CHƯƠNG 2

Ngôn ngữ PHP

Các chủ đề chính

Mục tiêu

Sau khi hoàn thành chương này, chúng ta sẽ có thể:

- Trình bày được ý nghĩa của Hằng và cách khai báo Hằng trong PHP.
- Trình bày được ý nghĩa của Biến, cách khai báo Biến trong PHP.
- Phân biệt được sự khác nhau cơ bản giữa Hằng và Biến trong PHP.
- Phân tích được sự khác nhau cơ bản giữa các toán tử trong PHP.
- Ứng dụng Biến, Hằng, Toán tử để áp dụng vào một chương trình cụ thể trong PHP.

Câu hỏi kiểm tra mở đầu

Trả lời các câu hỏi sau

1. Trong các ngôn ngữ lập trình, chúng ta sử dụng Hằng để?
   a. Lưu trữ các giá trị không đổi.
   b. Lưu trữ các giá trị có thể thay đổi được.
   c. Cả (a) và (b).
2. Trong C++ câu lệnh s /= i; tương đương với?
   a. S = i / s;
   b. S = s / i;
   c. Câu lệnh sai.
   d. S = i / i;
3. Trong các ngôn ngữ lập trình phép toán 7%2 cho chúng ta kết quả?
   a. 3
   b. 1

c. 2

d. 0

4. Cho biết kết quả của mã lệnh sau nếu chúng ta sử dụng ngôn ngữ C++?

```
int a = 5;
if(a = 5){
    a++;
    Cout << a;
}
```

a. 6

b. 5

c. 0

d. Không hiển thị kết quả.

5. Nếu ta có 8 / 4 / 2 thì kết quả sẽ là 1. Vậy theo bạn với mã lệnh PHP sau kết quả sẽ là bao nhiêu?

```
$h = 8; $i = 4; $j = 2;
$h /= $i /= $j;
echo($h);
```

a. 1

b. 2

c. 4

d. 6

1. Biến, hằng và kiểu dữ liệu

Trong chương 1, chúng ta đã biết qua về biến PHP và đã thấy một cách ngắn gọn về việc sử dụng chúng như thế nào. Như chúng ta đã biết, các biến PHP phải bắt đầu với ký tự dollar ($) và PHP là một ngôn ngữ loại yếu, các biến có thể bao gồm kiểu dữ liệu bất kỳ và không phải bị giới hạn trước là một chuỗi ký tự, số nguyên, vv. Chúng ta cũng đã thấy chúng ta có thể sử dụng các biến PHP để trích dữ liệu từ một Form HTMl như thế nào.

Trong phần này, chúng ta sẽ tìm hiểu chi tiết hơn về biến và kiểu dữ liệu. Chúng ta sẽ xem xét đầy đủ về các kiểu dữ liệu chi tiết hơn và chúng ta sẽ tìm hiểu một số hàm để chúng ta thao tác các biến. Chúng ta cũng sẽ thấy cách gán một tên tới một giá trị Hằng như thế nào.

1.1 Kiểu dữ liệu

PHP có ba kiểu dữ liệu cơ bản: `integer, double` và `string`. Cũng có những kiểu dữ liệu không cơ bản cụ thể là mảng `(arrays)` và đối tượng `(objects)`, chúng sẽ được thảo luận trong phần sau. Tất cả các biến có một kiểu xác định, Chắc chúng ta còn nhớ, kiểu của biến có thể thay đổi trong chương trình khi giá trị của biến thay đổi.

`Integer` sử dụng 4 bytes của bộ nhớ, giá trị của nó trong khoảng `-2` tỷ đến `+2` tỷ. Kiểu `Double` là kiểu số thực, phạm vi biểu diễn $\pm(10^{-308} \div 10^{308})$. Kiểu `string` dùng để chứa các giá trị bao gồm các ký tự và số.

```
    2           // Đây là số nguyên
    2.0         // Đây là số thực
```

```
"2"         // Đây là chuỗi ký tự
"2 hours"   // Đây là chuỗi ký tự khác
```

Nhiều ngôn ngữ bao gồm một kiểu dữ liệu Boolean để tương ứng với các giá trị logic TRUE và FALSE. PHP thì không. Nó sử dụng các biểu thức của ba kiểu cơ bản khác để đán giá các giá trị là đúng hoặc sai. Giữa các số nguyên, 0 được đánh giá là một giá trị sai và các số nguyên khác không được đánh giá là giá trị đúng. Cũng như vậy, giá trị số thực 0.0 (hoặc tương đương, chẳng hạn 0.0000) được đánh giá là FALSE và các giá trị khác 0 được đánh giá là TRUE. Giữa các chuỗi ký tự, chuỗi rỗng được đánh giá là FALSE. Chuỗi rỗng được mô tả là một cặp dấu nháy kép: “”. Các chuỗi không rỗng được đánh giá là TRUE.

1.2 Hằng

Hằng là là đại lượng có giá trị không đổi. Chúng ta thường dùng Hằng để lưu các giá trị không đổi trong suốt chương trình như: nhiệt độ ($0°C$), pi (gần bằng 3.14) và giá trị của “noon” (12:00). Trong ngôn ngữ lập trình, có hai loại hằng: hằng chữ và hằng biểu tượng. Hằng chữ là các giá trị không đổi đơn giản để được tham chiếu trực tiếp, không sử dụng từ định danh.

Khi chúng ta sử dụng “hằng”, chúng ta thường tham chiếu tới hằng biểu tượng. Hằng biểu tượng là một cách thuận lợi để gán một giá trị một lần tới một định danh và sau đó tham chiếu tới nó bằng cách định danh trong suốt chương trình của chúng ta.

Ví dụ, tên của công ty chúng ta là giá trị hằng. Như vậy, chuỗi chữ “Phop’s Bicycles” sẽ có trong khắp chương trình của chúng ta, chúng ta có thể định nghĩa hằng có tên COMPANY với giá trị “Phop’s

Bicycles" và sử dụng hằng này để tham chiếu tới tên công ty trong suốt mã lệnh của chúng ta. Chú ý rằng tên hằng không giống tên biến, không bắt đầu bởi ký tự $.

Định nghĩa Hằng

Hàm *define()* được sử dụng để tạo Hằng:

```
define("COMPANY", "Phop's Bicycles");
define("YELLOW", "#FFFF00");
define("VERSION", 3);
define("NL", "<BR>\n");
```

Trong ví dụ trên, chúng ta định nghĩa một Hằng được gọi là NL để đại diện cho một thẻ ngắt dòng HTML. Về cơ bản, chúng ta đã tạo ra một đường tắt mã hóa khi "
\n" là một sự kết hợp thường được sử dụng phổ biến. Với qui ước này, những người lập trình định nghĩa các biến sử dụng tất cả các từ in hoa. Một biến có thể bao gồm số hoặc xâu bất kỳ. Khi một Hằng được định nghĩa, chúng có thể được sử dụng thay cho các giá trị của chúng.

```
echo("Employment at " . COMPANY . NL);
```

Điều này tương đương với:

```
echo("Employment at Phop's Bicycles<BR>\n");
```

Lưu ý rằng Hằng xuất hiện ngoài dấu ngoặc kép. Dòng:

```
echo("Employment at COMPANY NL");
```

Sẽ hiển thị trên trình duyệt là: "Employment at COMPANY NL"

Defined()

Hàm Defined() cho phép chúng ta xác định có hay không một hằng tồn tại. Nó trả về 1 nếu hằng số tồn tại và 0 nếu hằng không tồn tại:

```
if (defined("YELLOW")) {
     echo ("<BODY BGCOLOR=" . YELLOW . ">\n");
}
```

Các hằng được xây dựng sẵn trong PHP

PHP có các hằng được dựng sẵn. TRUE và FALSE là được định nghĩa đầu tiên tương ứng với ccs giá trị là true (1) và false (0 hoặc chuỗi rỗng). hằng PHP_VERSION cho biết phiên bản của PHP mà hiện thời đang chạy, chẳng hạn 3.0.11. Hằng PHP_OS cho biết hệ điều hành phía server đang chạy.

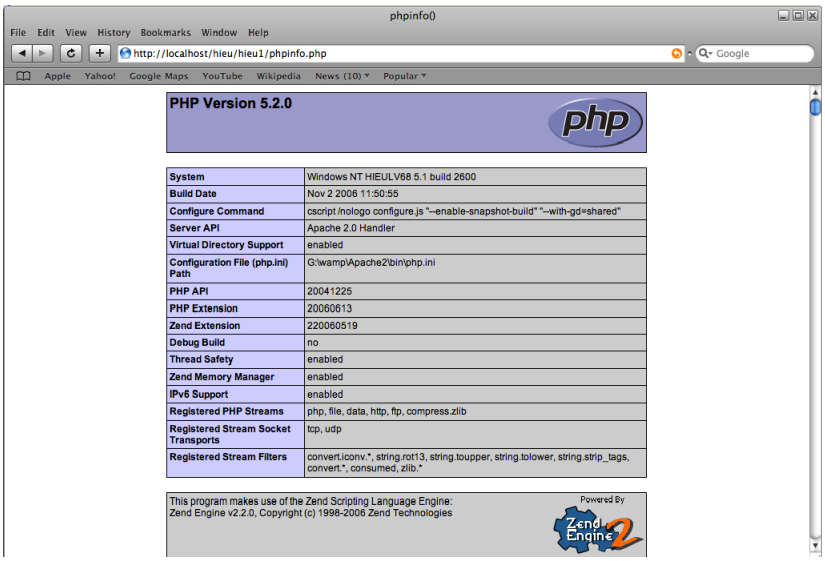
```
echo(PHP_OS);     // In ra "Linux" (ví dụ)
```

__FILE__ và __LINE__ giữ tên của kịch bản đang phân tích và số dòng hiện thời trong kịch bản. (Có hai ký tự đường gạch dưới trước và sau tên của các hằng này)

PHP cũng có một số hằng thông báo lỗi: E_ERROR, E_WARNING, E_PARSE và E_NOTICE.

```
<HTML>
<!-- phpinfo.php -->
     <BODY>
          <?php
               phpinfo()
          ?>
     </BODY>
</HTML>
```

Đoạn mã trên sẽ hiển thị:



1.3 Khai báo và khởi tạo biến

Cú pháp PHP chính là cú pháp trong ngôn ngữ C++, những ai làm quen với ngôn ngữ C++ thì có lợi thế trong lập trình PHP.

Khác với hằng, biến tự động được khai báo trong PHP khi chúng ta gán một giá trị tới nó. Việc gán giá trị rất đơn giản bằng cách sử dụng toán tử (=). Chú ý rằng toán tử (=) và (==) là khác nhau trong PHP. Chúng ta sẽ thấy điều này trong các phần tiếp theo.

```
$num_rows = 10;
$product = "Tire Pump";
$price = 22.00;
$shipping = 5.00;
$total = $price + $shipping;
```

Lừa kiểu và ép kiểu

Như đã đề cập phần trước, mọi biến PHP có một kiểu dữ liệu. Kiểu dữ liệu đó được quyết định một cách tự động bởi giá trị mà nó được gán cho biến.

```
$a = 1;      // $a is an integer
```

```
$a = 1.2;   // Now it's a double
$a = "A";   // Now it's a string
```

Khi chúng ta học những phần tiếp theo, cũng có nhiều cách để chỉ định rõ ràng kiểu dữ liệu của biến.

Chuyển đổi chuỗi và lừa kiểu

Nếu chúng ta thực hiện một số thao tác trên chuỗi ký tự, PHP sẽ đánh giá chuỗi ký tự là một số. Điều này được gọi là chuyển đổi chuỗi, mặc dù biến chứa chuỗi có thể không cần thiết thay đổi. Trong ví dụ sau, $str được gán một giá trị chuỗi ký tự:

```
$str = "222B Baker Street";
```

Nếu chúng ta cố gắng tăng thêm giá trị nguyên là 3 vào `$str`, `$str` sẽ đánh giá với số nguyên là `222` để phục vụ mục đích tính toán:

```
$x = 3 + $str;   // $x = 225;
```

Nhưng biến `$str` không thay đổi:

```
echo ($str);   // Prints: "222B Baker Street"
```

Chuyển đổi chuỗi đi theo sau một cặp luật:

- Chỉ phần đầu của chuỗi được đánh giá là số. Nếu chuỗi bắt đầu với giá trị số hợp lệ, chuỗi sẽ đánh giá là giá trị đó; ngoài ra nó sẽ đánh giá là 0. Chuỗi “3rd degree” sẽ đánh giá là 3 nếu sử dụng một toán tử số nhưng chuỗi “Catch 22” sẽ đánh giá là 0.

- Một chuỗi được đánh giá là số thực duy nhất nếu giá trị số thực được mô tả bao gồm toàn chuỗi ký tự. Các chuỗi “3.14”, “-4.01” và “4.2e6” sẽ được đánh giá là các số thực 3.4, -4.01 và 4.2000000. Tuy nhiên nếu các ký tự khác không phải là số thực có ở trong chuỗi, chuỗi sẽ đánh giá là số nguyên: “3.4 children” sẽ được đánh

giá là số nguyên 3. Chuỗi "-4.01 degree" sẽ được đánh giá là số nguyên -4.

Ngoài việc chuyển đổi chuỗi, PHP thực hiện lừa kiểu giữa hai kiểu số. Nếu chúng ta thực hiện một thao tác số giữa một số thực và một số nguyên, kết quả sẽ là một số thực:

```
$a = 1;           // $a là số nguyên
$b = 1.0;         // $b là số thực
$c = $a + $b;     // $c là số thực (value 2.0)
$d = $c + "6th";  // $d là số thực (value 8.0)
```

Ép kiểu

Ép kiểu cho phép chúng ta thay đổi dứt khoát kiểu dữ liệu của biến:

```
$a = 11.2;         // $a là số thực
$a = (int) $a      // Bây giờ nó là số nguyên (value 11)
$a = (double) $a   // Bây giờ nó trở lại là số thực (value 11.0)
$b = (string) $a   // $b là chuỗi ký tự (value "11")
```

Mảng và đối tượng cũng được phép. Integer là giống với int. float và real là giống với double.

Biến động

PHP hỗ trợ biến động (variable variables). Các biến thông thường có các giá trị động: Chúng ta có thể thiết lập và thay đổi giá trị của biến. Với biến động, tên của biến là động. Biến động thường tạo ra nhiều nhầm lẫn hơn là sự tiện lợi (đặc biệt là khi sử dụng mảng). Chúng ta thường ít sử dụng biến động vì trong thực tế chúng mang lại lợi ích rất ít. Đây là một ví dụ về biến động.

```
$field = "ProductID";
$$field = "432BB";
```

Dòng đầu tiên của đoạn mã trên tạo ra một biến $field và gán giá trị là "ProductID". Dòng thứ hai sử dụng giá trị của biến đầu tiên để tạo tên của biến thứ hai. Biến thứ hai có tên $ProductID và có giá trị "432BB". Hai dòng mã lệnh sau sẽ thực thi để hiển thị ra như thế này:

```
echo ($ProductID); // In ra: 432BB
echo ($$field);    // In ra: 432BB
```

1.4 Các hàm hữu ích cho biến

PHP có một số hàm được xây dựng sẵn để làm việc với biến.

gettype()

`gettype()` xác định kiểu dữ liệu của biến. Nó trả về một trong các giá trị sau:

- `"integer"`
- `"double"`
- `"string"`
- `"array"`
- `"object"`
- `"class"`
- `"unknown type"`

Chúng ta xem chi tiết hơn trên mảng, đối tượng và lớp trong các phần sau. Ví dụ về sử dụng `gettype()` có thể là:

```
if (gettype ($user_input) == "integer") {
```

```
    $age = $user_input;
    }
```

settype()

Hàm `settype()` thiết lập kiểu cho biến. Kiểu được viết là chuỗi và có thể là một trong các kiểu sau: `array, double, integer, object` hoặc `string`. Nếu kiểu không thể thiết lập thì một giá trị FALSE được trả về.

```
$a = 7.5;                    // $a là số thực
settype($a,"integer");//Bây giờ nó là số nguyên(giá trị 7)
```

`settype()` trả về một giá trị TRUE nếu quá trình chuyển đổi là thành công. Ngược lại, nó trả về FALSE.

```
    if (settype($a, "array")) {
    echo("Quá trình chuyển đổi thành công.");
    } else {
    echo ("Quá trình chuyển đổi lỗi."); }
```

isset() và unset()

unset() được sử dụng để hủy biến. Hàm isset() được dùng để xác định biến có tồn tại không. Nếu biến tồn tại thì nó trả về TRUE.

```
    $ProductID = "432BB";
    if (isset($ProductID)) {
        echo("Dòng này được in ra");
    }
    unset($ProductID);
    if (isset ($ProductID)) {
        echo("Dòng này sẽ không được in ra");
    }
```

Empty()

Empty() gần như đối ngược hoàn toàn với isset(). Nó trả về TRUE nếu biến không được thiết lập hoặc có một giá trị 0 hoặc một chuỗi rỗng. Ngoài ra nó trả về FALSE.

```
    echo empty($new);       // true
    $new = 1;
    echo empty($new);       // false
    $new = "";
    echo empty($new);       // true
    $new = 0;
    echo empty($new);       // true
    $new = "Buon giorno";
    echo empty($new);       // false
    unset ($new);
    echo empty($new);       // true
```

Hàm is…()

Hàm is_int(), is_integer() và is_long() là các hàm giống nhau để quyết định một biến là một số nguyên.

is_double(), is_float() và is_real()quyết định một biến là một số thực.

is_string(), is_array() và is_object() làm việc giống như các kiểu dữ liệu tương ứng của chúng.

```
    $ProductID = "432BB";
    if (is_string ($ProductID)) {
        echo ("String");
    }
```

Hàm …val()

PHP cung cấp cách khác để thiết lập rõ rang kiểu dữ liệu của biến: các hàm intval(), doubleval() và strval(). Các hàm này không sử dụng được để chuyển đổi mảng và đối tượng.

```
$ProductID = "432BB";
$i = intval($ProductID);       // $i = 432;
```

2. Các toán tử

Toán tử được sử dụng để xác định một giá trị bằng cách thực hiện một thủ tục hoặc một **thao tác.** Phép cộng là một trong những thao tác đơn giản nhất. Trong biểu thức 6 + 2, 6 and 2 là các toán hạng và biểu thức đánh giá là 8. Các toán tử trong PHP là khá giống với C, Perl và các ngôn ngữ liên quan. Phần này mô tả chúng một cách chi tiết.

2.1 Các toán tử số học

Giống như mọi ngôn ngữ lập trình, PHP sử dụng các toán tử toán học cơ bản:

Toán tử	Tên	Ví dụ	Mô tả
+	Phép cộng	7 + 2	Tính tổng của 7 và 2: 9
–	Phép trừ	7 – 2	Tính hiệu của 7 trừ 2: 5
*	Phép nhân	7 * 2	Tính tích của 7 và 2: 14
/	Phép chia	7 / 2	Phép chia của 7 và 2: 3.5
%	Chia lấy dư	7 % 2	Tính phần dư của phép chia 7 và 2: 1

PHP thường không để ý đến các ký tự trống. Mặc dù `$x = 6 * 2` và `$x = 6*2` là tương tương nhau.

2.2 Toán tử một ngôi

Ký hiệu (-) được sử dụng với một giá trị số đơn để phủ định một số (Nghĩa là một số dương sẽ thành âm và số âm sẽ thành dương). Ví dụ:

```
$a = 2;
$b = -$a; // $b = -2
$c = -4;
```

```
    $d = -$c; // $d = 4
```

2.3 Toán tử gán

Như chúng ta đã thấy trong mục 2, chúng ta sử dụng toán tử gán = để thiết lập các giá trị của biến.

```
$x = 1;
$y = x + 1;
$length = $area / $width;
$description = "Bicycle helmet";
```

Biến bên trái của dấu = đã lấy một giá trị của biểu thức bên phải dấu =. Điều quan trọng là không được nhầm lẫn giữa toán tử gán = với toán tử so sánh ==, chúng ta sẽ thấy điều đó trong phần tiếp theo.

2.4 Toán tử so sánh

Toán tử so sánh được dử dụng để kiểm tra điều kiện. Các biểu thức sử dụng các toán tử so sánh sẽ luôn đánh giá tới một giá trị Boolean, ví dụ là true hoặc false.

```
$i = 5;
if ($i < 6) echo ("Dòng này sẽ được in ra.");
//  biểu thức  '$i < 6'  đánh giá là 'true'
if ($i > 6) echo ("Dòng này se không được in ra.");
//  Biểu thức  '$i > 6'  đánh giá là 'false'
```

Chúng ta sẽ xem xét các câu lệnh `if` chi tiết trong chương sau. Các toán tử so sánh được tóm tắt trong bảng sau:

Toán tử	Nghĩa	Ví dụ	Đánh giá là true khi
==	Bằng nhau	$h == $i	$h và $i có giá trị bằng nhau
<	Nhỏ hơn	$h < $i	$h nhỏ hơn $i
>	Lớn hơn	$h > $i	$h lớn hơn $i
<=	Nhỏ hơn hoặc bằng	$h <= $i	$h nhỏ hơn hoặc bằng $i
>=	Lớn hơn hoặc bằng	$h >= $i	$h lớn hơn hoặc bằng $i
!=	Không bằng	$h != $i	$h không bằng $i
<>	Không bằng	$h <> $i	$h không bằng $i

Một lần nữa, hãy nhớ rằng toán tử so sánh là hai dấu bằng (==), dấu bằng đơn (=) đại diện cho toán tử gán. Toán tử gán được sử dụng để thiết lập giá trị của một biến, trong khi toán tử so sánh được sử dụng để xác định hoặc kiểm tra giá trị của một biến. Trong khi toán tử so sánh được sử dụng để quyết định hoặc kiểm tra giá trị của biến. Sự Thất bại để quan sát sự phân biệt này có thể dẫn tới những kết quả bất ngờ. Ví dụ, chúng ta có thể viết nhầm:

```
$i = 4;
if ($i = 7) echo ("seven");
// "seven" được in ra!
```

Đây là một luật hoàn toàn hợp lý trong PHP, do vậy chúng ta sẽ không thấy thông báo lỗi. Tuy nhiên, câu lệnh `if` này không kiểm tra giá trị của `$i` có bằng `7` không. Thực tế, nó gán giá trị `7` cho `$i` và sau đó đánh giá là `7`, nó là số khác `0` và do vậy `true`. Khi lỗi được phát sinh, sẽ rất khó để dò tìm ra. Một cách tổng quát, nếu chúng ta bắt gặp một câu

lệnh `if` mà luôn được đánh giá là `true` hoặc luôn được đánh giá là `false` bất chấp điều kiện, cơ hội tốt để chúng ta sử dụng = thay cho ==. Trong mã lệnh sau, chúng ta đã viết chính xác:

```
$i = 4;
if (7 == $i) echo ("seven");
// 7 == $i đánh giá là false, do vậy câu lệnh echo
        không thực thi
```

Ở đây chúng ta đã sử dụng toán tử gán bằng toán tử so sánh. Ngoài ra, chúng ta đã đặt giá trị thật bên trái và biến bên phải. Thói quen này tạo cho nó nhiều khó khăn để tạo ra lỗi trong tương lai: Nếu chúng ta viết sai `7 = $i`, PHP sẽ cố gắng gán giá trị của biến `$i` là 7. Điều này rõ ràng là không thể xảy ra, do vậy lỗi sẽ được tạo ra:

Chú ý rằng lừa kiểu và chuyển đổi kiểu xảy ra trong các so sánh; điều này có nghĩa là nếu hai biến có giá trị giống nhau sau khi chuyển đổi kiểu, PHP sẽ xem xét chúng để có các giá trị giống nhau dù chúng có các kiểu dữ liệu khác nhau. Ví dụ:

```
echo ("7" == 7.00);
```

Mã này hiển thị ra số 1 vì biểu thức “7” == 7.00 đánh giá là true. Trong hầu hết các trường hợp thực tế, đây không phải là kết quả. Nếu chúng ta cần tạo ra một sự khác biệt giữa một biến chứa “7” và một biến chứa “7.00”, chúng ta sẽ phải so sánh cả các giá trị và các kiểu của các biến:

```
$a = "7";
$b = 7.00;
echo ($a == $b);                    // In ra 1 (true)
echo (($a == $b) and (gettype ($a) == gettype ($b)));
    // In ra 0 (false)
```

2.5 Toán tử logic

Toán tử logic được dùng để kết hợp các điều kiện, do nhiều điều kiện có thể được đánh giá với nhau là một biểu thức đơn. ‘Logical and’ sẽ trả về true duy nhất nếu tất cả các điều kiện thỏa mãn; ‘logical or’ trả về true khi một hoặc nhiều điều kiện thỏa mãn. Toán tử logic cuối cùng, ‘logical not’ trả về true nếu biểu thức theo sau đánh giá là false.

Ví dụ	Tên toán tử	Đánh giá là true khi
$h && $i	And	Cả $h và $i đánh giá là true
$h || $i	Or	Một hoặc cả $h và $i đánh giá là true
$h and $i	And	Cả $h và $i đánh giá là true
$h or $i	Or	$h là đúng hoặc $i là đúng hoặc cả 2 là đúng
$h xor $i	Or loại trừ	Một của $h và $i đánh giá là đúng, nhưng không được cả 2 đúng
! $h	Not	$h không đánh giá là true

Chú ý rằng có hai toán tử “and” và hai toán tử “or”. Chúng hành xử giống nhau nhưng thứ tự khác nhau. Điều này có nghĩa là chúng sẽ được thực thi trong thứ tự khác nhau trong biểu thức chứa nhiều toán tử.

Những ví dụ sau đây nên làm cho tính hữu ích của những toán tử này rõ ràng hơn. Những kết quả đã định sẵn được dựa trên các giá trị sau: `$h == 4; $i == 5; $j == 6:`

```
if ($h == 4 && $i == 5 && $j == 6) echo ("Sẽ được in ra.");
```

Trong trường hợp này, tất cả các điều kiện là `true`, do vậy hàm `echo` sẽ được thực thi.

```
    if ($h == 3 or $i == 5) echo ("Sẽ được in ra.");
```

Đoạn mã trên, điều kiện đầu tiên `($h == 3)` đánh giá là `false` và điều kiện thứ hai `($i == 5)` là `true`.

Bởi vì chỉ một trong số những điều kiện được liên kết bởi `'or'` là `true`, biểu thức đó được đánh giá là `true`.

```
if !($h == 4 && $i == 5) echo ("Sẽ không được in ra.");
```

Ví dụ này là điển hình của toán tử "`not`". Biểu thức `($h == 4 && $i == 5)` đánh giá là `true`, do vậy khi được phủ định với `!`, biểu thức trở thành `false`. Dòng này cũng chỉ ra các thành phần trong ngoặc đơn có thể được sử dụng để liên kết một số điều kiện con để tránh các lỗi thứ tự trước sau. Một ví dụ cuối cùng chỉ ra điều này có thể là hữu ích như thế nào:

```
if (($h == 4 || $i == 4) xor ($h == 5 || $j == 5) xor ($i
== 6 || $j == 7))
   echo ("Sẽ được in ra");
```

2.6 Toán tử ghép chuỗi

Chúng ta đã thấy trong phần II mục 1 dấu chấm (.) được sử dụng trong PHP như thế nào khi toán tử ghép kết nối hai hoặc nhiều giá trị chuỗi vào thành một chuỗi đơn.

```
// Mã lệnh sau in ra "Phineas Phop"
$first = "Phineas";
$last = "Phop";
$full = $first . " " . $last;  // Tên đầu tiên cộng với khoảng trống cộng với tên sau cùng.
echo ($full);
// Mã lệnh sau in ra "Phop's Bicycles"
$last = "Phop";
echo ($last . "'s Bicycles");
```

Kiến thức về toán tử ghép không phải là cách duy nhất để xây dựng các chuỗi sử dụng dữ liệu biến. Khi chúng ta đã tìm hiểu trong Phần II mục 1, PHP nội suy một cách tự động các biến chuỗi trong chuỗi in sai bên trong dấu ngoặc kép đôi. Do vậy, cả hai dòng sau sẽ hiển thị `Phineas Phop`:

```
    echo ($first . " " . $last);  // Sử dụng ghép
    echo ("$first $last");        // Sử dụng nội suy
```

Dòng thứ hai có hiệu suất cao hơn cả về kiểu và thực thi. Tương tự, “Phop’s Bicycles” có thể được in bằng cách sử dụng dòng:

```
    echo ("$last's Bicycles");
```

Trong ví dụ này PHP biết rằng tên của biến là `$last` và không phải là `$last’s` vì dấu nháy `(‘)` không phải là ký tự hợp pháp trong từ định danh (Xem phần ‘Từ định danh’ trong phần trước). Chúng ta không

thể làm những vấn đề giống như thế này nếu chúng ta muốn in ra dòng Phop4bikes:

```
    echo ("$last4bikes"); // Không in ra cái gì!
```

Sẽ không in ra cái gì cả, vì PHP nghĩ chúng ta đang thử in ra giá trị của biến có tên là $last4bikes mà không phải giá trị của $last được theo sau bởi số 4 và chuỗi "bikes". Để sửa điều này, chúng ta có thể sử dụng toán tử ghép để thay thế hoặc cách ly tên của biến sử dụng dấu ngoặc nhọn, do vậy toán tử $ biết các ký tự đó là thành phần của biến:

```
    echo ("${last}4bikes"); // in ra Phop4bikes
```

Toán tử ghép thường được sử dụng để thu thập các chuỗi lớn – chẳng hạn như các truy vấn cơ sở dữ liệu – một phần tách ra một lần:

```
    // Mã lệnh sau tạo ra một truy vấn SQL
    $sql_query = "SELECT Position, Location " .
        "FROM JobOpenings " .
        "WHERE Salary > 60000 " .
        "ORDER BY Location";
```

Hãy cẩn thận khi sử dụng toán tử ghép với các chuỗi số:

```
    echo ("4" . "5"); // In ra 45
    echo (4 . 5 );    // In ra  45  (Với khoảng trống:
            // Giàng buộc giữa các chuỗi "4" và "5".)
    echo (4.5);     // In ra  4.5 (Không khoảng trống:
// PHP hiểu đây là số thực không phải là toán tử ghép!)
```

2.7 Toán tử ba ngôi

Cho đến bây giờ, tất cả các toán tử chúng ta đã thảo luận gồm có các toán tử một ngôi hoặc hai ngôi. Toán tử một ngôi (chẳng hạn như !) thực hiện thao tác của nó chỉ với một giá trị hoặc toán hạng. Toán tử ! đánh giá đối nghịch giá trị `boolean` toán hạng của nó. Nếu `$a` đánh giá là `false` thì `! $a` đánh giá là `true`. Toán tử hai ngôi (chẳng hạn như =) được sử dụng để thực hiện một thao tác trên hai toán hạng: `$a = $b` lấy giá trị của một toán hạng (`$b`) và gán giá trị của nó cho toán hạng khác (`$a`). Chúng ta đã thấy các biểu thức gồm 3 giá trị (chẳng hạn `$a = $b + $c`) nhưng những biểu thức này thường có 2 toán tử và do vậy mô tả 2 toán tử được thực hiện (đầu tiên là phép cộng, sau đó là phép gán, trong ví dụ này).

Có duy nhất một toán tử **ba ngôi**. Một toán tử ba ngôi thực hiện một thao tác đơn trên 3 giá trị khác nhau. Toán tử `? :` thường được ám chỉ tới "toán tử ba ngôi" hoặc "toán tử điều kiện". Nó được sử dụng để kiểm tra điều kiện `Boolean` và trả về một trong hai giá trị. Việc xây dựng gồm có ba phần: Một điều kiện `Boolean` trước dấu hỏi `(?)`, một giá trị giữa `?` và dấu hai chấm `(:)` (nó được trả về nếu điều kiện là `true)` và một giá trị sau dấu hai chấm (nó được trả về nếu điều kiện là `false`).

```
$a == 0 ? "zero" : "not zero"
```

Trong ví dụ này, toán hạng là điều kiện `Boolean $a == 0`. Nếu điều kiện được tìm thấy sẽ là `true`, thao tác trả về chuỗi `"zero".` Ngược lại nó trả về chuỗi `"not zero".` Toán hạng đầu tiên phải luôn

tương ứng với một giá trị Boolean. Hai toán hạng khác có thể là kiểu dữ liệu bất kỳ.

Toán tử ban ngôi về cơ bản là cách viết tắt của câu lệnh if ... else. Câu lệnh:

```
if ($positions > 1) {
   $title = "Available Bicycle Repair Positions";
} else {
   $title = "Available Bicycle Repair Position";
}
```

Có thể được thay thế bởi:

```
$title = "Available Bicycle Repair " . ($positions > 1 ?
"Positions" : "Position");
```

2.8 Các phép toán thao tác mức bit

Các phép toán thao tác mức bit hiếm khi được sử dụng trong PHP. Chúng cho phép so sánh mức thấp và thao tác các số nhị phân. Các toán tử thao tác mức bit được sử dụng để so sánh các giá trị nhị phân một bit một lần. Chúng thực hiện các toán tử tương tự and, or, xor và not trên mỗi tập bit.

Để làm điều này rõ ràng hơn, chúng ta sẽ khảo sát một ví dụ sử dụng toán tử &, nó đánh giá mỗi bit của các toán hạng của nó và thực hiện một Boolean AND trên chúng. Số thập phân 6 được mô tả là 110 trong dãy nhị phân và 5 được mô tả là 101. Nếu chúng ta đánh giá 6 & 5:

```
    echo(
          6   //            110
```

```
          & 5   //             101
        );     // equals 4 = 100
```

Bit quan trọng nhất của 6 và 5 là 1. Do vậy, PHP so sánh 1 & 1, nó tương ứng với biểu thức logic true && true. Vậy biểu thức lôgic này được đánh giá là true, bit kết quả là tập 1. Khi chúng ta khảo sát bit thứ 2 của mỗi toán hạng, chúng ta thấy 1 và 0. 1 & 0 đánh giá là false do vậy bit kết quả là tập 0. Tương tự, bit thứ ba, 0 và 1 kết quả là 0. Do vậy kết quả cuối cùng là 100, nó tương ứng với số thập phân 4. như vậy, 6 & 5 bằng 4. Nếu chúng ta bị nhầm lẫn bởi điều này, tin tức tốt lành là chúng ta hầu như chẳng bao giờ cần nó trong PHP. Các toán tử thao tác bit được tổng kết trong bảng sau:

Toán tử	Mô tả	Ví dụ
&	And	11 (1011 nhị phân) & 13 (1101 nhị phân) 9 (1001 nhị phân)
|	Or	11 (1011 nhị phân) | 13 (1101 nhị phân) 15 (1111 nhị phân)
^	Phủ của or	11 (1011 nhị phân) ^ 13 (1101 nhị phân) 6 (0110 nhị phân)
>>	Dịch bit sang phải bởi	11 (1011 nhị phân) >> 2 2 (10 nhị phân)
<<	Dịch bit sang trái bởi	11 (1011 nhị phân) << 2 44 (101100 nhị phân)
~	Not	~11 (decimal) equals -

		12 (decimal)

$a << $b di chuyển tất cả các bit trong $a sang trái bởi các vị trí $b. Các khoảng trống được tạo phía phải được điền là 0 và các bit đó "rơi xuống" vị trị trái (vợt qua 32 bit) đã không còn nữa. Toán tử >> thực hiện đánh giá dịch trái.

Toán tử phủ định thao tác bit (~) thay đổi mỗi bit của toán hạng của nó để đảo ngược giá trị.

2.9 Các toán tử rút gọn

Giống như các ngôn ngữ lập trình khác, nó cũng có thể thực hiện được trong PHP đó là toán tử rút gọn cho các câu lệnh gán nơi mà toán hạng đầu tiên là một biến và kết quả được lưu trữ trong biến đó. Bảng dưới đây là một danh sách các toán tử rút gọn:

Ví dụ	Tương đương với:
$h += $i	$h = $h + $i
$h -= $i	$h = $h - $i
$h *= $i	$h = $h * $i
$h /= $i	$h = $h / $i
$h %= $i	$h = $h % $i
$h &= $i	$h = $h & $i
$h |= $i	$h = $h | $i
$h ^= $i	$h = $h ^ $i
$h .= $i	$h = $h . $i
$h >>=2	$h = $h >> 2
$h <<= 2	$h = $h << 2
$h ++	$h = $h + 1
$h--	$h = $h – 1

Toán tử tăng dần ++ và toán tử giảm dần -- có thể xuất hiện trước hoặc sau các biến chúng đang xử lý. Vị trí của toán tử quyết định thứ tự của các sự kiện xảy ra. Nếu ++ được đặt trước biến, PHP tăng giá trị đầu tiên và sau đó trả về giá trị mới đã được tăng. Nếu đặt sau biến, PHP trả về giá trị của biến trước khi tăng đầu tiên và sau đó thực hiện tăng biến. Ví dụ:

```
/* Khi sử dụng ++, kết quả trả về có giống nhau hay
không phụ thuộc vào nó đứng trước hay sau biến: */
$a = 10;   // $a là 10
$a++;      // $a là11

$a = 10;   // $a là 10
++$a;      // $a là 11

// Nhưng:
$a = 10;   // $a là 10
$b = $a++; // $a là 11, nhưng $b là 10!
           // Gán xảy ra trước khi tăng

$a = 10;   // $a là 10
$b = ++$a; // $a là 11, và $b là 11
           // Gán xảy ra sau khi tăng
```

2.10 Các thao tác ưu tiên và các thao tác kết hợp

Thứ tự ưu tiên ám chỉ tới thứ tự trong các toán tử khác nhau sẽ được thực thi, xem xét biểu thức `9 - 4 * 2`. Chúng ta sẽ nghĩ rằng kết quả của biểu thức này là `10` (vì `9 - 4` là `5` và `5 * 2` là `10`). Tuy nhiên

toán tử * được ưu tiên trước toán tử -, do vậy phép nhân được thực hiện trước phép trừ. Do đó `9 – 4 * 2` thực tế được đánh giá là `1: 4 * 2` được tính đầu tiên và sau đó là kết quả của phép trừ từ 9. Nếu chúng ta muốn phép trừ xảy ra trước phép nhân, chúng ta có thể chỉ định ưu tiên rõ rang với dấu ngoặc đơn: `5 * (4 - 2).` Sử dụng các dấu ngoặc đơn luôn là một cách sang suốt để tránh nhầm lẫn và nhập nhằng.

Kết hợp ám chỉ tới thứ tự trong các toán tử được ưu tiên giống nhau sẽ xảy ra. Ví dụ, phép chia có sự kết hợp từ trái sang phải (Thường được tính toán bắt đầu từ bên trái), do vậy `8 / 4 / 2` được đánh giá thành `(8 / 4) / 2` hoặc `1` (và không đánh giá thành `8 / (4 / 2),` nó sẽ bằng 4). Giả sử chúng ta muốn nó thực thi từ phải sang trái, nếu chúng ta viết:

```
$h = 8;
$i = 4;
$j = 2;
$h /= $i /= $j;
echo($h);
```

Điều này tương đương với:

```
$h = 8;
$i = 4;
$j = 2;
$i = $i / $j;
$h = $h / $i;
```

```
echo($h);
```

Do vậy nó sẽ in ra 4 mà không phải 1. Danh sách các toán tử của bảng sau chỉ ra những toán tử sẽ được ưu tiên đầu tiên.

Toán tử	Thao tác	Kết hợp	Số toán hạng
()	Thiết lập quyền ưu tiên	-	Một ngôi
New	Khai báo đối tượng	-	Một ngôi
[]	Truy cập chỉ mục mảng	Phải	Hai ngôi
!	Not logic	Phải	Một ngôi
~	Not thao tác bit	Phải	Một ngôi
++ --	Tăng, giảm	Phải	Một ngôi
@	Hàm chặn lỗi	Phải	Một ngôi
* / %	Nhân, chia, lấy dư	Trái	Hai ngôi
+ - .	Cộng, trừ, ghép	Trái	Hai ngôi
<< >>	Dịch bit sang trái, dịch bit sang phải	Trái	Hai ngôi
< <= > >=	Nhỏ hơn, nhỏ hơn hoặc bằng, lớn hơn, lớn hơn hoặc bằng	-	Hai ngôi
== !=	Bằng, không bằng	-	Hai ngôi
&	Thao tác bit AND	Trái	Hai ngôi
^	XOR thao tác bit	Trái	Hai ngôi
|	OR thao tác bit	Trái	Hai ngôi
&&	AND logic	Trái	Hai ngôi

`||`	OR logic	Trái	Hai ngôi
`? :`	Điều kiện	Phải	Ba ngôi
`= += -= *= /= .= %= &= != ~= <<= >>=`	Gán	Phải	Hai ngôi
`And`	AND logic	Trái	Hai ngôi
`Xor`	XOR logic	Trái	Hai ngôi
`Or`	OR logic	Trái	Hai ngôi
`,`	Đánh giá nhiều giá trị	Trái	Hai ngôi

3. Tổng kết

Trong chương này chúng ta đã tìm hiểu thêm về biến PHP và kiểu dữ liệu của chúng ta. Các hằng và các biến mô tả các giá trị trừu tượng. Các hằng được định nghĩa bằng cách sử dụng hàm `define()`. Được định nghĩa một lần, một giá trị hằng không thể thay đổi. PHP có một số hằng được xây dựng sẵn bao gồm thông tin về kịch bản PHP và môi trường của nó.

Năm kiểu dữ liệu trong PHP là `integer, double, string, array`, và `object`. Các giá trị `Boolean` được mô tả bởi các số nguyên 0 và khác 0, mặc dù thỉnh thoảng chúng cũng được mô tả bởi chuỗi rỗng hoặc khác rỗng. Kiểu của một biến quyết đinh ngữ cảnh mà nó sử dụng. PHP cố gắng chuyển đổi hoặc lừa kiểu là cần thiết. Để ra lệnh rõ ràng một kiểu xác định, chúng ta có thể sử dụng ép kiểu, `settype()` hoặc hàm `...val()`. Hàm `is...()` có thể được sử dụng để quyết định kiểu hiện thời của biến.

Câu hỏi trắc nghiệm kết chương

1. Tất cả các biến trong PHP bắt đầu với ký tự nào?
   a. &
   b. !
   c. $
2. Trong PHP chúng ta có thể sử dụng cả dấu nháy đơn (‘’) và dấu nháy kép (“ ”) cho các chuỗi?
   a. Đúng
   b. Sai
3. Cách chính xác để tăng biến $count lên 1 đơn vị?
   a. $count =+1
   b. count++;
   c. ++count
   d. $count++;
4. Trong các cách khai báo biến sau, cách khai báo nào không hợp lệ?
   a. $myVar
   b. $my-Var
   c. $my_Var
5. Kết quả sau khi thực hiện câu lệnh echo(6 & 5); ?
   a. 4
   b. 5
   c. 6
   d. 1