

Tomada de decisões e repetições

Com a estrutura `if()`, você pode ter instruções em seu programa que só serão executadas se determinadas condições forem verdadeiras. Isso permite que o programa execute diferentes ações dependendo das circunstâncias. Por exemplo, você pode verificar se um usuário inseriu informações válidas em um formulário web antes de permitir que ele veja dados sigilosos.

A estrutura `if()` executará um bloco de código se sua expressão de teste for verdadeira. Isso é demonstrado no Exemplo 3.1.

```
if($logged_in)
{
    print "Welcome aboard, trusted user.";
}
```

A instrução `if()` avalia o valor de verdade da expressão que está dentro de seus parênteses (a expressão de teste). Se a expressão for avaliada como verdadeira, as instruções dentro das chaves após `if()` serão executadas. Se a expressão não for verdadeira, o programa continuará nas instruções após as chaves. Nesse caso, a expressão de teste é composta apenas pela variável `$logged_in`. Se `$logged_in` for verdadeira (ou tiver um valor que seja avaliado como verdadeiro, como 5, -2, 12.6 ou Grass Carp), a mensagem `Welcome aboard trusted user`, será exibida.

Você pode ter quantas instruções quiser no bloco de códigos dentro das chaves. No entanto, é preciso terminar todas elas com um ponto e vírgula. Essa regra é a mesma aplicada a códigos existentes fora de uma instrução `if()`. Porém, o símbolo de ponto e vírgula não é necessário após a chave de fechamento que delimita o bloco de código. Também não é necessário colocar ponto e vírgula após a chave de abertura. O exemplo 3.2 mostra uma cláusula `if()` que executa várias instruções quando sua expressão de teste é verdadeira.

```
<?php
$logged_in = false;
print "This is always printed
";
if($logged_in)
{
    print "Welcome aboard trusted user";
    print 'This is only print if $logged_in is true.';
}
print "This is also always printed";
?>
```

Para executar uma instrução diferente quando a expressão de teste `if()` for falsa, adicione uma

cláusula `else` à sua instrução `if()`. Isso é mostrado no Exemplo 3.3.

```
<?php
$logged_in = 'full';
if($logged_in)
{
    print "Welcome aboard, trusted user";
}
else
{
    print "Howdy, stranger";
}
?>
```

No Exemplo 3.3, a primeira instrução `print` só é executada quando a expressão de teste de `if()` (a variável `$logged_in`) é verdadeira. A segunda instrução `print`, localizada dentro da cláusula `else`, só executada quando a expressão de teste é falsa.

As estruturas `if()` e `else` podem se estendidas com o uso da estrutura `elseif()`. Você pode usar uma ou mais cláusulas `elseif()` com uma instrução `if()` para testar várias condições separadamente. O Exemplo 3.4 demonstra `elseif()`.

```
<?php
$logged_in = false;
$new_messages = '1';
$emergency = '4';
if($logged_in){
    // Essa instrução será executada se $logged_in for verdadeira
    print "Welcome aboard, trusted user.";
}elseif($new_messages){
    //Essa instrução será executada se $logged_in for falsa, mas $new_messages for verdadeira
    print "Dear stranger, there are new messages";
}elseif($emergency){
    //Essa instrução será executada se if $logged_in e new messages forem falsas,
    //mas $emergency for verdadeira
    print "stranger, there are no new messages, but there is an emergency.";
}
?>
```

Se a expressão de teste da instrução `if()` for verdadeira, o engine PHP executará as instruções existentes dentro do bloco de código situado após `if()` e ignorará as cláusulas `elseif()` e seus blocos de códigos. Se a expressão de teste da instrução `if()` for falsa, o engine passará para a primeira instrução `elseif()` e aplicará a mesma lógica. Se essa expressão de teste for verdadeira, ele executará o bloco de código dessa instrução `elseif()`. Se for falsa, o engine passará para a próxima `elseif()`.

Para um conjunto específico de instruções `if()` e `elseif()`, no máximo um dos blocos de código

será executado: o bloco de código da primeira instrução cuja expressão de teste for verdadeira. Se a expressão de teste da instrução `if()` for verdadeira, nenhum dos blocos de código de `elseif()` será executado, nem mesmo se suas expressões de teste forem verdadeiras. Quando uma das expressões de teste de `if()` ou de `elseif()` for verdadeira, o resto será ignorado. Se nenhuma das expressões de teste das instruções `if()` e `elseif()` for verdadeira, nenhum bloco de código será executado.

Você pode usar `else` com `elseif()` para incluir um bloco de código para ser executado se nenhuma das expressões de teste de `if()` ou `elseif()` forem verdadeiras. O Exemplo 3.5 adiciona uma instrução `else` ao código do Exemplo 3.4.

```
<?php
$logged_in = false;
$new_messages = false;
$emergency = false;
if($logged_in){
    // Essa instrução será executada se $logged_in for verdadeira
    print "Welcome aboard, trusted user.";
}elseif($new_messages){
    //Essa instrução será executada se $logged_in for falsa, mas $new_messages for verdadeira
    print "Dear stranger, there are new messages";
}elseif($emergency){
    //Essa instrução será executada se if $logged_in e new messages forem falsas,
    //mas $emergency for verdadeira
    print "stranger, there are no new messages, but there is an emergency.";
}else{
    //Essa instrução será executada se $logged_in, $new_messages e emergency forem falsas
    print "I don't know you, you have no messages, and there's no emergency";
}
?>
```

Construindo decisões complicadas

Os operadores lógicos e de comparação do PHP nos ajudam a compor expressões mais complicadas em que uma estrutura `if()` toma decisão. Esses operadores permitem comparar valores, negar igualdade de valores e encadear várias expressões dentro de uma única instrução `if()`.

O operador de igualdade é representado pelo símbolo `==` (dois sinais de igualdade). Ele retorna verdadeiro quando os valores testados são iguais. Os valores podem ser variáveis ou literais. Alguns usos do operador de igualdade são mostrados no Exemplo 3.6.

```
<?php
$new_messages = 10;
$max_messages = 10;
```

```
$max_messages = 10;
$dinner = false;
if($new_messages == 10)
{
    print "You have ten new messages";
}
if($new_messages == $max_messages)
{
    print "You have the maximum number of messages";
}
if($dinner == "Braised Scallops")
{
    print "Yum! I love seafood";
}
?>
```

Atribuição *versus* Comparação

Tome cuidado para não usar = quando na verdade queria usar ==. Um único sinal de igualdade atribui um valor e retorna o valor atribuído. O sinal duplo verifica se existe igualdade e retorna verdadeiro se os valores forem iguais. Quando não incluímos o segundo sinal de igualdade, geralmente obtemos um teste if() que sempre é verdadeiro, como na instrução a seguir:

```
if($new_messages = 12){

print "It seems you now have twelve messages.";

}
```

Em vez de verificar se \$new_messages é igual 12, o código mostrado aqui configura \$new_messages com 12. Essa atribuição retorna 12, o valor que está sendo atribuído. A expressão de teste if() é sempre verdadeira, independentemente do valor de \$new_messages. Além disso, o valor de \$new_messages é sobreposto. Uma maneira de evitar o uso de = no lugar de == é inserir a variável no lado direito da comparação e o literal no lado esquerdo, como mostrado a seguir:

```
if(12 == $new_messages){

print "You have twelve new messages";

}
```

Essa expressão de teste pode parecer um pouco estranha, mas dá alguma segurança caso acidentalmente você use = em vez de ==. Com um único sinal de igualdade, a expressão de teste é 12 = \$new_messages, que significa "atribua o valor \$new_messages a 12". Não faz nenhum sentido: você não pode alterar o valor de 12. Se o engine PHP encontrar essa expressão em seu programa, ele relatará um erro de análise e o programa não será executado. O erro de análise nos alerta para o fato de estar faltando um =. Com o literal

no lado direito da expressão, o código será analisado pelo engine, logo ele não relatará um erro.

O oposto do operador de igualdade é `!=`. Ele retornará verdadeiro se os dois valores que você testar usando-o não forem iguais, Consulte o Exemplo 3.7.

```
<?php
if($new_messages != 10)
{
    print "You don't have ten new messages";
}
if($dinner != 'Braised Scallops')
{
    print "I guess we're out of Scallops";
}
?>
```

Com o operador menor que (`<`) e o operador maior que (`>`), você pode comparar quantidades. Os operadores `<=` ("menor igual a") e `>=` ("maior ou igual a") são semelhantes a `<` e `>`. O Exemplo 3.8 mostra como usar esses operadores.

```
<?php
if($age > 17){
    print "You are old enough to download the movie";
}
if($age >= 65){
    print "You are old enough for a discount";
}
if($celsius_temp <=0){
    print "Uh-no, your pipes may freeze";
}
if($kelvin_temp < 20.3){
    print "Your hydrogen is a liquid or a solid now";
}
?>
```

Como mencionado em "números", na página 52, os números de ponto flutuante são armazenados internamente de tal forma que podem ser um pouco diferentes que os valores que foram atribuídos. Por exemplo, 50.0 poderia ser armazenado internamente como 50.00000002. Para verificar se dois números de ponto flutuante são iguais, examine se sua diferença é menor que algum limite aceitavelmente baixo em vez de usar o operador de igualdade. Por exemplo, se estiver comparando quantias financeiras, um limite aceitável seria 0.00001. O Exemplo 3.9 demonstra como comparar dois números de ponto flutuante.

```
if(abs($price1 - $price2) < 0.00001){
    print '$price1 and $price2 are equal'
}
else{
```

```
        print '$price1 and $price2 are not equal';
    }
```

A função `abs()` usada no Exemplo 3.9 retorna o valor absoluto de seu argumento. Com `abs()`, a comparação funciona apropriadamente se `$price_1` for maior que `$price_2` ou `$price_2` for maior que `$price_1`.

Repetindo-se

Quando um programa de computador faz algo repetidamente, esse evento se chama *looping*. Isso ocorre com muita frequência - por exemplo, quando queremos recuperar um conjunto de linhas de um banco de dados, exibir linhas de uma tabela HTML ou exibir elementos de um menu HTML `<select>`. As duas estruturas de loop discutidas nessa seção são `while()` e `for()`. Elas são diferentes em suas particularidades, mas ambas requerem a especificação dos dois atributos essenciais de qualquer loop: O código que será executado repetidamente e quando parar. O código a ser executado é um bloco como o que é incluído em chaves após uma estrutura `if()`. A condição de interrupção do loop é uma expressão lógica como a expressão de teste também de uma estrutura `if()`.

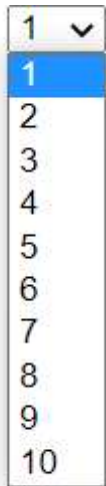
A estrutura `while()` é como uma instrução `if()` que se repete. Uma expressão é fornecida para `while()`, como ocorreria com `if()`. Se a expressão for verdadeira, um bloco de código será executado. Ao contrario de `if()`, no entanto, `while()` verifica a expressão novamente após executar o bloco de código. Se ela ainda for verdadeira, o bloco de código será executado mais uma vez (e continuará sendo executado enquanto a expressão for verdadeira). Quando a expressão for falsa, a execução do programa prosseguirá nas linhas após o bloco de código. Como você deve ter percebido, o bloco de código precisa fazer algo que altere o resultado da expressão de teste para que o loop não seja infinito.

O Exemplo 3.17 usa `while()` para exibir um menu `<select>` de 10 opções em um formulário HTML.

```
<?php
$i = 1; //$i é configurado como 1
print '<select name="people">';
while($i <=10) //expressão de teste compara $i a 10. Enquanto $i for menor ou igual a 10
    //as duas instruções do bloco de código serão executadas
{
    print "<option>$i</option>";
    // ...
}
```

```
;; //primeira instrucao imprime uma tag HTML
    $i++; //segunda instrucao incrementa a variavel $i, se não incrementassemos $i
} //a primeira instrução seria executada infinitamente.
print '</select>';
?>
```

O exemplo exibe:



1	▼
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	