ($a===$b)
        echo "A é idêntico a B";
else
        echo "A não é idêntico a B";
```

Perceba que a variável **$a guarda o número 3**, enquanto a variável **$b guarda a string "3"**. A maioria das linguagens consideraria $a diferente de $b... Mas não o PHP!

Como já vimos, o PHP tenta converter automaticamente os dados em uma expressão. Assim, a primeira condição mostraria a mensagem **"A é igual a B"**.

De forma diferente, a segunda condição mostraria **"A não é idêntico a B"**. Isso acontece porque para ser idênticas, duas variáveis devem ser **iguais E do mesmo tipo** (sem precisar de nenhum tipo de conversão prévia).

13) Considerando as instruções abaixo:

```
$x = "100 vergonha";
$y = 50;
$z = 2 * $y;
if ($x == $z)
        echo "São iguais";
else
        echo "São diferentes";
```

O que será exibido pelo trecho do código apresentado acima?

a. São iguais
b. São diferentes
c. Não será exibido nada
d. O PHP retornará ERRO

Operadores Lógicos

O PHP reconhece os seguintes operadores lógicos:

Op	Função
!	Negação Lógica
&&	E
||	OU
AND	E
OR	OU
XOR	OU Exclusivo

Sei que você já consegue identificar a diferença dos operadores lógicos, mesmo assim vamos ver uma pequena revisão e fazer uns exercícios.

Operador ! (NOT)

Nega o valor lógico de uma expressão. A tabela-verdade desse operador é a mais simples de todas:

x	!x
V	**F**
F	**V**

Fica fácil perceber que, logicamente, uma coisa que não é **falsa** é **verdadeira**. De maneira similar, uma coisa que não é **verdadeira** só pode ser **falsa**.

Operador || (OR)

Retorna verdadeiro caso **pelo menos uma** das premissas seja **verdadeira**. Vejamos a sua tabela-verdade:

x	y	z	x || y || z
V	V	V	**V**
V	V	F	**V**
V	F	V	**V**
V	F	F	**V**
F	V	V	**V**
F	V	F	**V**
F	F	V	**V**
F	F	F	**F**

Note que usando o operador OU, a expressão só vai retornar falso se todas as premissas também forem falsas.

Operador && (AND)

Retorna falso caso **pelo menos uma** das premissas seja **falsa**. Vejamos a sua tabela-verdade:

x	y	z	x && y && z
V	V	V	**V**
V	V	F	**F**
V	F	V	**F**
V	F	F	**F**
F	V	V	**F**
F	V	F	**F**
F	F	V	**F**
F	F	F	**F**

Note que usando o operador E, a expressão só vai retornar verdadeiro se todas as premissas também forem verdadeiras.

Operador XOR

É o operador OU EXCLUSIVO. Assim, a expressão retornará verdadeiro se pelo menos uma premissa for verdadeira, mas não quando TODAS forem verdadeiras.

x	y	z	x XOR y XOR z
V	V	V	**F**
V	V	F	**V**
V	F	V	**V**
V	F	F	**V**
F	V	V	**V**
F	V	F	**V**
F	F	V	**V**
F	F	F	**F**

Entenda o XOR como: **"OU uma OU outra, mas NUNCA em TODAS elas"**

Todos os Operadores Lógicos

Antes de partirmos para um exercício, vamos rever todos os operadores em uma só tabela:

x	y	z	||	&&	XOR
V	V	V	**V**	**V**	**F**
V	V	F	**V**	**F**	**V**
V	F	V	**V**	**F**	**V**
V	F	F	**V**	**F**	**V**
F	V	V	**V**	**F**	**V**
F	V	F	**V**	**F**	**V**
F	F	V	**V**	**F**	**V**
F	F	F	**F**	**F**	**F**

Só lembrando:

&& = AND (E)

|| = OR (OU)

XOR = OU EXCLUSIVO

Agora, vamos exercitar!

14) Considerando o trecho abaixo:
    $a = 4;
    $b = 8;
    $r = (($a==$b) || ($a+$b ==12));
    O valor lógico de $r é:
        a. Verdadeiro
        b. Falso
15) Considerando o trecho abaixo:
    $a = 4;
    $b = 8;
    $r = (($a==$b) && ($a+$b==12));
    O valor lógico de $r é:
        a. Verdadeiro
        b. Falso
16) Considerando o trecho abaixo:
    $a = 5;
    $b = 10;
    $r = (($a<=$b) XOR (2*$a == $b));
    O valor lógico de $r é:
        a. Verdadeiro
        b. Falso

Operador Ternário

Utilizaremos o operador **?:** para criar condições simples que realizam atribuições. Vejamos um exemplo utilizando a estrutura ***if***:

```
if ($m > 3)
        $a = $a + 1;
else
        $a = $a – 1;
```

Podemos simplificar a programação utilizando o operador ternário:

```
$a = ($m>3)?($a+1):($a-1);
```

Se você ainda não conhece a estrutura ***if***, não se desespere! Veremos seus detalhes mais adiante.

Ordem de Precedência dos Operadores

O PHP trata os operadores em determinada ordem dentro de uma expressão. Caso utilizemos os parênteses, forçaremos uma ordem específica, caso contrário, a execução se dá na seguinte orem:

Ordem	Operadores	Descrição
1º	- ! ~ ++ --	negativo, não-lógico, inversão, incremento e decremento
2º	* / %	multiplicação, divisão, resto
3º	+ - .	adição, subtração, concatenação
4º	<< >>	deslocamento binários
5º	> < >= <=	operadores relacionais
6º	== != <>	igual e diferente
7º	&	AND binário
8º	^	XOR binário
9º	|	OR binário
10º	&&	AND lógico
11º	||	OR lógico
12º	?:	operador ternário
13º	+= -= *= ...	operadores de atribuição
14º	AND	AND lógico (menor prioridade)
15º	XOR	XOR lógico (menor prioridade)
16º	OR	OR lógico (menor prioridade)

É bom você ter essa ordem anotada (o que é meio complicado) ou então utilizar os parênteses para evitar problemas com os seus programas.

Estruturas de Controle

Assim como todas as linguagens, o PHP possui estruturas de controle, que podem causar iterações e desvios condicionais do fluxo de execução de programas. Como já foi citado anteriormente, o PHP é uma linguagem "**C-Like**", por isso utilizarão a mesma sintaxe das estruturas na linguagem-irmã.

As estruturas de controle do PHP não poderiam deixar de ser as nossas velhas conhecidas:

- Estruturas Condicionais
- Estruturas Iterativas

Nosso objetivo é compreender a sintaxe e desenvolver exemplos práticos para uma perfeita compreensão de cada uma das estruturas.

Estruturas Condicionais

As estruturas condicionais suportadas pelo PHP são:

- **if**
- **switch**

Obs.: Lembre-se sempre que o PHP é case-sensitive, isto faz com que as instruções **IF**, **If**, **SWITCH** ou **Switch** sejam consideradas **inválidas**.

Comando if (instrução simples)

Apenas executa um grupo de instruções se o resultado lógico de uma expressão for verdadeiro.

Caso exista mais de uma instrução, será necessário criar um bloco.

```
Para uma única instrução
if (<expressão lógica>)
        <instrução>;

Para mais de uma instrução (bloco)
if (<expressão lógica>)
        {
                <bloco>;
        }
```

Exemplo:

```
$num = 3;
if ($num%2==1)
        $num++;
echo $num;
```

- Note que a estrutura if não possui cláusula **then**, como acontece com algumas outras linguagens.
- O operador **%** calcula o resto da divisão de $num por 2.
- Só existe uma instrução a ser executada caso a expressão seja verdadeira (**não será necessário criar um bloco**).
- A instrução **echo $num;** sempre será executada, pois está fora da estrutura if.
- A estrutura apresentada poderia ser substituída pela instrução ternária **$num=($num%2==1)?$num+1;**
- O resultado exibido será **4**.

Comando if (instrução estendida)

Desvia logicamente o fluxo de instruções para um determinado bloco, de acordo com o resultado de uma expressão lógica. Quando temos apenas uma instrução para cada caso (verdadeiro ou falso), não é necessário criar blocos.

```
if (<expressão lógica>)
        <instrução V>;
else
        <instrução F>;
```

Exemplo:

```
$n1 = 6; $n2 = 4;
$m = ($n1+$n2)/2;
if ($m>=7)
        $sit = "Aprovado";
else
        $sit = "Reprovado";
echo $sit;
```

- Só existe uma instrução a ser executada caso **$m >=7** (**não será necessário criar um bloco**).

- Só existe uma instrução a ser executada caso **$m < 7** (**não será necessário criar um bloco**).

- As duas instruções internas deverão terminar com ponto-e-vírgula.

- A instrução **echo $sit;** sempre será executada, pois está fora da estrutura if.

- A estrutura apresentada poderia ser substituída pela instrução ternária **$sit=($m>=7)?"Aprovado":"Reprovado";**
- O resultado exibido será **Reprovado** já que **$m = 5**.

Comando if (instruções completas)

Quando temos mais de uma instrução para cada caso (verdadeiro ou falso), torna-se necessário criar blocos.

```
if (<expressão lógica>)
        {
                <instrução V>;
        }
else
        {
                <instrução F>;
        }
```

Exemplo:

```
$n1 = 6; $n2 = 4; $ap = 0;
$rp = 0; $m = (n1+n2)/2;
if ($m>=7)
   {
        $sit = "Aprovado";
        $ap ++;
   }
else
   {
        $sit = "Reprovado";
        $rp++;
   }
echo $sit;
```

- Existem duas instruções a serem executadas caso **$m >=7** (**será necessário criar um bloco**).

- Existem duas instruções a serem executadas caso **$m < 7** (**será necessário criar um bloco**).

- Todas instruções internas deverão terminar com ponto-e-vírgula.

- A instrução **echo $sit;** sempre será executada, pois está fora dos blocos da estrutura if.
- O resultado exibido será **Reprovado**.

Comando if (instruções aninhadas)

Uma das maiores causas de dores de cabeça em programadores é o aninhamento de condições, que ocorrem quando temos "um if dentro do outro".

Graças ao PHP, você está livre deste mal, pois apresento-lhes a cláusula **elseif** (assim mesmo, sem espaços).

```
if (<expressão lógica 1>)
        {
                <bloco A>;
        }
elseif (<expressão lógica 2>)
        {
                <bloco B>;
        }
else
        {
                <bloco C>;
        }
```

Exemplo:

```
$n1 = 6; $n2 = 4;
$ap = 0; $rp = 0; $rc =0;
$m = (n1+n2)/2;
if ($m>=7)
   {
        $sit = "Aprovado";
        $ap ++;
   }
elseif ($m>=5 && $m<7)
   {
        $sit = "Recuperação";
        $rc++;
   }
else
   {
        $sit = "Reprovado";
        $rp++;
   }

echo $sit;
```

- Os blocos foram necessários, pois existe mais de uma instrução para cada cláusula.

- O **&&** é o operador lógico E.

- Podemos ter várias cláusulas **elseif** em uma mesma estrutura **if**.

- A instrução **echo $sit;** sempre será executada, pois está fora dos blocos da estrutura if.

- O resultado exibido será **Recuperação**, já que **$m = 5**.

Estruturas iterativas

As estruturas iterativas, também conhecidas como estruturas de repetição, permitem que um bloco de comandos possa ser executado mais de uma vez, sem a necessidade de escrevê-lo várias vezes.

O PHP possui três estruturas de repetição:

- **while**
- **do..while**
- **for**

Obs.: Lembre-se que o PHP é case-sensitive, o que faz com que While, WHILE e WhilE sejam consideradas instruções **inválidas**.

Instrução while

São estruturas de repetição com este lógico no início da iteração. Isto significa que em primeiro lugar será feito um teste lógico, para depois executar ou não o bloco de comandos.

```
while (<expressão lógica>)
        {
        <bloco de instruções>;
        }
```

Exemplo:

```
$a = 0;
while ($a<5)
   {
   $echo "Repetição $a<br>";
   $a++;
   }
```

- A variável **$a** é numérica e fará parte do teste lógico. Leremos a segunda linha como **"enquanto $a for menor que cinco"**.
- A instrução **$a++** pode ser substituída por **$a=$a + 1;** e fará um incremento unitário na variável.
- O resultado exibido no navegador será:

Repetição 0
Repetição 1
Repetição 2
Repetição 3
Repetição 4

17) No código abaixo:

```
<? $x = 2;
while ($x < 10)
{
        $x+=2;
}
echo $x; ?>
```

Qual valor será exibido na tela?

a. 10
b. 8
c. 2
d. 0

Instrução do..while

São estruturas de repetição com teste lógico no final da iteração. Isto significa que em primeiro lugar o bloco será executado, para depois fazer um teste lógico.

Perceba a diferença desta estrutura e a estrutura **while**, pois da primeira vez, o código será executado independente da expressão lógica.

```
do {
        <bloco de instruções>;
   } while (<expressão lógica>);
```

Exemplo:

```
$a = 0;
do {
   $echo "Repetição $a<br>";
   $a++;
   } while ($a<5);
```

- A variável **$a** é numérica e fará parte do teste lógico. Leremos a segunda linha e a quinta linha como **"faça...enquanto $a for menor que cinco"**.
- A instrução **$a++** pode ser substituída por **$a=$a+1;** e fará um incremento unitário na variável.
- O resultado exibido no navegador será:

Repetição 0
Repetição 1
Repetição 2
Repetição 3
Repetição 4

Tem diferença?

Você deve estar se perguntando: Mas qual é a diferença entre as estruturas **while** e **do..while**?

Vou te responder com 2 exemplos:

```
$x = 10;
while ($x<10)
   {
        $echo "O valor é $x";
   }
```

```
$x = 10;
do {
        $echo "O valor é $x";
   } while ($x<10);
```

E qual é a diferença?

Aparentemente são exemplos com a mesma lógica, mas elas se diferenciam por um pequeno detalhe:

- **No primeiro exemplo**, **$x** já vale **10**, assim o bloco não será executado, pois o valor já não satisfaz a expressão lógica (**$x < 10**).

- **No segundo exemplo**, primeiro será executado o comando echo para só depois verificar a expressão lógica.

Resultado: No **primeiro caso**, **nada será exibido**. Já no **segundo caso**, será exibida a mensagem **O valor é 10**.

Instrução for

São estruturas de repetição com variável de controle auto-incrementável. São as estruturas iterativas mais utilizadas quando sabemos quantas repetições são necessárias para desempenhar determinada tarefa.

```
for (<inicialização>; <condição>; <incremento> )
{
        <bloco de instruções>;
}
```

Exemplo:

```
for ($c=1; $c<20; $c+=5)
{
   $echo "Valor: $c <br>";
}
```

- A variável **$c** é numérica e fará parte do teste lógico. Leremos a primeira linha como **"$c começa com o valor 1, e enquanto for menor que 20, vai aumentando AUTOMATICAMENTE de 5 em 5"**.
- A instrução **$c+=5** pode ser substituída por **$c = $c+5;** e fará um incremento na variável.
- O resultado será:

Valor: 1
Valor: 5
Valor: 10
Valor: 15

Obs.: O valor final de $c não será exibido, pois na última iteração, $c é igual a 20 e não menor que 20.

18) Quantas vezes será exibida na tela a mensagem "Olá!", se executarmos o código abaixo?

```
for ($i=0; $i<=6; i++)
{
        echo "Olá!";
}
```

   a. 5 vezes
   b. 6 vezes
   c. 7 vezes
   d. nenhuma vez

19) Qual das estruturas abaixo possui incremento automático no loop?
   a. while
   b. do..while
   c. for
   d. nenhuma delas

20) Qual das opções abaixo causará um loop com exatamente **10 iterações**?
   a. for ($i=0; $i<=10; i++)
   b. for ($i=1; $i<10; i++)
   c. for ($i=10; $i<=100; i+=10)
   d. for ($i=10; $i<=10; i++)

21) O que será exibido no navegador após a execução do seguinte programa PHP?

```
<? $ast = 1;
for ($l = 1; $l<=3; $l++)
{
        for ($c=1; $c<=ast; $c++)
        {
                echo “*”;
        }
echo “<br>”
$ast+=2;
}?>
```

a.
```
*
***
*****
```

b.
```
  *
 ***
*****
```

c.
```
    *
  ***
*****
```

Formulários HTML

Os formulários criados em HTML não possuem uma utilidade imediata quando estamos iniciando nossos estudos. O PHP fará uso intensivo dos formulários, criando interfaces que permitam a interação do usuário com o site.

Vamos fazer uma breve revisão dos principais controles de formulário HTML, para que possamos mais tarde interligá-lo com o PHP.

Métodos de envio

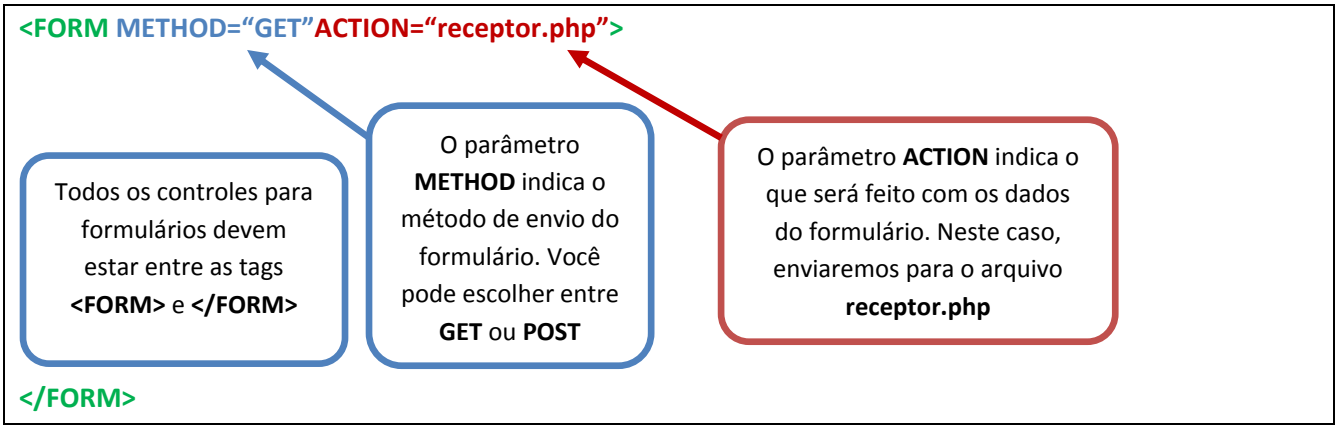
Antes de criarmos um formulário, devemos decidir qual será o método de envio dos dados.

Temos duas opções

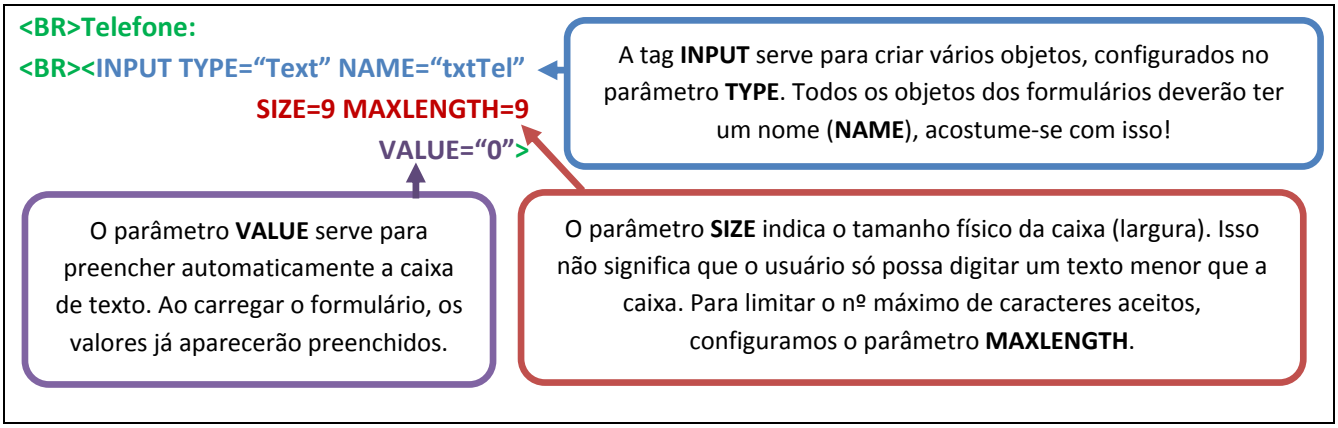
- GET: envia os dados formatados na própria URL. É o método mais simples e rápido, porém, menos seguro.
- POST: Envia os dados por pacotes, de maneira protegida. É o método mais seguro, o que faz com que ele seja um pouco mais lento.

Criando formulários HTML

Tag <FORM>



Caixas de Texto



```
<BR>Telefone:
<BR><INPUT TYPE="Text" NAME="txtTel"
SIZE=9 MAXLENGTH=9
VALUE="0"
TITLE="Digite seu Telefone"
READONLY="true"
>
```

O parâmetro **TITLE** cria uma dica (hint) do controle. Basta parar o cursor do mouse sobre o objeto e aguardar alguns instantes.

O parâmetro **READONLY** quando configurado com o valor "true", impede que o usuário altere o conteúdo da caixa de texto. O usuário poderá ler o conteúdo, mas não poderá mudá-lo.

Caixas de Senha

```
<BR>Senha:
<BR><INPUT TYPE="Password" NAME="txtSen"
SIZE=8 MAXLENGTH=8
TITLE="Digite sua senha">
```

As caixas de senha possuem os mesmos parâmetros das caixas de texto, mas apresentam a entrada do usuário em forma de símbolos, mantendo o sigilo do campo. Para criar um campo para senha, basta utilizar o parâmetro **TYPE="Password"**

Botões de Rádio

```
<BR>Sexo:
<BR><INPUT TYPE="Radio" NAME="radSex"
VALUE="M"> Masculino
<BR><INPUT TYPE="Radio" NAME="radSex"
VALUE="F"> Feminino
```

Um agrupamento de botões de rádio possuem a característica de permitir a seleção de *apenas um item*. O parâmetro **VALUE** configura o valor que será enviado caso o botão seja selecionado.

Importante: Para que os botões participem de um mesmo agrupamento, eles devem ter o mesmo **NAME**, caso contrário o usuário poderá selecionar mais de um item.

Caixas de Seleção

```
<BR>Interesses:
<BR><INPUT TYPE="Checkbox" NAME="chkCin"
VALUE="Cin"> Cinema
<BR><INPUT TYPE="Checkbox" NAME="chkEsp"
VALUE="Esp" CHECKED> Esportes
```

As caixas de seleção permitem que o usuário marque mais de uma opção de um mesmo grupo.
Para criar um objeto deste tipo, configure o parâmetro **TYPE="Checkbox"**

O parâmetro **CHECKED** já marca o item assim que o formulário é carregado.

Campos Escondidos

```
<INPUT TYPE="Hidden" NAME="hidVal" VALUE="12345">
```

Inicialmente, os campos ocultos podem parecer inúteis, já que eles não aparecem na tela. Porém, mais tarde veremos que eles são fundamentais para os scripts PHP.

Caixas de Seleção e Listas de Seleção

```
<SELECT NAME="selLing"> SIZE=3>
<OPTION VALUE="CFM">
        Macromedia ColdFusion
<OPTION>
<OPTION VALUE="ASP">
        Microsoft ASP
<OPTION>
<OPTION VALUE="JSP">
        Sun JSP
<OPTION>
<OPTION VALUE="PHP" SELECTED>
        Zend PHP
<OPTION>
</SELECT>
```

Cada item da caixa de seleção deve ser um elemento **<OPTION>**. O parâmetro **VALUE**, assim como em botões de rádio, configura o valor que será enviado caso o item seja selecionado.

Importante: Para transformar a caixa de seleção em uma **lista de seleção**, basta acrescentar o parâmetro **SIZE** à tag **SELECT**. Se o número de itens for maior que a lista, será exibida uma barra de rolagem.

O item que possui o parâmetro **SELECTED** já aparecerá marcado quando o formulário for carregado.

As caixas de seleção criam uma caixa "pop-up" com várias opções. Todos os itens da caixa de seleção devem estar entre as tags **<SELECT>** e **</SELECT>.**

Áreas de Texto

```
<TEXTAREA NAME="txtMail" ROWS=5 COLS=30>
</TEXTAREA>
```

As áreas de texto permitem a entrada de várias linhas de texto, recurso bastante utilizado em formulários de e-mails e similares. Os parâmetros **ROWS** e **COLS** indicam o tamanho da caixa.

As tags **<TEXTAREA>** devem ser fechadas com **</TEXTAREA>**. é muito comum esquecermos desse detalhe.

Botões de Envio e Reinicialização

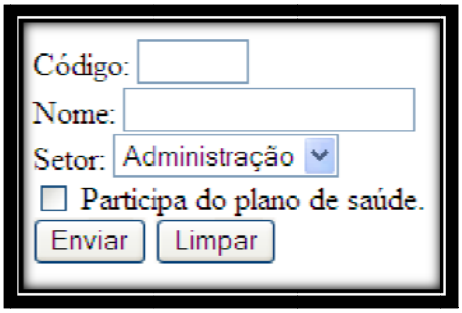
```
<INPUT TYPE="Submit" VALUE="Enviar">
<INPUT TYPE="Reset" VALUE="Limpar">
```

Os botões de ação de um formulário são configurados de acordo com os tipos **Submit** e **Reset**. O primeiro envia o formulário para o destinatário e o segundo limpa todos os campos, retornando aos valores padrões.

Manipulando Dados de Formulários

Nosso Exemplo (Será usado em todos os tópicos deste capítulo):

```
<form method="post" action="recupera.php">
Código:
<input type="text" name="txtCod" size="5" maxlength="5">
<br>Nome:
<input type="text" name="txtNom" size="20" maxlength="50">
<br>Setor:
<select name="selSet">
        <option value="adm">Administração</option>
        <option value="ven">Vendas</option>
        <option value="alm">Almoxarifado</option>
</select>
<br><input type="checkbox" name="chkInf">
Participa do plano de saúde.
<br><input type="submit" value="Enviar">
<input type="reset" value="Limpar">
</form>
```



Como o PHP "pega" os dados?

As primeiras coisas a saber são: qual é o **método** usado e para qual **arquivo** o formulário envia os dados? Vamos ver um trecho do código anterior:

```
<form method="post" action="recupera.php">
```

Agora já sabemos: o método usado foi o **POST** e o arquivo é o **recupera.php**.

E os nomes... São importantes?

Depois que demos o passo inicial, devemos identificar o nome dos controles. Tomaremos como exemplo uma caixa de texto.

```
<input type="text" name="txtCod" size="5" maxlength="5">
```

Para esta caixa, o nome é **txtCod**. Para buscar os dados via PHP, podemos utilizar a seguinte instrução:

```
$c = $_POST["txtCod"];
```

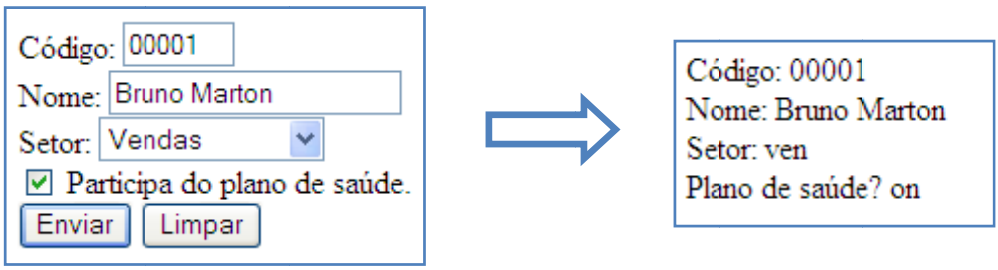
Como seria o início do arquivo **recupera.php** para recuperar todos os campos:

```
<?php
$cod=$_POST["txtCod"];
$nom=$_POST["txtNom"];
$set=$_POST["selSet"];
$pla=$_POST["chkInf"];
?>
```

Para exibir os valores das variáveis criadas, utilizaremos uma seqüência de saídas:

```php
<?php
echo "Código: $cod<br>";
echo "Nome: $nom<br>";
echo "Setor: $set<br>";
echo "Plano de Saúde? $pla";
?>
```

Com o formulário já preenchido, clique em **Enviar**:



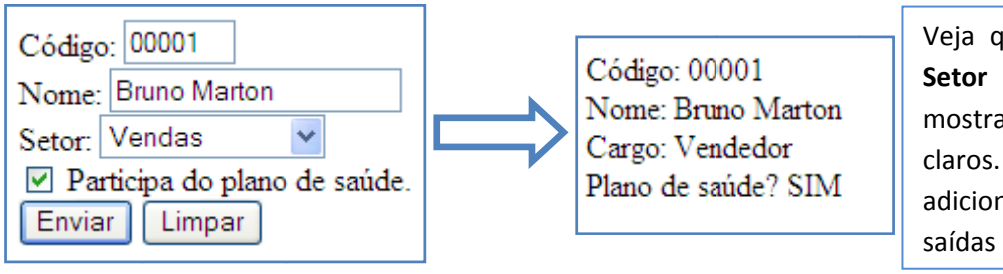
Formulário enviado para o arquivo **recupera.php**... Os campos **Setor** e **Plano de Saúde** mostraram dados um tanto quanto esquisitos, mas vamos resolver isso com um código simples.

```php
switch ($set)
{
case "adm" :
   $car = "Administrador";
   break;
case "ven" :
   $car = "Vendedor";
   break;
case "alm" :
   $car = "Almoxarife";
   break;
}
echo "Cargo : $car<br>";
```

```php
?php
echo "Código: $cod<br>";
echo "Nome: $nom<br>";
echo "Setor: $set<br>";
echo "Plano de Saúde? $pla";
?>
```

```php
if ($pla=="on")
  $resp="SIM";
else
  $resp="NÃO";
echo "Plano de Saúde? $resp";
```

Agora com as correções já feitas no código, preencha o formulário novamente e clique em **Enviar**:



Veja que agora os campos **Setor** e **Plano de Sáude** mostraram dados mais claros. Use sempre códigos adicionais para melhorar as saídas dos seus códigos.

E se o método for outro?

Você deve estar se perguntando: ***- "E se o formulário usar o método GET?"***

A resposta é simples:
se for GET, usa a instrução
$cod = $_GET["txtCod"];
se for POST, use a instrução
$cod = $_POST["txtCod"];

Fácil não? É só você não esquecer que o PHP é case-sensitive, e o comando é **$_GET** e não **$_get**. Sacou?

Nem tudo são flores...

O exemplo anterior pode parecer perfeito... Nada pode dar errado!

Acontece que se o usuário deixar o formulário em branco e clicar em Enviar, o resultado não será satisfatório. É preciso validar as entradas do usuário antes de dar prosseguimento.