



DEFINIÇÃO

A linguagem PHP (PHP: Hypertext Preprocessor), conceitos de programação, variáveis, estruturas de decisão e de repetição, arrays e funções.

PROPÓSITO

Conhecer a linguagem de programação PHP, seus conceitos básicos e recursos disponíveis é importante na formação do desenvolvedor Web do lado servidor.

PREPARAÇÃO

Para melhor compreensão do conteúdo deste tema, é recomendado um conhecimento básico de lógica de programação e de HTML.

Para a aplicação dos exemplos, será necessário um editor de texto com suporte à marcação PHP e um servidor Web com suporte à linguagem. Em relação ao editor, no sistema operacional Windows é indicado o *Notepad++*. No Linux, o Nano Editor. Quanto ao servidor, recomenda-se o Apache.

OBJETIVOS

MÓDULO 1

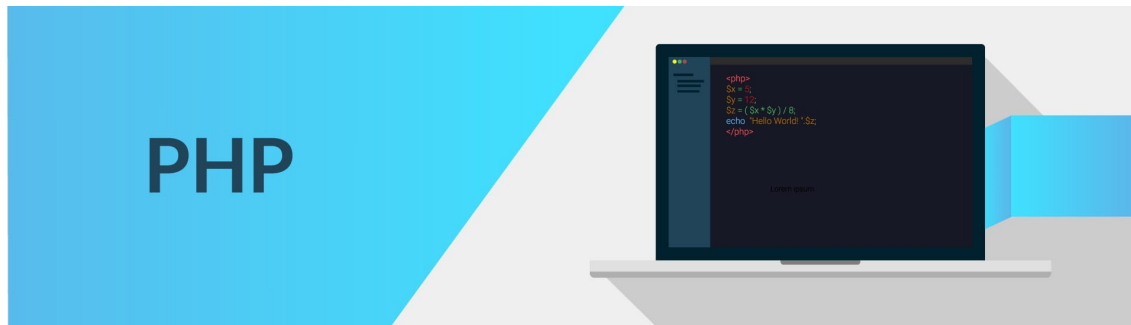
Examinar a linguagem de programação PHP e seus conceitos básicos

MÓDULO 2

Aplicar as estruturas de decisão e de repetição disponíveis em PHP

MÓDULO 3

Identificar conceitos relativos a vetores e funções em PHP



📷 Fonte: Shutterstock | Eny Setiyowati

INTRODUÇÃO

Como veremos neste tema, PHP é uma linguagem *server side*, gratuita e multiplataforma. Logo, para ser executada, precisa de um servidor com suporte à linguagem. Entre os mais utilizados estão o **Apache**, o **IIS** (da Microsoft) e o **Nginx**. A configuração desses ambientes exige um conhecimento intermediário/avançado e pode ser realizada tanto localmente – em computadores pessoais – quanto em servidores através de serviços contratados.

💡 DICA

É possível encontrar diversos tutoriais disponíveis na internet que orientam durante o processo de configuração do ambiente PHP.

Normalmente, o ambiente relacionado ao PHP é chamado de **AMP**, onde temos a combinação de três diferentes tecnologias comumente associadas: **Apache** (Servidor), **Mysql** ou **MariaDB** (Sistema Gerenciador de Banco de Dados) e **PHP**. Uma opção simples para configuração do AMP em computadores pessoais e que não requer instalação, já que pode ser apenas executada, é o **XAMPP**. Outra alternativa aos editores e servidor mencionados são os interpretadores online, como **PHPTester**, **Write PHP Online** e **Online PHP Editor**.

Vamos começar nossa jornada acessando os códigos-fontes originais propostos para o aprendizado de PHP. **Baixe o arquivo aqui**, descompactando-o em seu dispositivo. Assim, você poderá utilizar os códigos como material de apoio ao longo do tema!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



MÓDULO 1

A LINGUAGEM PHP



📷 Fonte: Shutterstock | gdainti

O PHP é uma linguagem de script *open source* de uso geral. Muito utilizada, é especialmente adequada para o **desenvolvimento Web** e pode ser embutida dentro do **HTML (PHP)**.

A explicação anterior consta no site oficial do PHP, de onde os fragmentos a seguir também foram retirados:

O QUE DISTINGUE O PHP DE ALGO COMO O *JAVASCRIPT* NO LADO DO CLIENTE É QUE O CÓDIGO É EXECUTADO NO SERVIDOR, GERANDO O HTML QUE É ENTÃO ENVIADO PARA O NAVEGADOR. O NAVEGADOR RECEBE OS

RESULTADOS DA EXECUÇÃO DESSE SCRIPT, MAS NÃO SABE QUAL ERA O CÓDIGO-FONTE (PHP).

O PHP, COMO É CONHECIDO ATUALMENTE, É NA VERDADE O SUCESSOR PARA UM PRODUTO CHAMADO PHP/FI. CRIADA EM 1994 POR RASMUS LERDOF, A PRIMEIRA ENCARNAÇÃO DO PHP FOI UM SIMPLES CONJUNTO DE BINÁRIOS *COMMON GATEWAY INTERFACE* (CGI) ESCRITO EM LINGUAGEM DE PROGRAMAÇÃO C (PHP).

EM JUNHO DE 1995, RASMUS LIBEROU O CÓDIGO-FONTE DO PHP TOOLS PARA O PÚBLICO, O QUE PERMITIU QUE DESENVOLVEDORES USASSEM DA FORMA COMO DESEJASSEM. ISSO PERMITIU – E ENCORAJOU – USUÁRIOS A FORNECEREM CORREÇÕES PARA BUGS NO CÓDIGO E, EM GERAL, APERFEIÇOÁ-LO. EM SETEMBRO DO MESMO ANO, RASMUS EXPANDIU O PHP E – POR UM BREVE PERÍODO – MUDOU O NOME, REFERINDO-SE, AGORA, À FERRAMENTA COMO FI, ABREVIÇÃO PARA "*FORMS INTERPRETER*". A NOVA IMPLEMENTAÇÃO INCLUIU ALGUMAS FUNCIONALIDADES BÁSICAS DO PHP

COMO BEM CONHECEMOS HOJE. TINHA VARIÁVEIS NO ESTILO PERL, INTERPRETAÇÃO AUTOMÁTICA DE VARIÁVEIS DE FORMULÁRIOS E SINTAXE HTML EMBUTIDA (PHP).

PHP, s.d.

Essas citações ajudam a entender o contexto e os propósitos iniciais da criação da linguagem. Como veremos a seguir, **a primeira finalidade do PHP foi interpretar, do lado servidor, os formulários HTML, fornecendo, assim, dinamismo às páginas Web.** Isso porque, com essa linguagem, é possível adicionar recursos como consulta a banco de dados, processamento e tratamento de dados e consumo de recursos externos – como APIs –, entre tantas outras possibilidades.

Para descrevermos a linguagem PHP, é necessário começar pela sua **sintaxe básica**, apresentando **as variáveis, os operadores e as formas de leitura de dados a partir da integração com a HTML.**

SINTAXE

TAGS PHP

O script PHP deve ser iniciado pela tag “<?php” e fechado com a tag “>”. Isso é necessário para que o servidor Web entenda qual código deve ser interpretado e qual deve ser apenas **renderizado**, uma vez que tags HTML podem ser inseridas dentro de um arquivo contendo código PHP. Veja o exemplo a seguir:

RENDERIZADO

Renderização é o processo pelo qual se obtém o produto final de um processamento digital qualquer. Esse processo aplica-se essencialmente em programas de modelagem 2D e 3D, bem como áudio e vídeo.

```
1 <!doctype html>
2 <html lang="pt-BR">
3 <head>Primeiro código PHP com tags HTML</head>
4 <body>
5 <h1>Título do texto</h1>
6 <p><?php echo "Olá, mundo"; ?></p>
7 </body>
8 </html>
```

O código anterior poderia ser salvo como um script PHP.

Por exemplo: “**ola_mundo.php**”. Ao serem interpretados pelo servidor, tanto o código HTML quanto o código PHP, dentro das tags **<?php e ?>**, são convertidos em código HTML normal e renderizados no navegador. O servidor Web pode ser configurado para interpretar scripts PHP sem que seja necessário utilizar a extensão “**.php**”. Nesse caso, é usada outra extensão, ou nenhuma – isso é útil quando não queremos revelar a linguagem utilizada em nosso site.

TÉRMINO DE INSTRUÇÕES E COMENTÁRIOS

As instruções PHP devem ser, obrigatoriamente, **terminadas com a utilização de ponto e vírgula**. Logo, ao final de cada comando, devemos indicar que ele foi terminado.

Em relação aos comentários, temos duas opções:

Os de uma linha são iniciados com duas barras: //

Os de múltiplas linhas são delimitados por /* e */

Veja a seguir os exemplos de finalização de comandos e de utilização de comentários:

```
1 <?php
```

```
2 //Comentários de uma linha são realizados com a utilização de barras duplas
```

```
3
```

```
4 /*
```

```
5 Cada comando deve ser terminado com ;
```

```
6 Abaixo temos alguns exemplos de término de comandos
```

```
7 */
```

```
8
```

```
9 echo "Olá, mundo";
```

```
10 $var1 = 2 + 2;
```

```
11
```

```
12 echo "O Resultado da soma é igual a: " . $var1;
```

DICA

Repare que a tag de fechamento “?” não é obrigatória quando temos apenas código PHP em um script.

VARIÁVEIS EM PHP

As variáveis são um dos principais recursos em uma linguagem de programação. Em PHP, a definição de uma variável é feita com a utilização do símbolo “\$” **seguido do nome da mesma**. No código de exemplo anterior, a variável “\$var1” foi declarada e, ao mesmo tempo, inicializada.

Em PHP, diferentemente de linguagens como Java, não é necessário informar o tipo de variável. Tal fato concede ao PHP a característica de **linguagem fracamente tipada**. Com isso, não há diferenças no momento da criação de variáveis para receber dados numéricos, textuais, alfanuméricos, entre outros.

ATENÇÃO

O único cuidado diz respeito ao momento de atribuição de valores, já que dados do tipo **string**, por exemplo, precisam ser envolvidos por aspas duplas (“ ”) ou simples (‘ ’).

A respeito dos nomes das variáveis temos, ainda, as seguintes regras:

Os nomes de variável são *case-sensitive*. Logo, há diferença entre letras maiúsculas e minúsculas.

Para ser válido, o nome da variável deve começar com uma letra ou um sublinhado (***underscore***).

Após o primeiro *caractere* (letra ou sublinhado) podem ser utilizados letras, números e sublinhados.

ATRIBUIÇÃO DE VALORES

A atribuição de valores a variáveis em PHP é realizada com a utilização do sinal de igual “=”. Veja este novo exemplo:

```
1 <?php
2 $_nomeCurso = "Programação de Páginas Dinâmicas com PHP";
3 $ano_criacao = 1994;
4 $flagLinguagemScript = true;
```

Repare que diversos tipos de dados foram utilizados e atribuídos às variáveis declaradas. Além disso, diferentes convenções de nomeação foram aplicadas – início com ***underscore***; separação por ***underscore***; ***CamelCase***. É boa prática escolher um único padrão e utilizá-lo em todo o projeto.

CAMELCASE

CamelCase é a denominação em inglês para a prática de escrever as palavras compostas ou frases, onde cada palavra é iniciada com maiúsculas e unidas sem espaços.

ATENÇÃO

Em PHP, as variáveis não inicializadas possuem um valor padrão. Nas do tipo booleano, por exemplo, o valor padrão é `false`. Logo, é recomendado – e também uma boa prática – inicializar as variáveis antes de utilizá-las, embora isso não seja obrigatório.

VARIÁVEIS DE REQUISIÇÃO HTTP

Na linguagem PHP estão disponíveis algumas variáveis predefinidas – também chamadas de **superglobais**. Entre elas, estão as de requisição HTTP: `$_REQUEST`, `$_POST` e `$_GET`. Em linhas gerais, essas três variáveis têm a mesma função, ou seja, **receber dados provenientes de formulários HTML ou de outras requisições HTTP que façam uso dos métodos POST e GET**.

SUPERGLOBAIS

Diversas variáveis predefinidas no PHP são "**superglobais**", o que significa que elas estão disponíveis em todos os escopos para todo o script. Não há necessidade de fazer `global $variable`; para acessá-lo dentro de funções ou métodos. Estas variáveis superglobais são: **`$GLOBALS`**.

MÉTODOS DE REQUISIÇÃO HTTP

A especificação do protocolo HTTP estabelece uma série de métodos de requisição cuja função é indicar qual ação deve ser executada por determinado recurso. Nesse sentido, cada um deles implementa uma diferente semântica. São nove os métodos disponíveis:

GET

HEAD

POST

PUT

DELETE

CONNECT

OPTIONS

TRACE

PATCH

SAIBA MAIS

Além dos métodos aqui mencionados, o protocolo HTTP possui, ainda, uma série de outras propriedades relevantes no que tange à programação Web. Recomenda-se, portanto, uma leitura mais aprofundada sobre esse protocolo.

A seguir, veremos exemplos de utilização dos métodos GET e POST, que são os mais usados em programas PHP.

MÉTODO GET

Esse método é utilizado na requisição e na recuperação de recursos de um servidor, como uma página ou um arquivo, entre outros. Veja o exemplo de uma requisição GET:

/endereco_servidor/script.php?var1=value1&var2=value2&var3=value3

 Figura 1: Exemplo de requisição com o método GET

Como visto na figura, a requisição GET é composta pelo endereço (URL e URI) e pela **query string** – pares de nome/valores (var1=value1, ...).

Em linhas gerais, não deve ser utilizado quando estamos lidando com informações sensíveis, uma vez que a **query string** fica visível na barra de endereços do navegador. Outra característica importante desse método é que ele pode ser usado a partir de formulários HTML.


MÉTODO POST

Esse método é usado no envio de dados para o servidor a fim de criar ou atualizar um recurso. A figura 2 mostra o corpo de uma requisição feita com POST:

POST /endereco_servidor/script.php

Host: dominio.com.br

var1=value1&var2=value2&var3=value3

 Figura 2: Exemplo de requisição com o método POST

Assim como o GET, esse método pode ser utilizado em formulários HTML, com a vantagem de não deixar os dados transmitidos visíveis na barra de endereços do navegador – embora seja possível acessá-los analisando a requisição em si.

VARIÁVEL \$_GET

Em PHP, essa variável predefinida é um **array** associativo que contém as variáveis recebidas de métodos HTTP GET. Voltando ao exemplo da figura 1, no script “script1.php” as variáveis passadas seriam representadas da seguinte forma:

```
<?php
```

```
//Requisição GET: /endereco_servidor/script.php?var1=value1&var2=value2&var3=value3
```

```
echo $_GET['var1']; //imprimiria value1
```

```
echo $_GET['var2']; //imprimiria value2
```

```
echo $_GET['var3']; //imprimiria value3
```

📷 Figura 3: Manipulando variáveis com **\$_GET**

Como visto na figura, no **array \$_GET** os índices correspondem aos nomes das variáveis da **query string** submetida com o método GET, assim como seus valores.

VARIÁVEL \$_POST

A exemplo de **\$_GET**, a variável predefinida **\$_POST** também é um **array** associativo. Entretanto, ela contém as variáveis recebidas através de métodos POST.

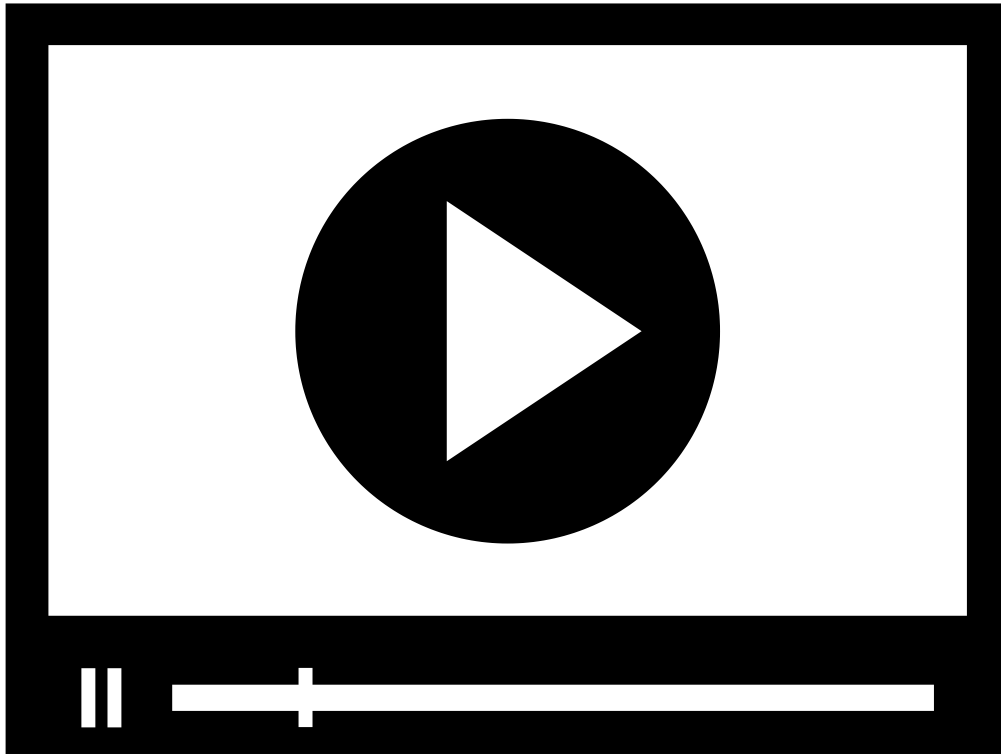
VARIÁVEL \$_REQUEST

É considerada "coringa", uma vez que exerce múltiplos papéis. Com ela, é possível receber tanto variáveis provenientes de métodos GET quanto POST – e também do método **cookies** (**\$_COOKIE**).

Sua utilização é semelhante ao que foi visto em **\$_GET** e **\$_POST**.

COOKIES

O cookie HTTP é um fragmento reduzido de dados que fica armazenado no navegador do usuário, proveniente de um servidor Web. São normalmente usados para fins de gerenciamento de sessões, armazenamento de preferências do usuário ou rastreamento.



EXEMPLO PRÁTICO DE VARIÁVEIS DE REQUISIÇÃO HTTP

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



OPERADORES

Um operador – no contexto de linguagens de programação ou mesmo em outras áreas, como na Matemática – tem a função de **receber um ou mais valores e resultar em outro valor ou valores**. Por exemplo:

$$2 + 2 = 4$$

$$2 - 2 = 0$$

$$2 * 2 = 4$$

$$2 / 2 = 1$$

Os sinais “+”, “-”, “*” e “/” representam as operações matemáticas de adição, subtração, multiplicação e divisão, respectivamente. Logo, são chamados de **operadores aritméticos**. Em PHP, além dos operadores aritméticos, há outros disponíveis. Veremos os principais a seguir:

OPERADORES ARITMÉTICOS

Além dos mencionados no exemplo, também estão disponíveis em PHP outros quatro operadores aritméticos, sendo os dois a seguir os mais importantes:

Operador / Exemplo de utilização		Para que serve
%	\$var1 % \$var2	Operador de módulo. Retorna o resto da divisão inteira de \$var1 por \$var2
**	\$var1 ** \$var2	Operador exponencial. Retorna o resultado de \$var1 elevado a \$var2

Atenção! Para visualizaçãocompleta da tabela utilize a rolagem horizontal

 Tabela 1: Operadores aritméticos em PHP

Esse tipo de operador é usado, sobretudo, com números (*int/integer* e *float*) para a realização de cálculos. Quando utilizado com outra forma de dado, é feita a conversão para o tipo numérico antes que a operação seja realizada.

OPERADORES DE ATRIBUIÇÃO

São utilizados na atribuição de valores a variáveis. Além de casos simples, como o que vimos com o operador “=”, é possível realizar a combinação de operadores de atribuição com os aritméticos. Para ficar mais claro, vamos aos exemplos:

```
<?php
```

```
$var1 = 4; //a variável foi inicializada com o valor de 4
```

```
$var1 += 2; //com a utilização da combinação de operadores a variável $var1 passou a ter o valor de 6 (4 + 2)
```

```
$var1 *= 2; //com a utilização da combinação de operadores a variável $var1 passou a ter o valor de 12 (4 + 2) * 2
```

```
$var2 = "Programação";
```

```
$var2 .= " com PHP"; //com a utilização da combinação de operadores a variável $var2 passou a ter o conteúdo "Programação com PHP"
```

```
$var = ($var4 = "Copie esses códigos") . " e pratique seus conhecimentos!" ;
```

```
/*
```

No exemplo acima o conteúdo da variável \$var3 é igual a "Copie esses códigos e pratique seus conhecimentos!"

Já a variável \$var4 possui o conteúdo "Copie esses códigos"

```
*/
```

 Figura 4: Combinação de operadores de atribuição e aritméticos

DICA

Repare nos comentários inseridos na imagem, onde o resultado de cada combinação é explicado. Para melhor fixação, copie os exemplos e os execute. Tente também alterar os valores ou realizar outras combinações.


OPERADORES DE COMPARAÇÃO

São utilizados para comparar dois valores. A tabela 2 apresenta os operadores disponíveis e suas funções:

Operador / Exemplo de utilização		Para que serve
==	\$var1 == \$var2	Verifica se \$var1 é igual a \$var2
===	\$var1 === \$var2	Verifica se \$var1 é idêntica a \$var2. Nesse caso, além do valor, verifica se ambas são do mesmo tipo
!=	\$var1 != \$var2	Verifica se \$var1 é diferente de \$var2
<>	\$var1 <> \$var2	
!==	\$var1 !== \$var2	Verifica se não são idênticas/iguais ou se não são do mesmo tipo
<	\$var1 < \$var2	Verifica se \$var1 é menor que \$var2
>	\$var1 > \$var2	Verifica se \$var1 é maior que \$var2

<=	\$var1 <= \$var2	Verifica se \$var1 é menor ou igual a \$var2
>=	\$var1 >= \$var2	Verifica se \$var1 é maior ou igual a \$var2

Atenção! Para visualizaçãocompleta da tabela utilize a rolagem horizontal

 Tabela 2: Operadores de comparação

DICA

A partir da versão 7 do PHP, um novo operador foi incluído, o “**spaceship**”, cuja forma de utilização é “**\$var1<=> \$var2**”. Ele retorna -1, 0 ou 1 quando **\$var1** for, respectivamente, menor, igual ou maior que **\$var2**.

OPERADORES LÓGICOS

São usados para combinar expressões lógicas. A tabela 3 mostra os operadores lógicos disponíveis em PHP:

Operador / Exemplo de utilização		Para que serve
<i>and</i>	\$var1 <i>and</i> \$var2	Retorna <i>true</i> se \$var1 E \$var2 forem verdadeiras

<i>or</i> >	<code>\$var1 or \$var2</code>	Retorna <i>true</i> se <code>\$var1</code> OU <code>\$var2</code> forem verdadeiras
<i>xor</i>	<code>\$var1 xor \$var2</code>	Retorna <i>true</i> se <code>\$var1</code> OU <code>\$var2</code> forem verdadeiras, mas não ambas
!	<code>!\$var1</code>	Retorna <i>true</i> se <code>\$var1</code> não for verdadeira
&&	<code>\$var1 && \$var2</code>	Retorna <i>true</i> se <code>\$var1</code> E <code>\$var2</code> forem verdadeiras
	<code>\$var1 \$var2</code>	Retorna <i>true</i> se <code>\$var1</code> OU <code>\$var2</code> forem verdadeiras

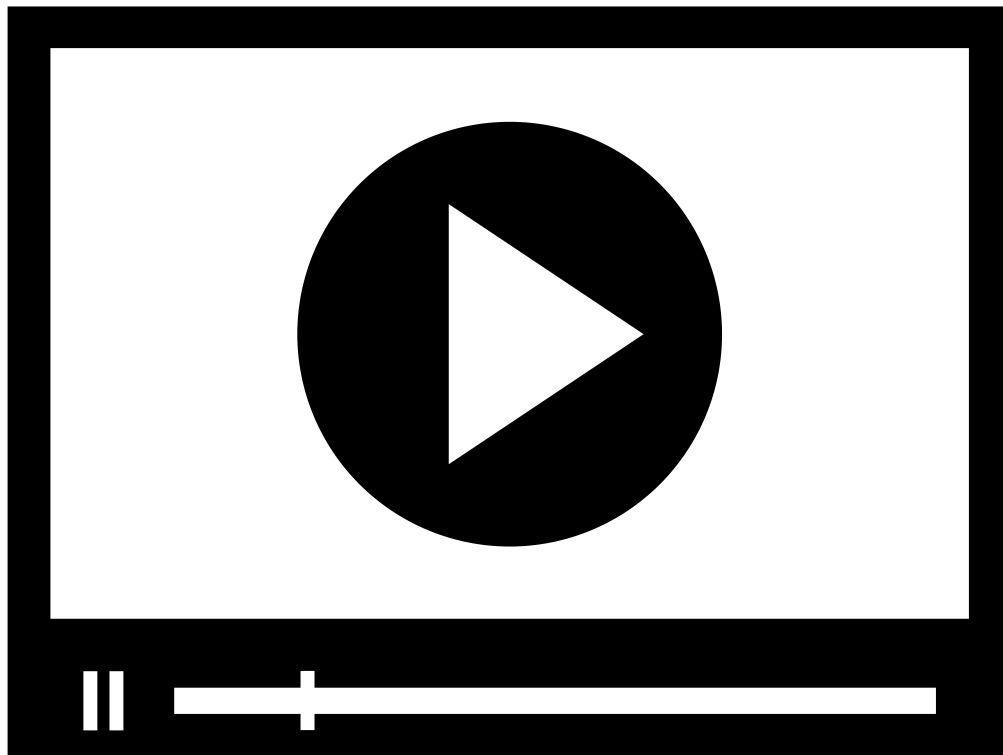
Atenção! Para visualizaçãocompleta da tabela utilize a rolagem horizontal

Tabela 3: Operadores lógicos

Como visto, os operadores “**and**” / “**&&**” e “**or**” / “**||**” têm a mesma função. Entretanto, “**and**” e “**or**” têm maior precedência que seus equivalentes.

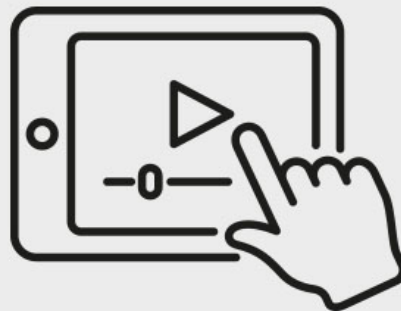
SAIBA MAIS

Além dos operadores apresentados, há outros disponíveis em PHP, como os **bit a bit (bitwise)** e os **de controle de *error***. Na seção Explore+, destacamos um site que contém a lista completa de operadores.



No vídeo a seguir, o professor Alexandre Paixão comenta sobre os conceitos básicos da linguagem PHP. Vamos assistir!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



VERIFICANDO O APRENDIZADO

1. ANALISE O FRAGMENTO DE CÓDIGO ABAIXO E ASSINALE A ALTERNATIVA CORRESPONDENTE À SUA SAÍDA:

```
<?PHP
```

```
$VAR1 == 2;
```

```
ECHO $VAR1;
```

A) 2.

B) *true*.

C) *null*.

D) Variável indefinida (*undefined variable*).

2. A RESPEITO DA MANIPULAÇÃO, EM PHP, DE VALORES RECEBIDOS ATRAVÉS DOS MÉTODOS HTTP POST E GET, ASSINALE A AFIRMATIVA CORRETA:

A) A linguagem PHP oferece amplo suporte ao tratamento de variáveis HTTP. Com isso, independentemente do método utilizado no envio, podemos escolher entre as três variáveis superglobais - **\$_POST**, **\$_GET** e **\$_REQUEST**.

B) Em PHP, é possível tratar variáveis enviadas pelo método HTTP chamado REQUEST fazendo uso da variável **\$_REQUEST**.

C) Variáveis enviadas através do método HTTP GET podem ser manipuladas em PHP através das variáveis globais POST ou GET. Entretanto, as enviadas pelo método POST só podem ser manipuladas com a variável **\$_POST**.

D) Em PHP, estão disponíveis três variáveis superglobais para o tratamento de valores recebidos através de métodos HTTP. Nesse contexto, temos as variáveis **\$_GET** para receber os dados enviados por GET, a **\$_POST** para receber os dados enviados por POST e a **\$_REQUEST**, que recebe tanto os dados enviados por POST quanto por GET.

GABARITO

1. Analise o fragmento de código abaixo e assinale a alternativa correspondente à sua saída:

```
<?php
```

```
$var1 == 2;
```

```
echo $var1;
```

A alternativa **"D "** está correta.

Como vimos, a atribuição de variáveis é feita com a utilização do operador "=", enquanto o sinal "==" é um operador de comparação. Logo, no código acima não foi realizada uma atribuição e nem mesmo uma comparação, uma vez que esta última precisa vir acompanhada de uma estrutura de decisão.

2. A respeito da manipulação, em PHP, de valores recebidos através dos métodos HTTP POST e GET, assinale a afirmativa correta:

A alternativa "D " está correta.

PHP possui variáveis globais específicas para tratar os dados recebidos através de métodos GET e POST, além de uma variável “**coringa**”, a **\$_REQUEST**, que pode receber os dados desses dois métodos e também do método COOKIES.

MÓDULO 2

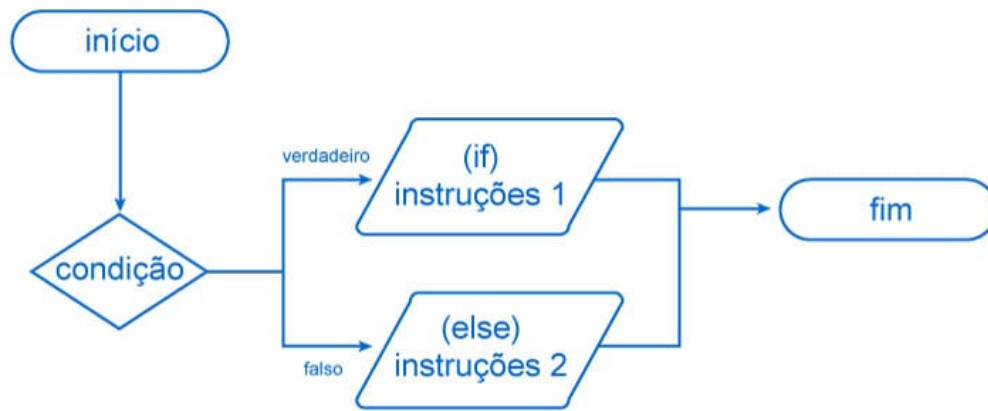
⦿ Aplicar as estruturas de decisão e repetição disponíveis em PHP



📷 Fonte: gdainti | Shutterstock

As **estruturas de decisão e de repetição** são recursos importantes em uma linguagem de programação. Com elas, é possível mudar o fluxo de um programa através de verificações (estruturas de decisão) e também executar diversas vezes partes do programa (estruturas de repetição). A seguir, veremos como utilizá-las em PHP.

ESTRUTURAS DE DECISÃO



Fonte:Shutterstock

📷 Figura 5: Fluxo das estruturas de decisão

A figura 5 demonstra o fluxo de uma **estrutura de decisão**. Nela é possível ver que, a partir da verificação de uma condição, o programa se divide em dois caminhos possíveis. Esse é um exemplo simples, uma vez que várias condições podem ser verificadas ao mesmo tempo, assim como vários caminhos alternativos podem ser seguidos.

Na linguagem de programação PHP, estão disponíveis as seguintes estruturas de decisão: **if**, **else**, **elseif/else if** e **switch**. Cada uma delas será vista a seguir.

IF

A sintaxe da estrutura de controle **if** em PHP é composta pela condição (ou condições) a ser verificada e, caso seja verdadeira, é seguida da instrução (ou instruções) a ser executada. Logo, temos que as condições são avaliadas por seus valores booleanos, isto é, se são verdadeiras ou falsas. Vejamos este fragmento de código PHP:

```
1 <?php
2
3 $var1 = 10;
4 $var2 = 20;
5
6 if($var1 > $var2){
7 echo "$var1 é maior que $var2";
8 }
```

No exemplo foi realizada apenas uma verificação – a comparação entre as duas variáveis definidas. Caso a primeira seja maior que a segunda, uma mensagem é exibida na tela. É possível também realizar outras verificações em um mesmo **if**. Para isso, basta utilizar subgrupos.

Antes de prosseguirmos, cabe destacar mais alguns elementos da sintaxe do **if**:

A condição (ou condições) a ser avaliada deve ser envolvida por parênteses, sendo possível incluir subgrupos dentro de novos parênteses.

Múltiplas instruções devem ser envolvidas com chaves. No exemplo, como só há uma instrução a ser executada, as chaves são opcionais e podem ser omitidas.

ELSE

Para apresentar a estrutura ***else***, voltaremos ao exemplo anterior, no qual é verificado se uma variável era maior do que a outra e, em caso positivo, exibida uma mensagem.

O QUE ACONTECE EM NOSSO PROGRAMA CASO A CONDIÇÃO EM QUESTÃO NÃO SEJA VERDADEIRA?

QUAL RETORNO É EXIBIDO NESSE CASO?

Execute o código e veja você mesmo:

Como \$var1 é menor que \$var2, nada é exibido no navegador, já que a condição é falsa. Nessa situação, para imprimirmos uma mensagem informando que a condição é falsa, ou seja, que \$var1 é menor que \$var2, faremos uso do ***else***. Veja como ficaria nosso código nesse ponto:

```
1 <?php
2
3 $var1 = 10;
4 $var2 = 20;
5
6 if($var1 > $var2){
```

```
7 echo "$var1 é maior que $var2";  
8 }else{  
9 echo "$var1 é menor que $var2";  
10 }
```

Como demonstrado no exemplo, a estrutura **else** tem a função de definir um fluxo alternativo ao nosso programa, caso uma determinada condição seja falsa.

Em relação à sua sintaxe, vale o que foi dito para **if**, sobre múltiplas instruções precisarem ser envolvidas em chaves.

ELSEIF /ELSE IF

É uma combinação das duas instruções vistas. Logo, sua função é definir fluxo (ou fluxos) alternativo caso uma condição verificada com **if** seja falsa. Entretanto, ela permite ainda que uma nova condição (ou até mesmo condições) seja avaliada. Vamos ao exemplo de seu uso:

```
1 <?php  
2  
3 $var1 = 10;  
4 $var2 = 10;  
5  
6 if($var1 > $var2){  
7 echo "$var1 é maior que $var2";  
8 }elseif($var1 < $var2){  
9 echo "$var1 é menor que $var2";
```

```
10 }else{  
11 echo "$var1 e $var2 são iguais";  
12 }
```

Repare que além da primeira verificação, com o ***if***, foi inserida uma segunda, com ***elseif***. Ao final, a instrução ***else*** representa o fluxo caso nem a condição do ***if*** e nem a do ***elseif*** sejam verdadeiras.

Sobre sua sintaxe, além do que já foi dito no ***if***, cabe destacar que não há limites de instruções ***elseif*** dentro de uma declaração ***if***.

SWITCH

Pode ser comparada a uma série de instruções ***if***, possuindo uma sintaxe um pouco diferente e usada sobretudo quando se deseja verificar diferentes valores, inúmeras vezes, em uma mesma variável. Vejamos sua sintaxe por meio de um novo exemplo:

```
1 <?php  
2  
3 switch($var1){  
4 case 10:  
5 echo "var1 é igual a 10";  
6 break;  
7 case 20:  
8 echo "var1 é igual a 20";  
9 break;  
10 default:
```



```
11 echo "var1 não é igual a 10 e nem a 20";  
12 break;  
13 }
```

Com o ***switch*** temos uma série de verificações e, ao final, uma instrução padrão (***default***) a ser executada, caso nenhuma das condições seja verdadeira.

FORMAS ALTERNATIVAS

PHP permite que sejam utilizadas formas alternativas das instruções vistas. Em linhas gerais, troca-se a chave de abertura por dois pontos e a de fechamento pela palavra reservada “***end***” seguida do nome da instrução. Veja o exemplo:

```
1 <?php  
2  
3 $var1 = 10;  
4 $var2 = 10;  
5  
6 if($var1 > $var2):  
7 echo "$var1 é maior que $var2";  
8 elseif($var1 < $var2?):  
9 echo "$var1 é menor que $var2";  
10 else:  
11 echo "$var1 e $var2 são iguais";  
12 endif;
```

Essa sintaxe é muito utilizada quando misturamos código HTML e PHP.

Outra sintaxe alternativa interessante presente no PHP é o **operador ternário**. Através dele é possível avaliar uma condição e atribuir um valor de acordo com a validação. Veja o exemplo para ficar mais claro:

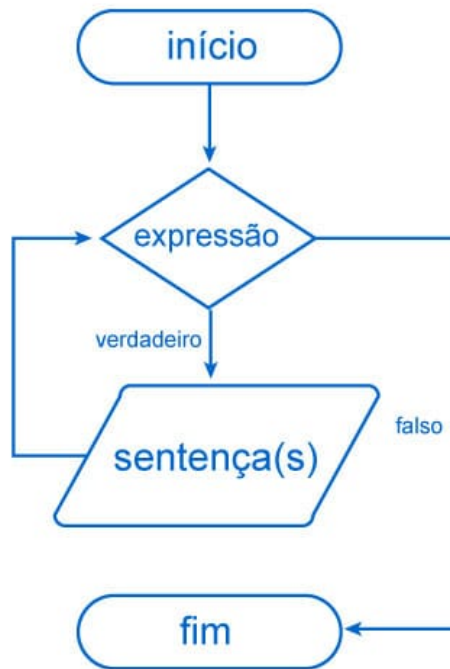
```
1 <?php
2
3 $var1 = 10;
4 $var2 = ($var1 >= 10) ? 11 : 9;
5 echo $var2; //imprimirá 11
```

Note que uma condição foi verificada dentro de parênteses. Caso verdadeira, após o sinal “?” é atribuído o valor “11”. Caso negativa, após o sinal “:” é atribuído o valor “9”.

ESTRUTURAS DE REPETIÇÃO

As **estruturas de repetição**, muitas vezes também chamadas de **laços**, permitem que instruções ou trechos de códigos sejam executados de forma repetitiva. Sua sintaxe define as condições ou expressões que devem ser verificadas e, caso essas sejam verdadeiras, quais instruções devem ser executadas e por quantas vezes. A figura 6 apresenta o fluxo básico das estruturas de repetição.

Em PHP estão disponíveis as seguintes estruturas: **while**, **do-while**, **for**, **foreach**. A seguir, cada uma delas será descrita.



Fonte:Shutterstock

📷 Figura 6: Fluxo das estruturas de repetição

WHILE

O laço ***while*** possui uma sintaxe simples: enquanto uma expressão for verdadeira, uma série de instruções será executada de forma repetida. Para imprimirmos na tela os números de 2 a 20, pulando de 2 em 2, poderíamos utilizar o seguinte código:

```
1 <?php
```

```
2
```

```
3 $i = 2;
4 while ($i <= 20) {
5 echo $i;
6 $i+=2;
7 echo "\n";
8 }
9
10 /*Sintaxe alternativa*/
11 $i = 2;
12 while ($i <= 20):
13 echo $i;
14 $i+=2;
15 echo "\n";
16 endwhile;
```

DICA

Repare que, no exemplo foram mostradas a forma normal e a forma alternativa do laço ***while*** – também presente nas demais estruturas de repetição em PHP. Para praticar, copie o código e o execute. Em seguida, tente alterar a condição e veja o resultado.

ATENÇÃO

O comando **'echo "\n"'** é utilizado para imprimir uma linha em branco quando o script é executado via linha de comando.

DO-WHILE

Esse laço é semelhante ao anterior, exceto pelo fato de que a verificação, aqui, é feita ao final. Com isso, a primeira instrução dentro do laço sempre será executada. Veja o exemplo:

```
1 <?php
2
3 $i = 2;
4 do {
5 echo $i;
6 $i+=2;
7 echo "\n";
8 }while ($i <= 20);
```

FOR

Esse laço possui sintaxe um pouco diferente do que vimos nos anteriores. Vamos ao exemplo:

```
1 <?php
2
3 for ($i = 1; $i <= 20; $i++) {
```

```
4 echo $i;  
5 echo "\n";  
6 }
```

Com o laço **for**, temos três expressões sendo avaliadas. Considerando o exemplo anterior, temos:

A primeira expressão – “**\$i = 0**” – é avaliada, incondicionalmente, no início da repetição.



A seguir, a cada interação, a segunda expressão – “**\$i <= 20**” – é avaliada. Caso seja verdadeira, o loop continuará.



Por último, ao final de cada interação, a terceira expressão – “**\$i++**” – é avaliada e executada.

💡 DICA

Outra possibilidade nesse laço é avaliar **múltiplas expressões**, que deverão ser separadas por vírgulas. Além disso, também é possível inserir **expressões vazias**.

FOREACH

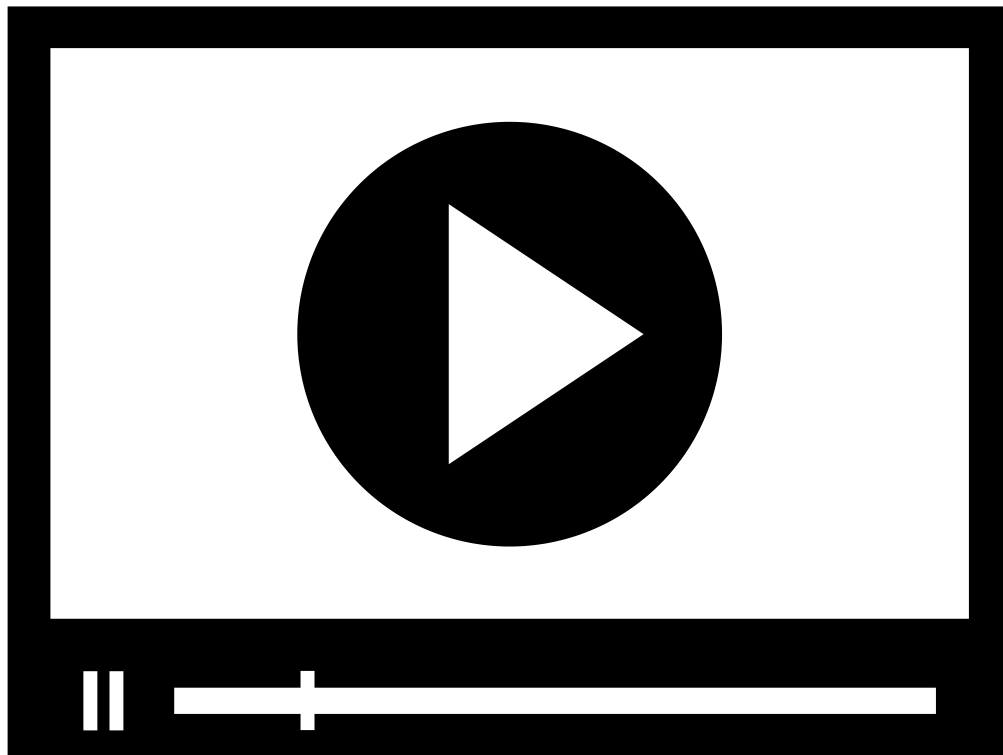
A última estrutura de repetição que veremos em PHP é a ***foreach***. Esse laço é parecido com o ***for***, possuindo uma sintaxe mais simples e sendo muito propício para realizar interações em ***arrays***. Veja o exemplo:

```
1 <?php
2 $carros = Array("Fusca", "Monza", "Passat");
3
4 foreach($carros as $carro){
5 echo $carro;
6 echo "\n";
7 }
8
9 for ($i = 0; $i < count($carros); $i++) {
10 echo $carros[$i];
11 echo "\n";
12 }
```

Para fins de comparação e demonstração da diferença entre ambos, em nosso exemplo a mesma operação – imprimir os nomes dos carros – foi realizada tanto com o laço ***foreach*** quanto com o laço ***for***.

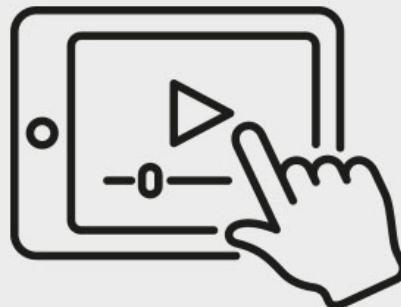
SAIBA MAIS

Além dessas estruturas, em PHP estão disponíveis outros comandos relacionados às estruturas de repetição, como o ***break*** e o ***continue***.



No vídeo a seguir, o professor Alexandre Paixão comenta sobre as diversas formas de estruturas de controle de decisão e de repetição na linguagem PHP. Vamos assistir!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



VERIFICANDO O APRENDIZADO

1. EM PHP É POSSÍVEL INCLUIR SUBCONDIÇÕES DENTRO DE UMA VERIFICAÇÃO A SER TRATADA POR UMA ESTRUTURA DE DECISÃO. CONSIDERANDO O CÓDIGO ABAIXO, ASSINALE A OPÇÃO EQUIVALENTE À INSTRUÇÃO A SER EXECUTADA DENTRO DO *IF*:

```
<?PHP
```

```
$VAR1 = 10;
```

```
$VAR2 = 20;
```

```
$VAR3 = 25;
```

```
IF($VAR1 > $VAR2 || $VAR3 < ($VAR2 + $VAR1)){
```

```
...;
```

```
}
```

A) echo "var3 é menor que a soma de var2 e var1".

B) echo "var3 é maior que a soma de var2 e var1".

C) echo "var3 é igual à soma de var2 e var1".

D) echo "var1 é maior que var2".

2. A RESPEITO DAS ESTRUTURAS DE REPETIÇÃO NA LINGUAGEM PHP, ASSINALE A ALTERNATIVA CORRETA:

- A)** A linguagem PHP possui dois pares de estruturas de repetição: *while* e *do-while*; *for* e *foreach*. Esses pares têm a mesma função, sintaxe, e são executados da mesma forma, tendo sido criados em duplicidade apenas para fornecer uma nomenclatura semelhante à vista em outras linguagens.
- B)** O laço *while* é o mais simples disponível em PHP. Nele, obrigatoriamente, a primeira instrução sempre será executada.
- C)** O laço *for* é bastante flexível, podendo ser usado tanto com múltiplas condições, quanto sem nenhuma condição para ser verificada.
- D)** O laço *foreach* é bastante similar ao *for*, sendo uma forma mais simples para realizar interações em *arrays* PHP.

GABARITO

1. Em PHP é possível incluir subcondições dentro de uma verificação a ser tratada por uma estrutura de decisão. Considerando o código abaixo, assinale a opção equivalente à instrução a ser executada dentro do *if*:

```
<?php
```

```
$var1 = 10;
```

```
$var2 = 20;
```

```
$var3 = 25;
```

```
if($var1 > $var2 || $var3 < ($var2 + $var1)){
```

```
...;
```

```
}
```

A alternativa **"A "** está correta.

Em PHP, é possível incluir e verificar uma subcondição dentro de uma condição a ser verificada em uma estrutura de decisão. Nesta questão, a utilização do operador "||" indica que apenas uma das condições verificadas deve ser verdadeira para que a instrução contida dentro dele seja executada. A primeira ($\$var1 > \$var2$) é falsa. Já a segunda é verdadeira, uma vez que a variável $\$var3$ será comparada com o resultado da soma entre $\$var2$ e $\$var1$ – soma essa que será realizada antes da comparação.

2. A respeito das estruturas de repetição na linguagem PHP, assinale a alternativa correta:

A alternativa **"D "** está correta.

A linguagem PHP possui quatro estruturas de repetição. Tais estruturas têm sintaxes diferentes, sendo cada uma mais indicada para determinadas situações.

MÓDULO 3

⦿ Identificar conceitos relativos a vetores e funções em PHP



📷 Fonte: gdainti | Shutterstock

VETORES

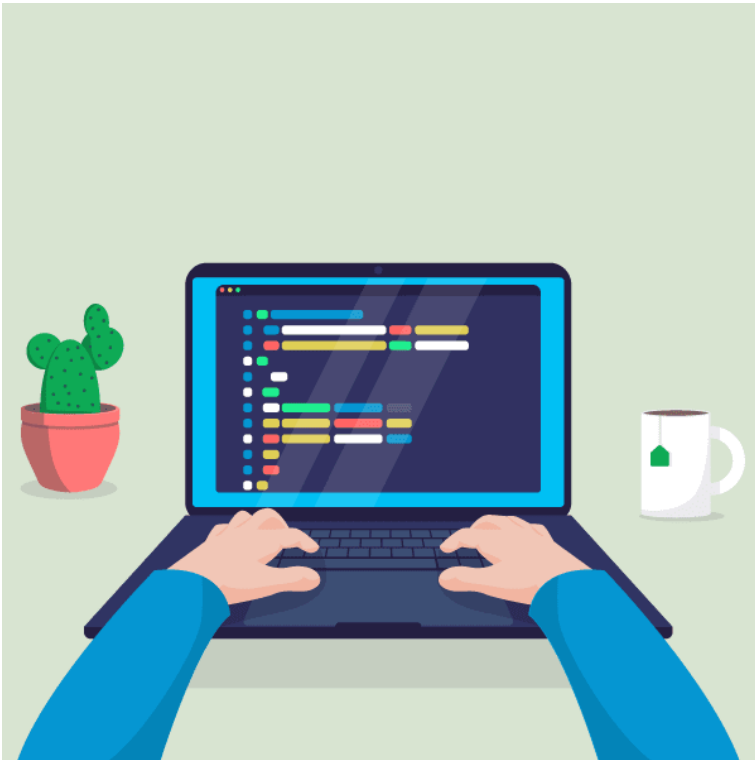
Os vetores, ou *arrays*, são variáveis que armazenam um grupo de itens relacionados. Observando o exemplo utilizado ao final do último módulo, vimos a variável “**\$carros**” armazenar nomes de carros.

Os ***arrays*** podem ser vistos, numa abstração com o nosso dia a dia, como listas escritas em uma folha: nela inserimos vários itens, de forma ordenada. Com isso, cada novo elemento é incluído ao final da lista – embora seja possível inseri-los também em outra ordem. Nas linguagens de programação em geral, e especificamente em PHP, os ***arrays*** funcionam exatamente dessa forma: **uma lista ordenada na qual novos itens podem ser inseridos, assim como os existentes podem ser deletados ou substituídos.**



Fonte: Mascha Tace | Shutterstock





Fonte: - Walnut Bird | Shutterstock

Os **arrays** são compostos por dois elementos principais: **o item em si** e **o seu índice**, que é a posição que este ocupa dentro de um **array**.

Esse número se inicia em **0**. Entretanto, os índices também podem ser formados por **strings**.

Em PHP estão presentes diferentes tipos de **arrays**. São eles:

ARRAY NUMÉRICO

Índice composto por números inteiros

ARRAY ASSOCIATIVO

Índice composto por *strings*

ARRAY MISTO

Índices numéricos e associativos

DICA

O mesmo vale para os elementos de um vetor: podemos tanto ter *strings* quanto números, entre outros tipos de dados, como *arrays* de *arrays*.

CRIAÇÃO DE ARRAYS E ATRIBUIÇÃO DE VALORES

A linguagem PHP, por ser simples e bastante flexível, permite diferentes formas de declarar e atribuir valores em um *array*. Não existe uma maneira melhor do que a outra – qualquer uma delas pode ser utilizada. Nesse sentido, cabe destacar a última forma: **ela é bastante útil em situações nas quais não sabemos o tamanho do *array* ou a quantidade de itens que ele receberá.**

Ao estudar os exemplos, repare, ainda, nos comentários inseridos no código. Vamos a eles:

```
<?php
```

```
//Declarando um array vazio
```

```
$carros = array();
```

```
//Primeira forma de declaração e atribuição de valores
```

```
$carros = Array("Fusca", "Monza", "Passat");
```

```
//Segunda forma de declaração e atribuição de valores
```

```
//Esta forma é semelhante à anterior, mas utilizando sintaxe similar a do Javascript
```

```
$carros = ["Fusca", "Monza", "Passat"];
```

```
//Terceira forma de declaração e atribuição de valores
```

```
$carros[0] = "Fusca";
```

```
$carros[1] = "Monza";
```

```
$carros[2] = "Passat";
```

```
//Quarta forma de declaração e atribuição de valores
```

```
$carros[] = "Fusca";
```

```
$carros[] = "Monza";
```

```
$carros[] = "Passat";
```

Os **arrays** anteriores são **numéricos**. Vejamos outros exemplos, agora com vetores **associativos**. Repare que a principal diferença é a utilização de **strings** no lugar de números para definir os seus índices.

```
1 <?php
```

```
2 //Primeira forma
```



```

3 $carros = array (
4 'vw'      => "Fusca",
5 'chevrolet' => "Monza",
6 'fiat'     => "Tempra"
7 );
8
9 //Segunda forma
10 $carros = [
11 'vw'      => "Fusca",
12 'chevrolet' => "Monza",
13 'fiat'     => "Tempra"
14 ];
15
16 //Terceira forma
17 $carros['vw']      = "Fusca";
18 $carros['chevrolet'] = "Monza";
19 $carros['fiat']     = "Tempra";

```

Por fim, veremos exemplos de **array** com índices **numéricos e associativos**. Repare que a sintaxe é parecida com a vista na declaração dos associativos, ou seja, cada par “**índice/valor**” é separado por ‘=>’. Veja a seguir o exemplo e teste você mesmo o código em questão. Declare o **array**, seus valores e depois imprima na tela. Vamos ao código:

```

1 <?php
2 $carros = array (
3 'vw'      => "Fusca",
4 0         => "Passat",

```

```
5 'chevrolet' => "Monza",
6 1          => "Chevette",
7 'fiat'     => "Tempra",
8 2          => "Uno"
9 );
10 //A impressão desse array gera a seguinte saída
11 //Faça você mesmo: utilize a função print_r($carros); e veja o resultado em sua tela.
12 /*
13 Array
14 (
15 [vw] => Fusca
16 [0] => Passat
17 [chevrolet] => Monza
18 [1] => Chevette
19 [fiat] => Tempra
20 [2] => Uno
21 )
22 */
```

As formas vistas nos exemplos anteriores são as mais simples para a criação e inserção de elementos. Entretanto, a linguagem oferece outras formas, através do uso de funções como a ***array_push*** (que adiciona elementos ao final de um ***array***) e a ***array_unshift*** (adiciona elementos no início de um ***array***).

Além dessas, há funções que permitem gerar novos vetores, combinar vetores existentes e muito mais. Pesquise nos sites indicados no Explore+.

REMOÇÃO DE ELEMENTOS DE UM *ARRAY*

Há algumas formas de remover elementos de um *array*.

1

2

A primeira é definindo o valor do elemento como vazio. Nesse caso, embora o valor do elemento seja removido, o seu índice permanece no vetor, que também mantém o seu tamanho inicial.

Outra forma é fazendo uso de duas funções: *unset* e *array_splice*.

O código a seguir – que como os demais é totalmente funcional e, portanto, deverá ser executado por você – contém exemplos de utilização das formas apresentadas. Seguem algumas observações sobre o código e o uso das funções:

“*print_r*”

A função “*print_r*” imprime os elementos de um *array*.

“*count*”

A função “*count*” retorna o tamanho (quantidade de elementos) de um *array*. Essa função, inclusive, é muito útil quando trabalhamos com vetores.

“*unset*”

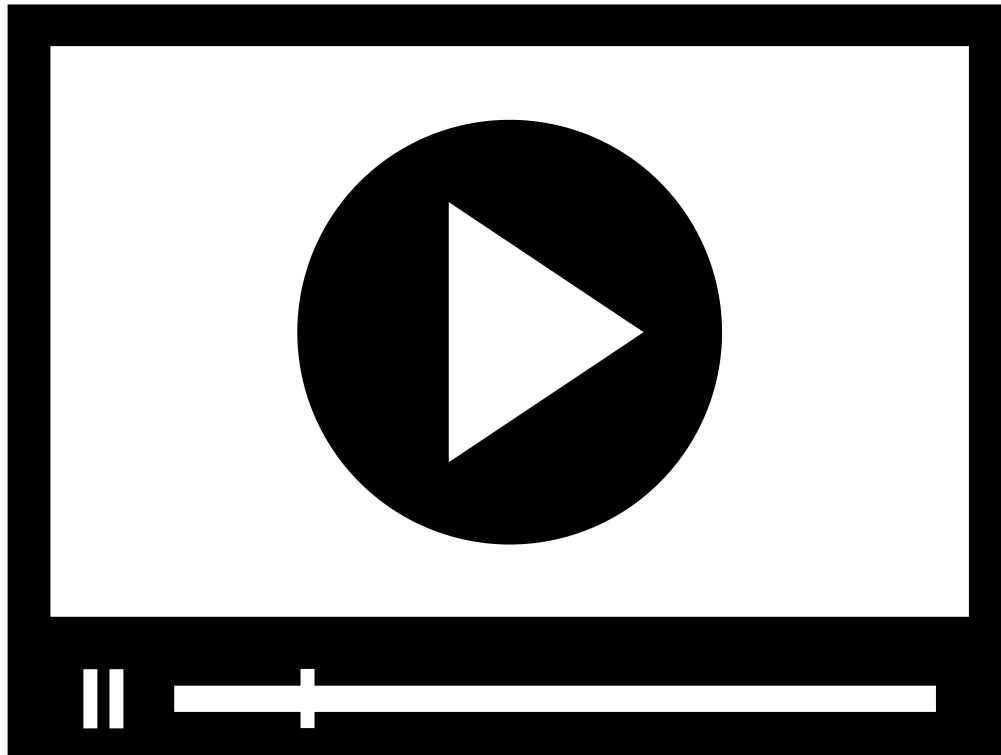
A função “***unset***” recebe como parâmetro o ***array*** e índice ou índices que desejamos remover. Além disso, é possível também remover o vetor inteiro, passando-o como parâmetro e sem definir nenhum índice.

“***array_splice***”

A função “***array_splice***” recebe como parâmetros o ***array*** a ser manipulado, o ***offset*** (índice a partir do qual desejamos excluir elementos) e o ***length*** (quantidade de itens que queremos excluir).

```
1 <?php
2 $carros = array (
3 'vw'      => "Fusca",
4 0         => "Passat",
5 'chevrolet' => "Monza",
6 1         => "Chevette",
7 'fiat'    => "Tempra",
8 2         => "Uno"
9 );
10 print_r($carros);
11 echo "O tamanho atual do array é: " . count($carros);
12 echo "\n\n";
13 //Definindo o valor do índice 0 como vazio
14 $carros[0] = "";
15
16 print_r($carros);
17 echo "O tamanho atual do array é: " . count($carros);
18 echo "\n\n";
19
```

```
20 //Removendo dois elementos do array com unset
21 unset($carros['fiat'], $carros[1]);
22
23 print_r($carros);
24 echo "O tamanho atual do array é: " . count($carros);
25 echo "\n\n";
26
27 //Removendo elementos do array com array_splice
28 array_splice($carros, 1,2);
29
30 print_r($carros);
31 echo "O tamanho atual do array é: " . count($carros);
```



MANIPULAÇÃO DE ELEMENTOS DE ARRAY NUMÉRICO, ARRAY ASSOCIATIVO E ARRAY MISTO

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



ARRAYS MULTIDIMENSIONAIS

O **array** que será visto na letra “b”, a seguir, é **multidimensional**. Ou seja, ele é composto por mais de uma dimensão – nesse caso, duas. Olhando para o exemplo, temos:

Dois índices principais, associativos, cujas chaves são ‘vermelhas’ e ‘cítricas’.

Cada uma dessas chaves possui um novo **array**, numérico, que contém quatro elementos.

Essas estruturas de dados são similares às vistas na disciplina de Matemática, onde temos também as **matrizes**, que podem ser chamadas de **vetores de vetores**. Trazendo esse conceito ao contexto em questão, temos um **array de arrays**, ou **um array dentro de outro array**.

TEORIA NA PRÁTICA

A melhor forma de fixar conteúdos quando se aprende uma linguagem de programação é praticando os conceitos estudados. Para isso, utilize seu ambiente AMP local ou um editor de código online e faça o seguinte:

Crie um ***array*** para armazenar frutas de acordo com seu tipo:

Frutas vermelhas: melancia, cereja, framboesa e morango.

Frutas cítricas: laranja, limão, abacaxi e mexerica.

Frutas tropicais: goiaba, maracujá, banana e manga.

Crie um novo ***array*** cujo resultado deverá ser:

RESOLUÇÃO

```
Array
(  
[vermelhas] => Array  
(  
[0] => melancia  
[1] => cereja  
[2] => framboesa  
[3] => morango  
)  
[citricas] => Array  
(
```



```
[0] => laranja  
[1] => limão  
[2] => abacaxi  
[3] => mexerica  
)  
)
```

FUNÇÕES

Funções, em linguagens de programação, são pedaços de código, encapsulados, que podem ser chamados em qualquer outro trecho do programa ou do código. Em relação à sua sintaxe, uma função deve ter **um nome, uma definição e uma posterior invocação à mesma**.

Em uma linguagem, as funções podem ser **nativas** – como a função “***print_r***”, utilizada em alguns exemplos anteriores – ou **construídas pelo desenvolvedor** – também chamadas de funções definidas pelo usuário.

DICA

Em termos práticos, pense nas funções como um código criado para resolver problemas singulares ou executar tarefas específicas. Além disso, tenha em mente que esses códigos poderão ser usados mais de uma vez ao longo do seu projeto. Logo, em vez de reescrever um mesmo código, faça uso de funções.

FUNÇÕES EM PHP

O exemplo a seguir descreve a sintaxe de criação de uma função em PHP e a forma de invocá-la.

```
1 <?php
2 $num1 = 10;
3 $num2 = 15;
4
5 $num3 = soma($num1,$num2);
6 imprimir_resultado($num3);
7
8
9 function soma($numero1, $numero2)
10 {
11 return $numero1 + $numero2;
12 }
13 function imprimir_resultado($numero)
14 {
15 echo "O resultado da operação é igual a: " . $numero;
16 }
```

Como visto no exemplo (aproveite para executá-lo antes de continuar a leitura), a sintaxe de uma função contém os seguintes elementos:

Palavra reservada “**function**” seguida do nome da função.

Nome da função seguido de parênteses – “()”. Caso receba parâmetros, eles deverão ser declarados dentro dos parênteses. Do contrário, deverão ficar sem conteúdo.

Instruções da função envoltas em chaves – “{}”.

As funções em PHP podem ou não retornar resultados. A primeira delas, “**soma**”, através do operador “**return**”, devolve o resultado da soma entre os dois parâmetros recebidos. Repare que a variável “\$num3” recebe justamente esse resultado. Já a função “imprimir_resultado” não retorna valores, apenas imprimindo uma frase na tela.

ATENÇÃO

Outra particularidade em PHP é que **as funções não precisam estar definidas para serem invocadas**. Repare que chamamos as duas antes mesmo de codificá-las. Devemos, porém, nos atentar para a quantidade de parâmetros a serem passados. Além disso, sua sintaxe é simples: “**nome-da-funcao(parâmetros)**” ou “**nome-da-funcao()**”.

NOMENCLATURAS DE FUNÇÕES E OUTRAS BOAS PRÁTICAS

A nomeação de funções em PHP segue as mesmas regras para a definição de variáveis, com alguns padrões utilizados e que são considerados como boas práticas. As mesmas dicas cabem, portanto, aqui:

VOCÊ PODE CRIAR NOMES DE FUNÇÕES SEPARANDO NOMES COMPOSTOS POR UNDERSCORE “_” OU COMO CAMELCASE, POR EXEMPLO. DEFINA UM PADRÃO E SIGA-O

POR TODO O SEU CÓDIGO.

Sobre a sintaxe da função, mais precisamente sobre o posicionamento da chave de abertura, há duas vertentes defendendo que:

A chave de abertura deve ser inserida logo após o fechamento dos parênteses que guardam os parâmetros da função.



A chave de abertura deve ser inserida na linha seguinte – essa foi a aplicada nos códigos anteriores.

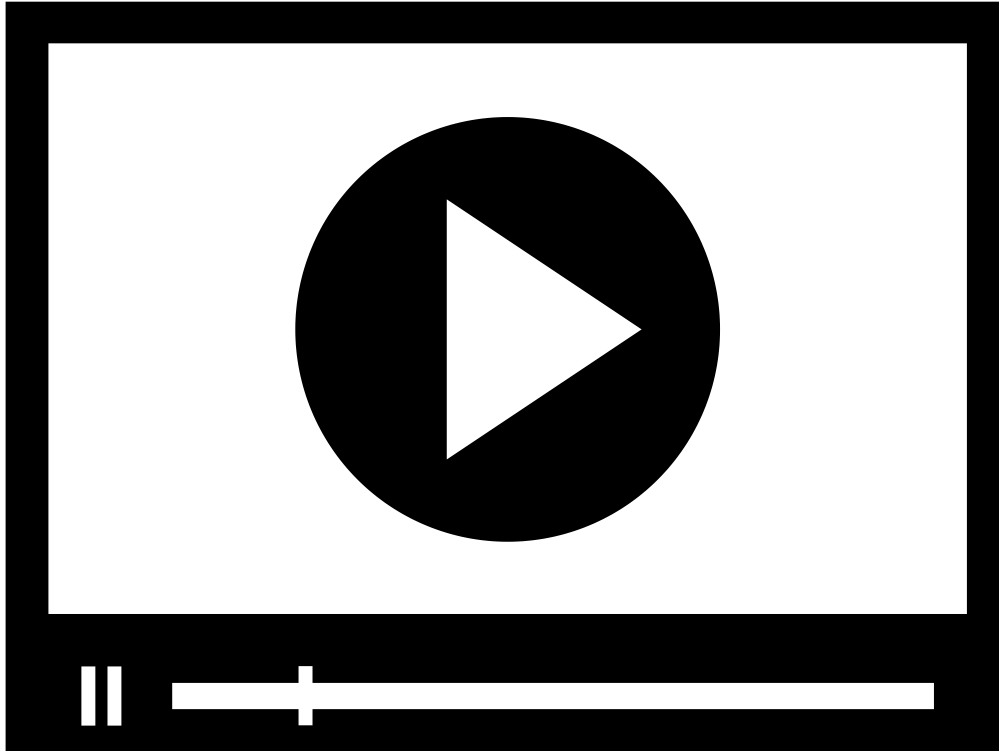
Por fim, outra boa prática recomendada: **indente seu código** – não só nas instruções inseridas dentro das funções, mas ao longo de todo o programa. Veja o código de exemplo e perceba que as instruções dentro da função não estão coladas no início da linha. **Indentar um código ajuda na sua compreensão e no seu entendimento, além de deixar clara a hierarquia existente.**

FUNÇÕES NATIVAS

A linguagem PHP disponibiliza uma série de **funções nativas**, para as mais variadas necessidades. Há funções para a manipulação de **arrays** (como as que vimos nos exemplos anteriores), de **strings**, de arquivos, de acesso de banco de dados, entre tantas outras. A documentação oficial da linguagem ou Manual do PHP é uma fonte extensa e que deve ser sempre consultada.

 **SAIBA MAIS**

A Zend, fabricante norte-americana de software orientado para PHP, é uma das empresas mais conhecidas e relevantes sobre o assunto. Além de um Framework bastante conhecido, que leva o seu nome, ela também é responsável pela certificação profissional PHP mais importante do mercado. Entre todo o material que disponibiliza, há um **manual de boas práticas** com uma série de convenções relacionadas à produção de código em PHP. Vale a pena a leitura desse material, conforme sugerido na seção Explore +.



No vídeo a seguir, o professor Alexandre Paixão nos apresenta alguns exercícios, demonstrando o uso de vetores e funções em PHP. Vamos assistir!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



VERIFICANDO O APRENDIZADO

1. EM RELAÇÃO AOS CONCEITOS DE VETORES, ASSINALE A ALTERNATIVA INCORRETA:

- A) Vetores podem ser vistos como pilhas ou filas de itens, onde um novo elemento é, naturalmente, incluído ao seu final.
- B) Podemos incluir novos elementos em qualquer posição de um *array*, seja no início, meio ou fim.
- C) Um vetor pode conter diferentes tipos de dados, incluindo até mesmo outros vetores.
- D) Um *array* multidimensional é um vetor que possui tanto índice numérico quanto associativo.

2. AS FUNÇÕES SÃO UM PODEROSO RECURSO DISPONÍVEL NAS LINGUAGENS DE PROGRAMAÇÃO. SOBRE OS CONCEITOS E A SINTAXE DAS FUNÇÕES NA LINGUAGEM PHP, ASSINALE A ALTERNATIVA CORRETA:

- A)** Uma função que não recebe parâmetros não poderá, nunca, retornar resultado.
- B)** Não é possível declarar variáveis dentro do escopo de uma função. Com isso, só estarão disponíveis as recebidas como parâmetro.
- C)** Em PHP, diferentemente de outras linguagens, uma função pode ser invocada antes de ser codificada.
- D)** Em PHP é possível iniciar o nome de uma função utilizando qualquer caractere, inclusive números.

GABARITO

1. Em relação aos conceitos de vetores, assinale a alternativa incorreta:

A alternativa **"D "** está correta.

Um *array* multidimensional é composto por mais de uma dimensão. Em outras palavras, um *array* que contém outro pode ser chamado de multidimensional.

2. As funções são um poderoso recurso disponível nas linguagens de programação. Sobre os conceitos e a sintaxe das funções na linguagem PHP, assinale a alternativa correta:

A alternativa **"C "** está correta.

As funções, em PHP, a exemplo do que ocorre com outros de seus recursos, possuem bastante flexibilidade. Dessa forma, é possível declarar novas variáveis dentro do seu escopo ou ter funções retornando ou não valores, independentemente de receberem ou não parâmetros. Além disso, elas podem ser invocadas antes mesmo de serem definidas. No entanto, é preciso ter atenção com algumas regras. O nome de uma função, por exemplo, deve seguir as mesmas normas para os nomes de variáveis.

CONCLUSÃO

CONSIDERAÇÕES FINAIS

Neste tema, foi apresentada a programação de páginas dinâmicas utilizando PHP, linguagem de script ***server side***. Ao longo de três módulos, através da explanação de conceitos e da aplicação de exemplos práticos e funcionais, vimos a criação de variáveis e atribuição de valores, alguns dos operadores disponíveis e a manipulação de variáveis recebidas de métodos HTTP POST e GET. Foram listadas, ainda, as estruturas de decisão e de repetição disponíveis na linguagem e, por fim, descritos os recursos de ***vetor/array*** e funções.

Para ouvir um *podcast* sobre o assunto, acesse a versão online deste conteúdo.



ANÁLISE DO TEMA

REFERÊNCIAS

PHP. **Manual do PHP**: O que é o PHP? Consultado em meio eletrônico em: 16 ago. 2020.

EXPLORE+

Para saber mais sobre JavaScript, leia o livro ***JavaScript: The Definitive Guide***, de David Flanagan.

Acesse o site Apache Friends para aprofundar seus conhecimentos sobre o XAMPP, o mais popular ambiente de desenvolvimento PHP.

Para testar seus códigos PHP, utilize os sites Online PHP Editor, PHPTester e Write PHP Online.

Acesse o Manual do PHP e leia os textos:

O que as referências fazem

Escopo de variáveis

Precedência de operadores

Operadores

for

break

continue

Visite o site da comunidade Mozilla e pesquise sobre os temas: HTTP; GET; e POST.

Acesse o site W3Schools e leia: ***The GET method*** e ***The POST method***.

Leia o manual ***Zend Framework coding standard for PHP***, da Zend Framework.

CONTEUDISTA

Alexandre de Oliveira Paixão

 **CURRÍCULO LATTES**