

Cài đặt Web server - bước đầu tiên để học PHP!

"Trường học" vừa khai giảng, trong lúc đợi bài học đầu tiên ra lò, cho phép tớ được "múa riu qua mắt thợ cái"! 🇺🇸

Như bác ngocha85 đã nói, để học PHP và MySQL, một trong những thứ cần chuẩn bị là web server chạy trên PC của mình. Để cho nhanh chóng, theo tớ tốt nhất nên cài bộ XAMPP.

Câu hỏi 1: XAMPP là gì?

Trả lời: XAMPP giống với WAMP, nghĩa là người mới học ko cần phải biết cách cài đặt riêng lẻ từng thành phần như Apache, PHP và MySQL. Chỉ cần download một gói về là xong.

Các tính năng có trong XAMPP:

1. Apache 2 => server
2. PHP 5 => ngôn ngữ lập trình
3. MySQL => cơ sở dữ liệu
4. Webalizer => quản lý statistic của site
5. Mercury => giả lập gửi email
6. FileZilla => giả lập FTP server
7. Rất nhiều tính năng chuyên sâu khác...

Câu hỏi 2: Tại sao ko dùng WAMP?

Trả lời: Vì cái này tớ chưa dùng bao giờ 🇺🇸 + Cái này bác ngocha85 chắc sẽ giới thiệu => tránh đụng hàng là hơn. Với lại cái XAMPP này theo tớ cũng rất hay, thậm chí ko cần cài đặt, chỉ cần copy và chạy.

Câu hỏi 3: Down XAMPP ở đâu?

Trả lời: Ở đây: <http://www.apachefriends.org/en/xampp.html>

Có đủ bộ XAMPP cho Windows, Linux, MacOS và cả Solaris, vì vậy mọi hệ điều hành nó đều chấp tất! 🇺🇸

Câu hỏi 4: Cài đặt và sử dụng XAMPP như thế nào?

Trả lời: Sau khi tải về, bạn sẽ có 1 file zip. Giải nén file đó ra 1 thư mục bất kỳ, ví dụ C:\XAMPP. Để chạy web server, bạn kích hoạt file xampp_control.exe, bấm nút Start bên cạnh Apache và nút close để XAMPP Control tự động chuyển xuống system tray.

Ngoài ra, bạn có thể khởi động MySQL nếu dùng cơ sở dữ liệu, FileZilla nếu dùng FTP và Mercury nếu dùng email.

Để biết chắc web server đã chạy đúng, bạn mở trình duyệt web của mình, gõ <http://localhost> vào thanh Address, sau đó enter. Một trang thông báo sẽ hiện ra, cho biết quá trình "cài đặt" đã hoàn tất.

Câu hỏi 5: Làm thế nào để chạy các script viết bằng PHP?

Trả lời: Bạn cho script vào thư mục C:\XAMPP\htdocs\ sau đó gọi file đó qua URL http://localhost/script_name.php

Vậy là hết "bài chuẩn bị cho bài khởi động" của bác ngocha85 sẽ post trong vài ngày tới. Tèn tén ten! 🇺🇸

Thừa thắng xông lên, tớ làm luôn bài "Hello World". 🇺🇸

Bài 1: Nói "hello world" với PHP

Cần chuẩn bị những gì?

1. Web server cần đảm bảo sẵn sàng. Apache được khởi động theo bài post ở trên.
2. Một script editor. Cái này có rất nhiều, như PHP Designer, Dev-PHP, ... Thậm chí dùng notepad cũng được. Nhưng tốt nhất nên dùng một editor có hỗ trợ unicode. Như tớ dùng SCiTE.

Bạn vào trang này để xem list và review các PHP editor: <http://www.php-editors.com/>

3. 5 phút thời gian rảnh rỗi.

Bắt đầu!!!

1. Tạo một file mang tên "helloworld.php" trong thư mục htdocs. Mở file đó bằng script editor.
2. Gõ đoạn code sau vào editor:

PHP Code:

```
<?php
    echo "Hello World!";
?>
```

3. Mở trình duyệt web, gõ <http://localhost/helloworld.php> [enter].
4. Nhắm mắt lại trong 0.0001 giây. Nếu mở mắt ra mà bạn thấy dòng chữ Hello World là đã thành công rồi đó! 🇺🇸

Giải thích

1. Dòng thứ nhất của file helloworld.php là "<?php" và dòng cuối cùng là "?>". Đây là 2 thẻ (tag) để báo cho server biết điểm bắt đầu và kết thúc của một đoạn code PHP. Nói cách khác, bằng cách này bạn có thể nhúng code PHP trong bất cứ file HTML có sẵn nào. Khi thực thi file PHP, web server sẽ chỉ thực hiện những đoạn code đặt trong 2 thẻ này và bỏ qua tất tần tật những phần còn lại.
2. Dòng 2 là một lệnh của PHP: Lệnh echo. Lệnh này làm nhiệm vụ in một xâu ra ngoài màn hình. Cần nhớ một lệnh PHP luôn kết thúc bằng dấu chấm phẩy ";". Nếu thiếu dù chỉ một dấu chấm phẩy, code của bạn sẽ ko chạy và dừng lại biểu tình ngay. 🇺🇸
3. Cũng ở dòng 2, xâu "Hello World" được đặt trong dấu ngoặc kép. Nếu ko, sẽ có lỗi.

Một vài câu hỏi

1. Có cần thiết phải trình bày như trên ko?
=> Ko. Bạn có thể trình bày code theo bất cứ cách nào bạn muốn. Lùi vào 10 dấu cách, mỗi dòng cách nhau 3 hàng, ... Điều đó là tùy bạn. Tuy nhiên cần phải viết code cho thật dễ đọc và dễ hiểu để tiện cho việc sửa đổi và chia sẻ code sau này.
2. Có cách nào báo hiệu một đoạn code PHP ngoài cách dùng <?php ko?
=> Có. Nhiều cách là đằng khác. Ví dụ bạn có thể viết

PHP Code:

```
<?  
    // Code ở đây  
?>
```

Tuy nhiên các cách khác đều ít thông dụng và được khuyến cáo ko nên sử dụng.

3. Có thể đặt xâu Hello World trong dấu ngoặc đơn ko?

=> Có thể. Bạn có thể dùng dấu ngoặc kép và dấu nháy đơn để chứa xâu. Sự khác nhau giữa chúng sẽ được thảo luận sau.

4. Nếu trong xâu cũng có dấu ngoặc / xâu là một đoạn văn bản rất dài thì sao?

=> Ko có gì phải lo lắng. Cái gì cũng có cách giải quyết. Vấn đề là cách đó ko nằm trong bài học hôm nay. Hết 5 phút rồi, bạn hãy nghỉ ngơi đã. :P

Bài tập

Vì Bài 1 hết sức đơn giản, chỉ theo tinh thần Hello World nên bài tập cũng sẽ chỉ có bài, và cũng rất rất đơn giản.

Hãy cho biết lỗi sai trong các đoạn code sau:

1.

PHP Code:

```
echo "Hello World!";
```

2.

PHP Code:

```
<?php  
    echo "Hello World!"  
?>
```

3.

PHP Code:

```
<?php  
    echo "Hello World!";  
?>
```

4.

PHP Code:

```
<?php
    echo "Hello World!";
?>
```

Bài 2 - Mục 1: Lưu trữ dữ liệu trong PHP. Vài điều cần nói về biến.

Trước khi bắt đầu bài 2, tớ xin trình bày về cách chú thích (comment) trong PHP. Đây có thể coi là một kỹ năng cũng được, vì bạn rất **KHÔNG NÊN** viết code mà ko có chú thích. Có thể đoạn code rất dễ hiểu vào thời điểm viết, nhưng nếu ko có chú thích, chỉ vài tháng sau bạn có thể quên ngay mình đã viết cái gì. Viết chú thích ngay vào thời điểm code là cách tốt nhất.

Trong PHP, một dòng chú thích được đặt sau 2 dấu sổ chéo //

Ví dụ

PHP Code:

```
// Đây là một dòng chú thích
```

Nếu chú thích của bạn dài hơn 1 dòng, bạn có thể để nó trong 1 block, mở đầu bằng /* và kết thúc bằng */

PHP Code:

```
/*
    Chú thích dòng thứ nhất
    Thứ 2
    Thứ 3
    Vân vân...
*/
```

Còn một cách nữa, **KHÔNG** phổ biến (ít ra là tớ thấy thế), đó là chú thích đặt sau dấu #. Chú thích này cũng chỉ cho phép 1 dòng giống như //

Một điều khác cũng rất cần chú ý đó là PHP ko cho phép đặt chú thích trong chú thích (nested comment).

Ta bắt đầu vào Bài 2.

Bài 2: Lưu trữ dữ liệu trong PHP

Khi bắt tay vào lập trình một chương trình, hiển nhiên ta sẽ cần phải lưu trữ dữ liệu. Cụ thể, dữ liệu có thể được lưu trữ bằng biến (variable). Khác với các ngôn ngữ lập trình khác, trong PHP các biến ko cần phải khai báo (declare) trước khi sử dụng. Để sử dụng biến, bạn chỉ cần gán (assign) cho nó một giá trị (value). Biến sẽ tự động được tạo. Cực kỳ đơn giản và nhanh chóng!

1. Biến. Khai báo. Đặt tên.

Biến trong PHP bắt đầu bằng dấu dollar (\$), theo sau là tên biến. Tên biến có thể bắt đầu bằng dấu gạch dưới (_ gọi là underscore) hoặc chữ cái. Tiếp sau đó là các chữ cái, số hoặc lại là dấu gạch dưới. Một số ký tự mở rộng (extended character) có thể được sử dụng, nhưng tốt nhất là nên tránh.

Một số ví dụ về biến ĐÚNG: \$suds, \$update_softs, \$suds_has_more_than_26000_members

Biến sai: abc vì thiếu dấu dollar, \$124adfd vì bắt đầu bằng số

Cũng cần thảo luận thêm một chút: Cũng vì sự dễ dãi trong việc ko phải khai báo biến nên sẽ có lúc bạn gõ nhầm tên biến. Ví dụ \$suds gõ thành \$usd (Ặc!)

Ví dụ:

PHP Code:

```
<?php
    $suds = "Welcome to UDS!";
    echo $suds;
?>
```

May mắn làm sao, từ bản PHP 5 trở lên, sẽ có một cảnh báo (warning) khi bạn chạy script, cho biết bạn chưa gán giá trị cho biến \$usd.

À, còn một vấn đề chưa nói đến: Đó là trong PHP, tên biến CÓ phân biệt chữ hoa chữ thường (case-sensitive). Nghĩa là \$suds hoàn toàn khác với \$UDS hay \$uDs. Nói chung nên tránh việc đặt tên biến chỉ khác nhau cách viết hoa thường này, vừa đỡ mất công giữ Shift, vừa đỡ nhớ nhầm tên biến.

Mục 2 sẽ mang tên **Một số kiểu dữ liệu trong PHP**. Mọi người đón đọc nhé! 📖

Bài tập

Trong các biến sau đây, biến nào được đặt tên đúng, biến nào bị đặt tên sai: 😊

1. this_is_a_variable
2. \$yet another variable
3. \$simplevariable
4. \$blah_blah_blah_123456789_____
5. \$123456789_____abacabadfskdjsfksdfkdserwuewrjfdksj fdksljf
6. \$^
7. \$__A__VARIABLE__
8. \$THiS_iS_ThE_LaST_ONe

Bài 2 - Mục 2: Lưu trữ dữ liệu trong PHP. Một số kiểu dữ liệu cơ bản [updated]

Ngoài lề một chút: Lúc đầu tớ cũng ko định tách Bài 2 ra làm mấy thread, nhưng nếu để như thế kia thì dài quá, sợ đọc theo các bác mệt mắt => nản lòng. 🤔

Ta có một số kiểu dữ liệu cơ bản sau đây trong PHP:

- Kiểu số (number)
- Kiểu xâu (string)
- Kiểu boolean (boolean)

a. Kiểu số

Trong kiểu số (lại) có 2 kiểu cơ bản khác: Số nguyên (int) và số thực (float). Số nguyên có thể biểu diễn bằng số thập phân (hệ 10 - decimal), hệ 8 (octal) và hệ 16 (hexadecimal).

Ví dụ ta gán giá trị cho một số biến kiểu NGUYÊN như sau:

PHP Code:

```
<?php
$a = 27;
$b = -27;
$c = 027;
$d = -027;
$e = 0x27;
$f = -0x27;
?>
```

Ở ví dụ trên, cả 6 biến từ \$a đến \$f đều có giá trị là 27 hoặc -27. Tuy nhiên, với biến \$a và \$b, ta dùng kiểu biểu diễn số thập phân (viết như số ta viết hàng ngày). Với \$c và \$d, dùng kiểu số hệ 8 (bắt đầu với chữ số 0). Với \$e và \$f dùng kiểu hệ 16 (bắt đầu với chữ số 0 và chữ cái x).

Nếu đã từng học qua Pascal, chắc chắn bạn sẽ hỏi tớ: Thế nếu tớ dùng 1 biến kiểu int, gán cho nó một giá trị cao bằng max của int, thì khi đem số đó cộng với 1, giá trị có bị chuyển thành âm do tràn số (overflow) ko?

Câu trả lời là ko. Một biến kiểu int có giá trị cực lớn trong PHP là 2147483647, khi cộng 1 vẫn sẽ trả giá trị đúng là 2147483648, nhưng lần này sẽ thuộc kiểu float. Nói cách khác, PHP tự chuyển số bị tràn lên kiểu float.

Nếu thích đặt câu hỏi, chắc chắn (lại một lần nữa) bạn sẽ hỏi tớ: Sao cậu biết điều ấy?

Câu trả lời rất đơn giản: Bạn hãy cùng tớ làm ví dụ với đoạn code sau:

PHP Code:

```
<?php
$a = 2147483647;
var_dump($a);
$a = $a + 1;
```

```
var_dump($a);  
?>
```

Sau khi chạy script, kết quả trả về sẽ là

```
int(2147483647) float(2147483648)
```

=> Đúng như tớ nói nhá! 🤖

Tớ xin giải thích như thế này:

Ở dòng thứ nhất, ta đem gán giá trị 2147483647 cho \$a. Đây là một giá trị cực to, nhưng vẫn nằm trong int, vì vậy \$a sẽ thuộc kiểu int.

Dòng thứ 2 và thứ 4, ta dùng lệnh var_dump(\$a); Đây là lệnh in ra kiểu và giá trị của một biến trong PHP. Chú ý nhé, lệnh này khá phổ biến và hay được dùng để debug code.

Ở dòng thứ 3, ta dùng lệnh gán \$a = \$a + 1; Với các bạn đã học lập trình, điều này chẳng có gì khó hiểu. Sau khi thực thi lệnh, \$a sẽ mang giá trị của \$a cộng thêm với 1. Còn nếu (chẳng may) bạn chưa học lập trình bao giờ, thì tớ (lại) xin giải thích như thế này:

- Dấu bằng ở đây là lệnh gán, đem giá trị của vế phải gán cho vế trái, chứ ko phải là dấu bằng trong biểu thức toán học mà mình vẫn học. Do đó, ko có gì là trái với lẽ tự nhiên cả. :P

Một điều khác mà bạn nên nhớ, đó là hãy THẬT cẩn thận khi sử dụng số kiểu float trong PHP. Nó luôn chỉ là những giá trị xấp xỉ, và ko hề chính xác tuyệt đối. Do đó tốt nhất là chuyển số float sang int khi có thể. Cách làm sẽ được thảo luận sau.

Giờ ta sang kiểu xâu.

b. Kiểu xâu

Định nghĩa một cái nào: Xâu là một chuỗi các ký tự. Một câu tớ xì pam là một xâu. Cả cái bài viết này cũng có thể là một xâu.

Để sử dụng xâu, có 3 cách (hic, bắt đầu phức tạp rồi => mọi người đừng dậy vượn vai cái cho tỉnh táo! :P):

Cách 1 là dùng nháy đơn.

Cách 2 là dùng ngoặc kép (hay gọi là nháy kép gì cũng được).

Cách 3 là dùng kiểu HEREDOC.

Nói rõ nhé:

Cách 1: Xâu được đặt trong dấu nháy đơn.

PHP Code:

```
<?php  
echo 'Đây là xâu đặt trong dấu nháy đơn';  
?>
```

Sẽ có bạn hỏi tớ (sao hỏi nhiều thế!): Trong xâu có thể đặt dấu nháy đơn được ko? Kiểu như

xâu là 'I'm a student ý.

Câu trả lời là bạn phải thêm một dấu sọc (hay sọc gì ý) trước dấu nháy đơn "bất thường" ý. Như thế này:

PHP Code:

```
echo 'Trong nháy đơn lại có một nháy đơn như thế này \', và như thế này nữa \.';
```

Đặt cái dấu đó (\) gọi là "escape the character". Nói nhỏ nhá: Bài tớ viết hay chèn tiếng Anh vào là để các bạn đỡ "bỡ ngỡ" khi đọc tut hay doc bằng Eng.

Một lần nữa, (lại) có một câu hỏi được đặt ra: Nếu trong xâu cũng có một dấu \ thì sao? Câu trả lời cũng rất giản dị: Dùng thêm một dấu \ nữa ngay trước dấu \ ý. Như thế này \

Lần này, sẽ ko có một câu hỏi, mà sẽ là một tiếng thở dài: Sao lăm thứ thế? Còn cái dấu nào phải "escape" như dấu \ và ' ko?

Có. Đó là:

1. \n : Báo hiệu xuống dòng trong PHP. Giống như
 trong HTML.
2. \t : Thay mặt cho Tab
3. \\$: Dấu dollar (tránh "cạnh tranh lành mạnh" với tên biến mà! :P)
4. ... Để gặp nói sau. Nói nhiều e "tẩu hỏa nhập ma" chết!

Quên mất, trừ \ và \, mấy cái escape này chỉ dùng trong trường hợp xâu đặt trong dấu ngoặc kép.

Hờ hờ, lại quên một điều phải nói trước khi chuyển qua phần kế tiếp: Nếu trong xâu ta ko thêm dấu \, cũng kóc thêm dấu \, mà dùng cả \ cho "dân chơi" thì sao? 🤖

Trả lời: Thì cứ làm như bình thường thôi. Như thế nè: '\\'. Dấu \ thứ 1 để escape cho dấu \ thứ 2. Dấu \ thứ 3 để escape cho dấu ' cuối cùng. Thường thôi!

Cách 2: Xâu được đặt trong dấu ngoặc kép (hay nháy kép - whatever)

Trường hợp này rất giống với sử dụng dấu nháy đơn đã nói ở trên.

PHP Code:

```
<?php
    echo "Xâu này đặt trong dấu ngoặc kép";
?>
```

Sở dĩ nói RẤT giống mà ko phải HOÀN TOÀN giống vì giữa chúng có điểm khác nhau: Khi thực thi, PHP sẽ tìm và thay thế trong xâu những ký tự đặc biệt được escape (như \n, \t...) như đã nói ở trên, cùng với các biến (nếu có) trong xâu.

Ví dụ:

PHP Code:

```
<?php
    $a = 1;
```

```
echo "Biến \$a có giá trị là $a";  
?>
```

Sẽ cho ta kết quả: Biến \$a có giá trị là 1

Trong khi đó, nếu sử dụng dấu nháy đơn:

PHP Code:

```
<?php  
$a = 1;  
echo 'Biến \$a có giá trị là $a';  
?>
```

Lại in ra: Biến \\$a có giá trị là \$a

Điều đó cho thấy: Khi sử dụng dấu nháy đơn, giá trị của biến trong chuỗi, cùng với các ký tự đặc biệt cần escape sẽ không được in ra. Các bạn nhớ kỹ điều này nhé!

Ta sang cách thứ 3: Chuỗi đặt trong cấu trúc HEREDOC

Ở cách 1, PHP sẽ nhận thấy 1 chuỗi được bắt đầu với dấu nháy đơn thứ nhất và kết thúc với dấu nháy đơn thứ 2. Tương tự với cách 2, nhưng là dấu ngoặc kép.

Ở cách 3 này, PHP sẽ coi một chuỗi bắt đầu bằng 3 dấu nhỏ hơn viết liền nhau <<<, đi kèm với 1 tên định danh (identifier) tùy ý bạn đặt tên, ví dụ là HERE, kết thúc là tên đó kèm theo dấu ;

Nghe có vẻ hơi phức tạp, nhưng bạn hãy cùng thử ví dụ sau: (chú ý là chữ HERE có thể thay bằng bất cứ chữ gì, tên bạn chẳng hạn, miễn là nó tuân theo nguyên tắc đặt tên biến của PHP. À, mà nhớ là mở bằng <<<HERE thì phải đóng bằng HERE; nhé, không được mở cửa ra vào, đóng cửa sổ đâu!)

PHP Code:

```
<?php  
echo <<<HERE  
    Xâu &#273;ược ghi ở dòng thứ nhất  
    Dòng th&#7913; 2  
    Dòng th&#7913; 3  
    Văn bản  
HERE;  
?>
```

Nhìn vào ví dụ trên, bạn có nhận xét gì?

Thứ nhất, xâu ko nhất thiết phải thuộc một dòng. Nó ko nhất thiết phải ngắn gọn, mà có thể dài "tràng giang đại hải" ra mấy chục dòng cũng được. Điều này rất tiện nếu bạn muốn echo một lúc cả một bài thơ chẳng hạn! 🤖

Thứ hai, chữ HERE; ở dòng cuối cùng tở ko cần lè với chữ echo ở dòng 1. Đó là LUẬT, dù tở thấy hơi "bất công" và "nghiệt ngã" một tí:

- Sau <<<HERE phải xuống dòng. Ko được phép có dù chỉ 1 ký tự trắng (dấu cách ý)
- Trước và sau HERE; cũng thế. Ko được phép có dù chỉ 1 ký tự trắng. Nói cách khác, đừng đại gì cần lè cho dòng này. 😊

Cái gì là LUẬT thì phải THEO, cãi ko được 🙄 Còn nếu ko theo, PHP sẽ báo lỗi:

Quote:

```
Parse error: parse error, unexpected T_SL in E:\XAMPP\htdocs\test.php on line 2
```

Tở nói dòng dài như vậy là vì đã từng mất bao nhiêu thời gian mới tìm ra được lỗi sai của mình. Chỉ vì một dấu cách mà chương trình đình công, ko thèm chạy! Kinh nghiệm xương máu!

Còn một ý nữa: Nếu từ nãy đến giờ bạn chỉ đọc "chay", ko thực hành thì (chưa chắc) đã nhận thấy: Khi chạy chương trình, thay vì in ra mấy dòng như trên, PHP lại in mọi thứ ra cùng 1 dòng:

Quote:

```
Xâu được ghi ở dòng thứ nhất Dòng thứ 2 Dòng thứ 3 Vân vân
```

Sửa chữa điều này cũng khá đơn giản:

PHP Code:

```
<?php
    $s = <<<HERE
        Xâu &#273;ược ghi ở dòng thứ nhất
    Dòng th&#7913; 2
        Dòng th&#7913; 3
    Vân vân
HERE;
    echo nl2br($s);
?>
```

Có gì bí ẩn ở đây ko? Thay vì echo thẳng mấy dòng kia ra, ta đi "vòng vèo" một chút bằng cách gán xâu chứa mấy dòng đó cho biến \$s, sau đó echo nl2br(\$s) ra màn hình.

nl2br() được gọi là một hàm (function). Nó nhận một xâu làm tham số (parameter), ở đây là xâu \$s, sau đó in ra theo luật: Cứ gặp dấu xuống dòng trong code là chuyển thành dấu xuống dòng trong HTML.

Cái tên nl2br cũng chẳng phải thần chú gì khó nhớ, nó rất giản dị: chỉ là viết tắt của new-line-to-br. New-line là dấu xuống dòng trong code, 2 là to 🇺🇸, br là
 (thẻ xuống dòng trong HTML).

Vậy là vấn đề đã được giải quyết. Kết quả in ra đúng như mong đợi:

Quote:

```
Xâu được ghi ở dòng thứ nhất  
Dòng thứ 2  
Dòng thứ 3  
Vân vân
```

Kiểu dữ liệu cơ bản cuối cùng mà tớ sẽ nói tới chính là Kiểu boolean.

c. Kiểu boolean

Đây là kiểu dữ liệu đơn giản nhất trong PHP (đỡ quá!). Ý tưởng rất đơn giản: Mọi thứ chỉ thuộc vào 1 trong 2 loại: Đúng hoặc Sai, Có và Không, 1 và 0. Không có ngoại lệ. Anh ko là True thì sẽ là False. Ở đây ko có chỗ cho người ba phải!!!

Giá trị của biến kiểu boolean là TRUE hoặc FALSE. Hai từ này hoàn toàn ko phân biệt hoa thường, vì vậy có thể viết như thế nào cũng được: TRue, tRUe, true, ...

Ví dụ:

PHP Code:

```
<?php  
    $a = TRUE;  
    $b = false;  
?>
```

Một kiểu dữ liệu đơn giản đồng nghĩa với việc ko cần giải thích nhiều về ví dụ của nó. 🇺🇸

Nhưng nó lại ko đồng nghĩa với việc: Kiểu boolean chẳng có gì đáng nói! Thực tế là kiểu này rất hay dùng trong PHP, ví dụ khi tính toán một biểu thức và xem giá trị của nó có lớn hơn một số nào đấy hay ko... (biểu thức điều kiện)

Xin được kết thúc Bài 2 tại đây. Cảm ơn quý vị đã quan tâm theo dõi...

COMING UP NEXT: **Một số hàm cần thiết khi debug code**

Bài 2 - Mục 3: Các kiểu dữ liệu quan trọng khác

Tiếp sau mục 2: Các kiểu dữ liệu cơ bản, tớ xin giới thiệu thêm một vài kiểu dữ liệu quan trọng khác của PHP: Mảng, Đối tượng, Null và Resource (sozy vì 2 kiểu cuối ko rõ dịch như thế nào)

1. Mảng (array)

Mảng được sử dụng khi bạn muốn lưu trữ một số lượng lớn các biến. Một ví dụ hết sức đơn giản: Một lớp có 50 học sinh, và bạn muốn quản lý cả 50 học sinh đó. Để đại diện cho một học sinh, tất nhiên bạn sẽ muốn 1 biến. Nhưng nếu đặt tên là hs1, hs2, ... hs50 thì quả là quá mất thời gian! Và đây chính là lý do để mảng có "đất dụng võ".

Mảng chứa rất nhiều giá trị (value), mỗi giá trị được truy cập nhờ khóa (key). Khóa có thể chỉ là những số đếm thông thường như 1, 2, 3, hay có thể là chuỗi, như "abc", "def", "ghi". Mảng có khóa là chuỗi như vậy được gọi là associative array.

Để khai báo một mảng, chúng ta có thể sử dụng cách như ví dụ sau:

PHP Code:

```
<?php
$a = array(1, 2, 3, 4);
$b = array("a", "b", "c");
$c = array(1, "a", array(3, 4));
?>
```

Như ở ví dụ trên, \$a, \$b, \$c đều là mảng. Mảng \$a chứa các số từ 1 đến 4, mảng \$b chứa các chuỗi "a", "b", "c". Còn mảng \$c sành điệu hơn, chứa cả số lẫn chuỗi, thêm cả một mảng ở bên trong nó nữa. 🍷

Sau khi khởi tạo giá trị trong mảng \$a, mặc định mỗi phần tử (element) trong nó sẽ được gán cho một khóa là số nguyên. Nó bắt đầu từ 0, ko phải là 1. Do đó, phần tử thứ 0 sẽ là 1, thứ 1 sẽ là 2, vân vân.

Ví dụ:

PHP Code:

```
<?php
echo $a[2];
?>
```

Sẽ in ra màn hình giá trị 3 - tức là phần tử mang khóa là 2 trong mảng \$a.

Như tớ đã nói ở trên, một khóa có thể là một chuỗi, nghĩa là người ta có thể truy cập mảng \$d (chẳng hạn) bằng cách dùng \$d["blah"]. Vậy ta khởi tạo giá trị của \$d như thế nào?

Rất đơn giản, ta sử dụng toán tử (operator) =>

PHP Code:

```
<?php
$d = array("blah" => 1, "abc" => 2, "def" => "ghi");
?>
```

Có thể dễ dàng đoán được: Nếu dùng lệnh `echo $d["def"]` sẽ cho ra kết quả là "ghi".

Tìm hiểu sâu thêm về Mảng, kiểu dữ liệu mạnh mẽ của PHP, sẽ là phần việc của một Bài học sau này.

Đối tượng (object)

PHP5 là một ngôn ngữ lập trình hướng đối tượng (OO - Object Oriented). Nói một cách đơn giản nhất (nhưng vẫn nghe ù tai nếu bạn chưa nghe về đối tượng bao giờ) thì lập trình hướng đối tượng (OOP - Object Oriented Programming) là việc tạo ra một kiểu dữ liệu mới (đối tượng - object hay lớp - class). Thay vì việc phải tạo một dãy các hàm liên quan đến đối tượng đó, bạn sử dụng thuộc tính (properties) và phương thức (method) trực tiếp của đối tượng ý.

Hãy nhắm mắt vào tưởng tượng. Bạn có một quả bóng bay. Quả bóng ý có những thuộc tính gì? À, rất đơn giản thôi: Đó có thể là kích thước, màu sắc hay độ căng - xẹp của bóng.

Còn phương thức: Quả bóng có thể căng lên, hoặc xẹp đi. Rất dễ dàng phải ko?

Giờ hãy tưởng tượng, bạn có một đối tượng mang tên QB (quả bóng 🎈). Để tạo ra một quả bóng, bạn dùng lệnh:

PHP Code:

```
<?php
    $bong = new QB();
?>
```

Quả bóng có kích thước (KT), màu sắc (MS) và độ căng - xẹp (CX). Để \$bong mang màu đỏ, bạn có thể viết:

PHP Code:

```
<?php
    $bong->MS = red;
?>
```

Tương tự, nói đến kích thước, độ căng - xẹp của quả bóng, ta có thể dùng `$bong->KT`, `$bong->CX`.

Thế còn phương thức? Như đã nói, quả bóng có thể căng lên (CL) hoặc xẹp đi (XD). Để thực thi các phương thức này, ta làm như ví dụ sau:

PHP Code:

```
<?php
    $bong->CL();
?>
```

Tạm dừng việc "cưỡi ngựa xem hoa" phần đối tượng tại đây.

3. Null

Một biến được coi là NULL (không có giá trị) nếu nó thỏa mãn cả 3 điều kiện sau:

1. Nó được gán là NULL (không phân biệt hoa thường)
2. Nó chưa bao giờ "được" (hay "bị") gán giá trị.
3. Nó đã bị "xử đẹp" bằng unset - hàm hủy bỏ các biến chỉ định.

Để kiểm tra một biến có là NULL hay không, ta có thể sử dụng hàm `is_null(biến)`. Ví dụ:

PHP Code:

```
<?php
    $test = NULL;
    echo is_null($test);
?>
```

Cho ra kết quả là 1.

4. Resource

Có những lúc PHP cần xử lý các đối tượng như kết nối cơ sở dữ liệu hay các đối tượng của hệ điều hành. Chúng sẽ được coi là resource.

Nói chung trong hầu hết các trường hợp, bạn thậm chí không nhận ra việc mình có phải đang làm việc với resource hay không.

Bài 3: Kết hợp PHP và HTML

Nói đến PHP, người ta nói đến lập trình web. Nói đến HTML, người ta cũng nói đến làm web. Vậy không có lý gì HTML và PHP lại không đi được cùng với nhau! Bài 3 sẽ đề cập tới một vấn đề rất phổ biến khi lập trình PHP: Kết hợp mã PHP với HTML.

Trước hết, chúng ta hãy dành ít phút tìm hiểu cách thức hoạt động của World Wide Web (WWW).

Hãy tưởng tượng, bạn đang muốn truy cập trang web www.example.com/welcome.html. Bạn mở trình duyệt web, gõ vào ô địa chỉ: www.example.com/welcome.html và bấm Enter. Trang web sẽ hiện ra, gần như ngay tức khắc (ở đây không nói đến mạng dial up siêu chậm nhé 😊)

Vậy, điều gì đã xảy ra từ lúc bạn bấm Enter cho đến lúc trang web xuất hiện? Hãy cùng tớ xem xét những đoạn băng "behind the scene" này:

1. Ngay sau khi bạn bấm Enter, trình duyệt bạn đang dùng sẽ gửi một thông điệp (message) lên mạng, cho biết bạn đang muốn yêu cầu (request) trang www.example.com/welcome.html
2. Thông điệp đó được chuyển tới máy tính tại địa chỉ www.example.com/welcome.html
3. Máy chủ trên máy tính đó sẽ nhận được thông điệp và bắt đầu tìm kiếm file HTML được yêu cầu.
4. Máy chủ gửi file HTML đó về máy tính vừa yêu cầu (chính là máy tính của bạn). Nếu không tìm

thấy file HTML được yêu cầu, đơn giản là máy chủ sẽ trả lại một thông báo lỗi.

5. Trình duyệt của bạn, sau khi nhận về trang HTML, sẽ hiển thị nó ra màn hình.

Ở bước thứ 4, nếu file bạn yêu cầu là 1 file mang đuôi .php, thay vì gửi trả lại nội dung nguyên gốc của file, máy chủ sẽ lần lượt thực hiện thêm các bước:

1. Quét file trong chế độ HTML, gửi trả về nội dung HTML.

2. Ngay khi gặp <?php, máy chủ sẽ chuyển sang chế độ PHP, bắt đầu thực thi các lệnh PHP cho đến khi gặp ?>. Hiển nhiên nếu các lệnh PHP có output, máy chủ sẽ trả những output đó cho trình duyệt.

3. Kết thúc chế độ PHP (ra ngoài ?>), máy chủ quay lại chế độ HTML.

Quá trình cứ thế tiếp tục, cho đến khi kết thúc file .php.

Vậy là đã xong phần nói ngoài lề. Giờ ta bắt đầu vào Bài 3.

Ở Bài 1, tớ đã cùng các bạn viết chương trình đầu tiên, Hello World, bằng PHP. Giờ thử nhìn một file .php cũng mang nội dung Hello World:

HTML Code:

```
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>
```

Như ví dụ trên đây, các bạn có thể thấy: Đây chỉ đơn thuần là một file HTML, mang đuôi .php. Chẳng có gì đặc biệt! Và khi trình duyệt yêu cầu file này, máy chủ chỉ việc gửi trả nội dung nguyên gốc mà ko cần phải xử lý một chút lệnh nào cả.

Giờ hãy thử nâng cấp file .php đó bằng cách thêm vào nó một chút mã PHP:

PHP Code:

```
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <?php
      echo "<p>Hello World!</p>";
    ?>
```

```
</body>
</html>
```

Khi chạy script này, kết quả khi view source code cũng ko khác gì ví dụ đầu tiên. Chỉ có cách làm là khác, thay vì chỉ sử dụng HTML, ta kết hợp cả PHP và HTML trong cùng một file.

Giờ, nếu ta muốn in ra màn hình chữ Hello ở một dòng, và World ở một dòng, ta sẽ làm ntn?

Nếu các bạn có biết về HTML, thì sẽ nghĩ ngay đến thẻ `
`:

PHP Code:

```
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <?php
      echo "<p>Hello<br />World!</p>";
    ?>
  </body>
</html>
```

Kết quả output thật mỹ mãn và chẳng có gì đáng nói.

Tuy vậy, nếu các bạn còn nhớ, tớ đã từng nói \n có thể dùng để xuống dòng trong PHP. Vậy, thừa thặng xông lên, bạn sẽ thay `
` bằng `\n`:

PHP Code:

```
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <?php
      echo "<p>Hello\nWorld!</p>";
    ?>
  </body>
</html>
```

Kết quả? Thất bại. Trên màn hình, chữ Hello và World vẫn nằm cùng một dòng. Tại sao lại như

vậy? Làm thế nào để giải quyết vấn đề này?

Trả lời: \n đúng là để xuống dòng, nhưng đó là xuống dòng trong PHP output, nó ko đảm bảo việc xuống dòng khi cái PHP output đó được trình duyệt xử lý dưới dạng mã HTML.

Để trình duyệt xử lý chính xác những vấn đề ntn, ta cho toàn bộ xâu đó vào thẻ <pre>, thẻ quyết định việc giữ nguyên định dạng của xâu:

PHP Code:

```
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <?php
      echo "<p><pre>Hello\nWorld!</pre></p>";
    ?>
  </body>
</html>
```

Một lần nữa, kết quả hiện ra thật mỹ mãn.

Ta xét thêm một ví dụ nữa.

PHP Code:

```
<?php
  print_r($_SERVER);
?>
```

Script trên làm trò gì vậy ta? 🤖 Chưa cần biết print_r và \$_SERVER là gì, bạn chỉ cần thấy đoạn nó output ra mấy dòng sau: (tớ phải post ảnh vì UDS ko cho phép đưa đoạn ý vào bài viết)

```
Array ( [HTTP_USER_AGENT] => Opera/9.00 (Windows NT 5.1; U; en) [HTTP_HOST] => localhost [HTTP_
jpeg, image/gif, image/x-bitmap, */*;q=0.1 [HTTP_ACCEPT_LANGUAGE] => vi,en_US;q=0.9,en;q=0.8 [HT
[HTTP_ACCEPT_ENCODING] => deflate, gzip, x-gzip, identity, */*;q=0 [HTTP_COOKIE] => txp_name=QA;
[HTTP_COOKIE2] => $Version=1 [HTTP_CACHE_CONTROL] => no-cache [HTTP_CONNECTION] =>
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem [SystemRoot] => C:\WINDOWS [
=> .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH [WINDIR] => C:\WINDOWS [SERVER_SIG
Apache/2.0.55 (Win32) mod_autoindex_color mod_ssl/2.0.55 OpenSSL/0.9.8a PHP/5.0.5 Server at localho
[SERVER_SOFTWARE] => Apache/2.0.55 (Win32) mod_autoindex_color mod_ssl/2.0.55 OpenSSL/0.9.8a Pl
[SERVER_PORT] => 80 [REMOTE_ADDR] => 127.0.0.1 [DOCUMENT_ROOT] => E:/XAMPP/htdocs [SE
test.php [REMOTE_PORT] => 4227 [GATEWAY_INTERFACE] => CGI/1.1 [SERVER_PROTOCOL] => H
test.php [SCRIPT_NAME] => /test.php [PHP_SELF] => /test.php [argv] => Array () [argc] => 0 )
```

Phản ứng đầu tiên? Bạn sẽ thấy hơi chóng mặt phải ko ạ? Bạn sẽ tự hỏi: Nhiều thứ thế kia viết lú lú vào nhau thì ai mà đọc được? Tại sao ko tách dòng ra chứ?

À, nói đến tách dòng, bạn sẽ nhớ ngay tới thẻ `<pre>` mà tớ nói bên trên. Bạn sẽ thêm nó vào script của mình:

PHP Code:

```
<?php
    echo "<pre>";
    print_r($_SERVER);
    echo "</pre>";
?>
```

Kết quả trả về ko thể nói là dễ đọc, mà phải nói là rất dễ đọc 🤖, tuy (có thể) bạn chẳng hiểu cái gì sất!

Hãy tạm hài lòng với những gì mình vừa làm được và thư giãn một chút trước khi ta bước vào Bài 4.

Bật mí trước: Chúng ta đã biết tới hàm `var_dump` in ra kiểu và giá trị của biến, hàm `print_r` (bạn đoán là) in ra các giá trị của một cái `$_SERVER` gì đó. Vậy trong Bài 4, ta sẽ cùng tìm hiểu `print_r` là gì, sử dụng ra sao, và còn những hàm nào như vậy nữa.

Bài 4: Các toán tử

Để thực hiện việc tính toán các giá trị trong PHP, ta sử dụng toán tử (operator).

1. Gán (assignment)

Toán tử gán (dấu `=`) được sử dụng hết sức đơn giản. Ví dụ:

PHP Code:

```
<?php
    $a = 1;
    $b = 1;
    $c = "cool";
?>
```

Sau ví dụ, biến `$a` và `$b` mang giá trị 1, `$c` mang giá trị "cool".

Để cho ngắn gọn, thay vì phải mất 2 dòng khai báo `$a` và `$b`, ta có thể gộp:

PHP Code:

```
<?php
    $a = $b = 1;
    // Hoặc: $b = $a = 1;
?>
```

Kết quả vẫn đúng như mong đợi.

2. Toán tử số học (arithmetic)

Các toán tử này gồm có: + (cộng - addition), - (trừ - subtraction), * (nhân - multiplication), / (chia - division) và % (tính modul - modulus).

Ví dụ:

PHP Code:

```
<?php
    $a = 10;
    $b = 5;
    $c = $a + $b; // $c = 15
    $d = $c - $a; // $d = 5
    $e = $a / $b; // $e = 2
    $f = $e * $b; // $f = 10
    $g = $a % $e; // $g = 0
?>
```

Ngoài ra, để sau khi tính toán, giá trị \$a bằng \$a nhân 2 chẳng hạn, thay vì viết \$a = \$a * 2; ta có thể viết ngắn gọn: \$a *= 2;

Tương tự, có thể viết \$a += 10; \$a -= 1; \$a /= 3; \$a %= 1; Cấu trúc này rất giống C và C++, nên nếu bạn đã biết qua 2 ngôn ngữ này thì ko có gì phải bỡ ngỡ.

3. Toán tử so sánh (comparision)

Toán tử so sánh gồm những toán tử sau:

```
== Mang giá trị TRUE khi 2 vế mang cùng giá trị
=== Mang giá trị TRUE khi 2 vế mang cùng giá trị VÀ cùng kiểu
!= Mang giá trị TRUE khi 2 vế ko cùng giá trị
<> Mang giá trị TRUE khi 2 vế ko cùng giá trị
!== Mang giá trị TRUE khi 2 vế ko cùng giá trị HOẶC ko cùng kiểu
< Mang giá trị TRUE khi vế trái mang giá trị nhỏ hơn vế phải
> Mang giá trị TRUE khi vế trái mang giá trị lớn hơn vế phải
<= Mang giá trị TRUE khi vế trái mang giá trị nhỏ hơn hoặc bằng vế phải
>= Mang giá trị TRUE khi vế trái mang giá trị lớn hơn hoặc bằng vế phải
```

Ta sẽ viết là $\$a == \b , $\$a !== \b , $\$a > \b ...

Ví dụ:

PHP Code:

```
<?php
    "123" == 123    // Đúng
    "123" === 123  // Sai
    "123" === "123" // Đúng
?>
```

Còn một loại toán tử so sánh nữa, được viết dưới dạng:

$exp1 ? exp2 : exp3$

Ví dụ:

PHP Code:

```
<?php
    $a = ($b > 1) ? 2 : 1;
?>
```

Có thể giải thích như sau: Nếu $\$b > 1$ thì $\$a$ mang giá trị 2, còn ko $\$a$ mang giá trị 1.

4. Toán tử logic (logical)

Gồm có:

&& Mang giá trị TRUE nếu cả 2 vế đều là TRUE
 || Mang giá trị TRUE nếu một trong 2 vế là TRUE
 ! Mang giá trị TRUE nếu vế có giá trị FALSE
 xor Mang giá trị TRUE nếu có đúng 1 trong 2 vế là TRUE

Ta viết: $\$a \&\& \b , $\$a \text{ xor } \b ...

Có thể dùng "and" thay cho && và "or" thay cho || cũng ko sao.

5. Toán tử bit (bitwise)

Toán tử để xử lý bit bao gồm:

& Phép And
 | Phép Or
 ^ Phép Xor
 ~ Phép Not
 << Phép Shift Left
 >> Phép Shift Right

Có thể viết \$a >> 2, \$b | \$c,...

6. Toán tử dùng trong chuỗi

Để nối 2 chuỗi, ta dùng toán tử nối chuỗi (concatenation), biểu diễn bằng dấu chấm (.)

Ví dụ \$a . \$b, "Xâu" . "Một chuỗi khác"

Hiển nhiên có thể viết \$a .= "Một chuỗi nào đó"

6. Toán tử dùng trong mảng

+ Gộp 2 mảng (union)

== So sánh bằng, mang giá trị TRUE nếu các phần tử của 2 mảng mang cùng khóa & giá trị (có thể thứ tự khác nhau)

!= hoặc <> Mang giá trị TRUE nếu các phần tử của 2 mảng ko cùng khóa & giá trị

=== So sánh bằng, mang giá trị TRUE nếu các phần tử của 2 mảng mang cùng khóa & giá trị & thứ tự giống nhau

!== Mang giá trị TRUE nếu 2 mảng ko giống hệt nhau (non-identical)

7. Các toán tử khác

Các toán tử khác có thể kể đến toán tử tự tăng (auto-increment) và tự giảm (auto-decrement), ký hiệu tương ứng là ++ và --.

Ví dụ \$a++, \$b--, --\$c, ++\$d

Một toán tử khác là toán tử @, cho phép PHP bỏ qua lỗi của một lần gọi hàm.

Ví dụ:

```
$test = @file("Bạn ko có quyền sờ đến file này");
```

Tận dụng mã nguồn có sẵn bằng cách include file

Sau khi đọc xong bài Sử dụng hàm trong PHP, bạn sẽ có trong tay rất nhiều hàm cần thiết khi code. Số lượng hàm này chắc chắn sẽ tăng dần theo thời gian.

Ví dụ bạn đang viết rất nhiều hàm về tính toán với hình tròn:

PHP Code:

```
function tinh_dien_tich_hinh_tron($ban_kinh) {  
    // code  
}  
  
function tinh_chu_vi_hinh_tron($ban_kinh) {  
    // code  
}
```

```
// rất nhiều hàm khác
```

Thay vì copy và paste các hàm trên vào tất cả các file dính dáng đến việc tính toán với hình tròn, bạn hãy cho những hàm đó vào 1 file riêng, đặt tên là `hinh_tron.php` chẳng hạn. Mỗi lần muốn sử dụng các hàm trong file `hinh_tron.php` đó, bạn chỉ việc include nó vào file hiện tại bằng một trong các cách sau:

1. Dùng include

PHP Code:

```
include(tên_file);
```

2. Dùng require

PHP Code:

```
require(tên_file);
```

File được include có thể mang định dạng bất kỳ, `php`, `inc`, `lib`... tùy bạn chọn.

Câu hỏi 1: include với require làm gì?

Khi bạn include/require 1 file nào đó, ko cần biết file đó mang định dạng gì, PHP sẽ quét nội dung file đó và bắt đầu xử lý 2 trường hợp:

- Với những đoạn nằm trong thẻ `php`, PHP sẽ thực thi như với file PHP thông thường
- Với những đoạn nằm ngoài thẻ `php`, PHP sẽ output ra màn hình

Câu hỏi 2: include khác require ở chỗ nào?

Một file được include nếu (chẳng may) ko tồn tại sẽ khiến PHP báo lỗi, tuy nhiên phần còn lại của script vẫn sẽ được thực thi.

Ngược lại, một file được require nếu (xui xẻo) ko tồn tại sẽ khiến PHP đứng luôn, ko chạy tiếp script. Nói cách khác, file được require là file tối quan trọng, ảnh hưởng tới việc thực thi 1 script.

Ngoài việc sử dụng include và require, bạn còn có thể dùng `include_once` và `require_once`. Về cơ bản, include và `include_once` giống nhau. require và `require_once` cũng thế.

Điểm khác biệt là khi sử dụng `require_once` hay `include_once`, file đó chỉ được include đúng 1 lần duy nhất.

Hãy xét 1 ví dụ: Bạn có 3 script A, B, C. Trong đó A include B, B include C và C include lại A.

Nếu bạn chỉ dùng `include()` trong cả 3 trường hợp, chắc chắn PHP sẽ báo lỗi. Để tránh trường hợp này, bạn chỉ việc sửa 3 cái `include()` thành `include_once()` => Vấn đề được giải quyết!

Cấu trúc điều khiển trong PHP

Các câu lệnh điều kiện: các câu lệnh này cho phép chúng ta phân biệt các khối mã lệnh mà sẽ được thực thi chỉ khi gặp phải các điều kiện nào đó. PHP cung cấp hai cấu trúc lệnh điều kiện. Đầu tiên là if...elseif...else, cho phép chúng ta có thể kiểm tra một số lượng các biểu thức và thực thi các câu lệnh theo giá trị của chúng. Nếu chúng ta mong muốn kiểm tra một biểu thức đơn lẻ với một số lượng các giá trị, PHP cũng cung cấp một cấu trúc switch...case mà có thể làm đơn giản hoá đi phép toán này.

1) Câu lệnh If: Câu lệnh If là một trong những đặc tính quan trọng nhất của mỗi ngôn ngữ lập trình. Nó cho phép thực thi chọn lựa các dòng mã lệnh chỉ khi thoã mãn các điều kiện cụ thể.

Chẳng hạn:

PHP Code:

```
if ($country=="ca")echo ("Canada");//Canada được in ra khi biến $country là ca
```

Nếu nhiều hơn một câu lệnh được thực hiện khi thoã mãn điều kiện thì sử dụng dấu {} để chỉ ra những dòng lệnh nào là nằm trong khối if:

//Canada sẽ chỉ được in nếu biến \$country là ca

PHP Code:

```
if ($country=="ca"){  
  
    echo("Canada");  
  
    echo("Ottawa");  
  
}
```

Điều kiện được kiểm tra trong dấu {} phải trả về giá trị Boolean, hoặc là true hoặc là false. Cũng như bất kỳ điều kiện nào mà không được thoã mãn, zero hay là chuỗi rỗng (""), các giá trị không được định nghĩa thì tất cả đều trả về giá trị là false. Các điều kiện có thể được nối với nhau bằng các toán tử logic and (&&), or(||) và xor. Ví dụ như sau:

PHP Code:

```
if (((4 < 5) && (3 > 2)) xor (5 == 5)) echo ("This will not print");
```

Các điều kiện phân nhánh: Nếu điều kiện được kiểm tra mà trả về false, thì PHP cho phép ta chỉ ra một khối lệnh khác cần được thực hiện bằng cách dùng từ khoá else. Mỗi thứ trong khối mã lệnh thực thi điều kiện này được xem như là một phân nhánh và mỗi nhánh phải được định vị trong các dấu ngoặc nếu chức nhiều hơn một dòng lệnh. Ví dụ:

PHP Code:

```
if ($h < 0) {  
  
    echo ("Negative");  
  
} else {  
  
    echo ("Positive");  
  
}
```

PHP cũng cung cấp từ khoá elseif để kiểm tra các điều kiện lựa chọn nếu điều kiện trong câu lệnh if là không đúng. Một số câu lệnh elseif có thể được sử dụng với câu lệnh if. Nhánh else cuối cùng cho phép chúng ta định vị đoạn mã mà nên được thực hiện nếu cả điều kiện if và elseif đều không đúng.

PHP Code:

```
if ($h < 0) {  
  
    echo ("Negative");  
  
} elseif ($h == 0) {  
  
    echo ("Zero");  
  
} else {
```

```
    echo ("Positive");  
  
}
```

Ta cũng có thể kiểm tra những điều kiện hoàn toàn khác nhau khi sử dụng elseif:

PHP Code:

```
if ($country == "ca") {  
  
    // do something ...  
  
} elseif ($position == "h") {  
  
    // do something else ...  
  
}
```

Chú ý: cả hai điều kiện trên đều là true, nhưng chỉ có nhánh lệnh thứ nhất là được thực hiện.

Cũng có thể sử dụng các câu lệnh if lồng nhau trong câu lệnh if khác. Ví dụ:

PHP Code:

```
if ($country == "ca") {  
  
    if ($position == "h") {  
  
        echo ("Human resources positions in Canada.");  
  
    } elseif ($position == "a") {  
  
        echo ("Accounting positions in Canada.");  
  
    }  
  
}
```

Các câu lệnh trên cũng tương tự như sau:

PHP Code:

```
if ($country == "ca" && $position == "h") {  
  
    echo ("Human resources positions in Canada.");  
  
} elseif ($country == "ca" && $position == "a") {  
  
    echo ("Accounting positions in Canada.");  
  
}
```

PHP cũng cung cấp một cú pháp lựa chọn cho câu lệnh if, đó là if...endif. Ví dụ:

PHP Code:

```
if ($country == "ca"):  
  
    echo ("Canada");  
  
elseif ($country == "cr"):  
  
    echo ("Costa Rica");  
  
else:  
  
    echo ("the United States");  
  
endif;
```

2) Câu lệnh switch: được sử dụng khi một biến riêng rẽ đang được kiểm tra so với các giá trị khác.

Ví dụ:

PHP Code:

```
switch ($country) {
```

```
    case "ca":  
  
        echo ("Canada");  
  
        break;  
  
    case "uk":  
  
        echo ("the United Kingdom");  
  
        break;  
  
    default:  
  
        echo ("the United States");  
  
}
```

Khi câu lệnh switch thực hiện kiểm tra giá trị của biến \$country và so sánh nó với mỗi một trong các giá trị trong các mệnh đề case. Khi một giá trị thích hợp được tìm thấy, các câu lệnh kết hợp với case được thực hiện cho đến khi gặp câu lệnh break. Còn nếu không tìm ra được giá trị thích hợp nào thì câu lệnh default sẽ được thực hiện. Chú ý rằng lệnh switch trong PHP thì linh hoạt hơn nhiều so với hầu hết các ngôn ngữ khác. Không giống như C, Java và ngay cả JavaScript, các giá trị case cũng có thể là một trong các loại vô hướng, bao gồm tất cả các số, các chuỗi và ngay cả các biến. Ví dụ:

PHP Code:

```
$val = 6;$a = 5;$b = 6;  
  
switch ($val) {  
  
    case $a:  
  
        echo ("five");  
  
        break;  
  
    case $b:
```

```
        echo ("six");

        break;

    default:

        echo ("$val");

    }
```

Các mảng và các đối tượng chỉ là những loại dữ liệu là không phải là những nhãn đúng của case trong PHP.

Vòng lặp: Các vòng lặp chính là các phương tiện của việc thực thi một khối mã lệnh trong một số lần cho trước hay là cho đến khi gặp phải một điều kiện nhất định. PHP có hai loại vòng lặp: vòng lặp while kiểm tra điều kiện trước hay là sau mỗi bước tính lặp đi lặp lại và thực hiện lặp lại chỉ khi điều kiện là đúng. Một kiểu lặp khác là for, trong trường hợp này, số lượng bước tính lặp đi lặp lại được qui định trước khi lặp lần đầu và không thể bị thay đổi.

1. Vòng lặp while: là câu lệnh lặp đơn giản nhất. Cú pháp tương tự như câu lệnh if:

PHP Code:

```
while (condition) {

    //các câu lệnh

}
```

Một vòng lặp while sẽ kiểm tra một biểu thức Boolean. Nếu biểu thức là false thì đoạn mã bên trong dấu ngoặc móc sẽ được bỏ qua. Ngược lại, nếu có giá trị true thì đoạn mã bên trong dấu ngoặc móc sẽ được thực hiện. Khi gặp dấu } thì điều kiện kiểm tra sẽ được thực hiện lại và nếu có giá trị là true thì đoạn mã trong vòng lặp sẽ được thực hiện lại. Điều này sẽ tiếp tục cho đến khi gặp phải điều kiện. Chú ý rằng điều kiện chỉ được kiểm tra mỗi khi bắt đầu vòng lặp, bởi vậy ngay khi sự chính xác của điều kiện thay đổi trong suốt đoạn giữa của khối lệnh, thì mã lệnh sẽ vẫn được thực thi cho đến hết. Để thoát khỏi vào thời điểm sớm hơn, ta có thể sử dụng lệnh break. Ví dụ:

PHP Code:

```
 $\$i = 11;$ 

while (-- $\$i$ ) {
```

```
    if (my_function($i) == "error") {  
  
        break; // dừng vòng lặp!  
  
    }  
  
    ++$num_bikes;  
  
}
```

Trong ví dụ này, nếu ta hình dung rằng hàm `my_function` không trả về bất kì lỗi nào thì vòng lặp sẽ lặp đi lặp lại 10 lần và dừng lại khi biến `$i = 0`. Còn nếu `my_function` trả về lỗi, thì câu lệnh `break` sẽ được thực hiện và vòng lặp sẽ dừng lại. Có nhiều trường hợp mà chúng ta mong muốn kết thúc chỉ khi sự lặp lại hiện thời của vòng lặp không phải là toàn bộ vòng lặp của chính nó. Để đạt được điều này, ta sử dụng lệnh `continue`. Ví dụ:

PHP Code:

```
$i = 11;  
  
while (--$i) {  
  
    if (my_function($i) == "error") {  
  
        continue;  
  
    }  
  
    ++$num_bikes;  
  
}
```

Đoạn mã này cũng lặp đi lặp lại 10 lần nếu không có lỗi nào được trả về bởi hàm `my_function`. Tuy nhiên tại lúc này, nếu có lỗi xảy ra, việc thực hiện sẽ lướt qua sự lặp lại kế tiếp của vòng lặp, mà không tăng biến đếm `$num_bikes`. Giả sử biến `$i` vẫn lớn hơn 0, vòng lặp sẽ tiếp tục như bình thường.

2. Vòng lặp do...while: vòng lặp này cũng giống như while, ngoại trừ điều kiện được kiểm tra tại cuối mỗi vòng lặp, thay vì là ở đầu. Điều này có nghĩa là vòng lặp sẽ luôn luôn thực hiện ít nhất một lần.

Ví dụ:

PHP Code:

```
echo("<SELECT name='num_parts'>\n");

$i = 0;

do {

    echo("\t<OPTION value=$i>$i</OPTION>\n");

} while(++$i < $total_parts);

echo("</SELECT>\n");
```

Với đoạn mã trên, giá trị zero luôn luôn xuất hiện như là một tùy chọn trong thành phần <SELECT>, ngay cả nếu biến \$total_parts=0.

Các câu lệnh while và do...while thường được dùng với các toán tử tăng hay giảm để điều khiển khi nào thì bắt đầu và dừng như ví dụ trên. Các biến thường được dùng cho mục đích này đôi khi được định nghĩa như là các biến điều khiển vòng lặp. Thông thường sử dụng các câu lệnh while trong việc đọc các records từ một truy vấn cơ sở dữ liệu, từ các dòng trong một file hay là từ các nhân tố trong một mảng.

3. Vòng lặp for: Cấu trúc của vòng lặp for là khá phức tạp hơn mặc dầu các vòng lặp for thường tiện lợi hơn các vòng lặp while:

PHP Code:

```
for ($i = 1; $i < 11; ++$i) {

    echo("$i <BR> \n"); //\n từ 1 đến 10

}
```

Câu lệnh for chứa ba biểu thức bên trong dấu ngoặc đơn của nó, phân biệt với nhau bởi dấu chấm phẩy. Biểu thức thứ nhất là một câu lệnh gán để khởi tạo biến điều khiển vòng lặp. Câu lệnh này được thực thi chỉ một lần trước sự lặp lại lần đầu của vòng lặp. Biểu thức thứ hai là

biểu thức Boolean mà được thực thi tại đầu mỗi lần lặp. Nếu giá trị trả về là true thì vòng lặp sẽ tiếp tục thực hiện. Nếu là false thì vòng lặp kết thúc. Biểu thức thứ ba là một câu lệnh mà thực thi tại giai đoạn cuối của mỗi lần lặp của vòng lặp. Nó thường được dùng để tăng hay giảm các biến điều khiển vòng lặp.

Hàm (Functions) trong PHP

Không thể không nói đến hàm trong việc lập trình, nhờ có nó mà chương trình của chúng ta trở nên dễ dàng tổ chức hơn. Như các ngôn ngữ khác, PHP có khả năng cung cấp những hàm do người dùng tự định nghĩa. Đồng thời, PHP cũng có một số cải tiến để việc viết hàm được dễ chịu và mạnh mẽ hơn.

Định nghĩa và gọi hàm

Rất dễ để **định nghĩa một hàm trong PHP**:

PHP Code:

```
<?php

function tên_hàm([các tham số truyền vào ...])
{
    [thân hàm ...]
}

?>
```

- Từ khoá *function* báo cho PHP biết rằng đây là một hàm. Tiếp theo đó là tên hàm. Tên hàm của PHP có thể là bất cứ ký tự Unicode gì (kể cả tiếng Việt, tiếng Trung..., nhưng không được phép bắt đầu bằng số). Thật sự mạnh mẽ, nhưng bạn sẽ gặp vấn đề khi lưu file đó. Thôi thì cứ đặt tên không dấu là ổn nhất 😊. Ví dụ:

PHP Code:

```
<?php

function this_is_my_hàm()
{
    echo "Hoàn toàn hợp lệ !!!";
}

?>
```

- Sau tên hàm là danh sách tham số truyền vào và phần thân hàm. Phần thân hàm phải bắt đầu và kết thúc bằng cặp dấu { }. Phần thân này được thực thi khi tên hàm được gọi.

- Chú ý: mỗi tên hàm chỉ được định nghĩa một lần. Với một số ngôn ngữ khác, hàm có thể được gọi đi khi danh sách tham số truyền vào là khác nhau (Java chẳng hạn), nhưng PHP thì không có việc đó.

Gọi hàm cũng khá dễ. Bạn chỉ việc gọi tên hàm cùng danh sách tham số đi kèm. Hay hơn, việc gọi hàm **KHÔNG PHÂN BIỆT CHỮ HOA-CHỮ THƯỜNG**. Tuy nhiên, khuyến cáo là nên gọi hàm theo đúng tên hàm đã đặt, như thế để quản lý hơn.

PHP Code:

```
<?php

generate_left_menu_bar();
GeNeRaTe_LeFt_MEnu_BaR(); // cũng được, nhưng không nên dùng !!!
process_user_information($current_user, "new user", 65.0);
generate_copyright_notices();
generate_left_menu_bar; // Sai !! Vì không có dấu ()!!

?>
```

Chú ý ví dụ trên, khi gọi tên hàm, luôn phải có cặp dấu () nếu hàm không nhận tham số nào (còn nếu nhận tham số thì tất nhiên cặp dấu đó để chứa tham số rồi, phải không 🤔).

Ngừng việc thực thi hàm

- Vào bất cứ thời điểm nào trong quá trình thực thi hàm, bạn cũng đều có thể dừng công việc của hàm bằng từ khoá return.

PHP Code:

```
<?php

function work_work_work()
{
    $dow = date('l');

    if ($dow == 'Saturday' or $dow == 'Sunday')
    {
        // nghỉ việc vào cuối tuần
        return;
    }

    // work hard
    work_harder();
}

?>
```

- Khi mà hàm `work_work_work` được gọi vào thứ 7 hoặc Chủ nhật, nó trả về "không gì cả", còn nếu không, nó trả về giá trị "làm việc chăm chỉ hơn đi !!" (Ví dụ chỉ mang tính minh họa 🤪).

Đưa tham số vào hàm

- Ví dụ cho một cấu trúc cơ bản:

PHP Code:

```
<?php

function my_new_function($param1, $param2, $param3, $param4)
{
    echo <<<DONE
    You passed in: <br/>
    \$param1: $param1 <br/>
    \$param2: $param2 <br/>
    \$param3: $param3 <br/>
    \$param4: $param4 <br/>

    DONE;
}

?>
```

- Khi đưa một số tham số vào hàm, bạn phải phân cách chúng bằng dấu phẩy (.). Bạn có thể truyền bất kỳ tham số nào vào hàm, bất kể là biến, hằng số.. hoặc thậm chí là một hàm khác:

PHP Code:

```
<?php

// gọi hàm với nhiều loại tham số truyền vào
my_new_function($userName, 6.22e23, pi(), $a or $b);

?>
```

Giá trị trả về của hàm

- Thông thường, người ta lập trình hàm chỉ để xử lý một công việc nhất định mang tính lặp lại, và giá trị trả về của hàm là không có (null). Nhưng không hẳn tất cả mọi trường hợp đều như vậy:

PHP Code:

```
<?php

function is_even_number($number)
{
    if (($number % 2) == 0)
        return TRUE;
    else
        return FALSE;
}

?>
```

-> Hàm trên có giá trị trả về là một giá trị boolean True hoặc False.

Lời kết:

Hàm là một công cụ rất mạnh trong PHP. Việc sử dụng hàm không chỉ để tối ưu các đoạn code, nó còn làm cho chương trình dễ đọc hơn và thích hợp để làm trong một nhóm với nhau.

Chúc bạn thành công,

iSheep 🐑

Kiểu dữ liệu PHP (tiếp theo - bon tren)

PHP hỗ trợ tám kiểu dữ liệu nguyên thủy.

Bốn kiểu thông thường là: **boolean, integer, floating-point number(float), string**.

Hai kiểu phức tạp là: **mảng(array) và đối tượng (object)**. Và cuối cùng là hai kiểu đặc biệt : **resource và NULL**. Loại dữ liệu của biến thông thường không được gán bởi người lập trình mà được quyết định tại thời gian chạy của PHP, phụ thuộc vào ngữ cảnh mà biến được dùng.

1. Boolean: đây là kiểu đơn giản nhất. Một kiểu boolean biểu thị một giá trị thật. Nó có thể là TRUE hay FALSE.

Cú pháp: để chỉ định một giá trị boolean, có thể sử dụng từ khoá TRUE hay là FALSE. Cả hai đều không phân biệt chữ hoa hay chữ thường.

Ví dụ:

Code:

```
$foo=True; // gán giá trị TRUE cho biến $foo.
```

Để có thể chuyển một giá trị sang kiểu **boolean**, chúng ta có thể dùng (bool) hay (boolean). Tuy nhiên trong hầu hết các trường hợp bạn không cần phải sử dụng việc ép kiểu này, bởi giá trị sẽ được tự động chuyển nếu nó là một toán tử, hàm hay là cấu trúc điều khiển đòi hỏi một tham số kiểu boolean.

Chú ý: -1 được xem là TRUE, giống như các giá trị khác 0 khác (bất kể là số dương hay âm).

2. Integer: là một tập hợp bao gồm các số {...,-2,-1,0,1,2,...}.

Cú pháp: Integer có thể được chỉ định trong cơ số 10, cơ số thập lục phân hay cơ số bát phân, tùy chọn đi trước bởi dấu - hay +. Nếu bạn sử dụng với cơ số bát phân, bạn phải theo thứ tự với 0 đứng trước, còn đối với số thập lục phân thì 0x.

Ví dụ như sau:

Code:

```
$a = 1234; # số thập phân

$a = -123; # số âm

$a = 0123; # số bát phân

$a = 0x1A; # số thập lục phân
```

Kích thước của kiểu dữ liệu này là 32bit, và PHP không hỗ trợ kiểu unsigned integer. Nếu bạn chỉ định một số vượt qua biên của kiểu dữ liệu integer, nó sẽ được xem như kiểu float. Tương tự như vậy, khi bạn thực hiện một phép toán mà kết quả trả về là một số vượt qua biên của kiểu integer, thì kiểu float sẽ được trả về. Tuy nhiên, có một lỗi trong PHP mà không phải bao giờ điều này cũng đúng, nó liên quan đến các số âm. Chẳng hạn, khi bạn thực hiện `-50000 * $million`, kết quả sẽ là `429496728`. Tuy nhiên, khi cả hai toán tử đều là số dương thì không có vấn đề gì xảy ra.

Để chuyển một giá trị sang kiểu integer, ta có thể dùng toán tử ép kiểu (`int`) hay (`integer`). Tuy nhiên, trong hầu hết các trường hợp bạn không cần phải dùng toán tử ép kiểu đó, bởi giá trị sẽ được tự động chuyển sang nếu toán tử, hàm hay cấu trúc điều khiển đòi hỏi một đối số integer.

3. Kiểu số thực (`floats,doubles,hay real numbers`) : có thể được chỉ định bằng cách sử dụng một trong các cú pháp sau:

Code:

```
$a = 1.234; $a = 1.2e3; $a = 7E-10;
```

Kích cỡ của kiểu float tùy thuộc vào platform, giá trị lớn nhất là xấp xỉ `1.8e308`

4. **String**: là những chuỗi các kí tự. Trong PHP, một kí tự cũng tương tự như một byte, do đó có chính xác 256 kí tự khác nhau.

Cú pháp: có thể khai báo bằng ba cách khác nhau như sau:

- Dấu nháy đơn: cách dễ dàng nhất để chỉ định một chuỗi đơn giản là đóng nó trong một dấu nháy đơn. Ví dụ: `echo 'le bao vy';`
- Dấu nháy kép: nếu chuỗi được đóng trong dấu nháy kép(""), PHP hiểu sẽ có thêm các chuỗi cho các kí tự đặc biệt Ví dụ: `\n;\t;\;\;\$;...`
- Heredoc: các khác để phân định chuỗi là sử dụng cú pháp ("`<<<`"). Chỉ nên cung cấp một định danh sau `<<<`, sau đó là chuỗi và tiếp là cùng tên định danh để đóng dấu nháy. Định danh dùng để đóng phải bắt đầu bằng cột đầu tiên của dòng. Định danh được dùng phải có tên giống như trong các quy luật đặt tên biến trong PHP.

5. **Mảng** : là một danh sách các phần tử có cùng kiểu dữ liệu. Mảng có thể là mảng một chiều hay nhiều chiều.

- Mảng một chiều có chỉ mục: là mảng được quản lý bằng cách sử dụng chỉ số dưới kiểu integer để biểu thị vị trí của giá trị yêu cầu. Cú pháp: `$name[index1];`

Ví dụ: một mảng một chiều có thể được tạo ra như sau:

Code:

```
$meat[0]="chicken";  
  
$meat[1]="steak";  
  
$meat[2]="turkey";
```

Nếu bạn thực thi dòng lệnh sau: `print $meat[1];` thì trên trình duyệt sẽ hiển thị dòng sau: `steak`.

Bạn cũng có thể sử dụng hàm **array()** của PHP để tạo ra một mảng. Ví dụ:

Code:

```
$meat=array("chicken","steak","turkey");
```

- Mảng một chiều kết hợp: rất thuận lợi khi dùng để ánh xạ một mảng sử dụng các từ hơn là sử dụng các integer, nó giúp ta giảm bớt thời gian và các mã yêu cầu để hiển thị một giá trị cụ thể.

Ví dụ: bạn muốn ghi lại tất cả các thức ăn và các cặp rượu ngon.

Code:

```
$pairings["zinfandel"] = "Broiled Veal Chops";

$pairings["merlot"] = "Baked Ham";

$pairings["sauvignon"] = "Prime Rib";
```

Một cách khác là bạn có thể sử dụng hàm **array()** của PHP để tạo ra một mảng loại này, ví dụ như sau:

Code:

```
$pairings = array( zinfandel => "Broiled Veal Chops",

merlot => "Baked Ham",sauvignon => "Prime Rib",

sauvernes => "Roasted Salmon");
```

· Mảng nhiều chiều có chỉ mục: chức năng của nó cũng giống như mảng một chiều có chỉ mục, ngoại trừ việc nó có thêm một mảng chỉ mục được dùng để chỉ định một phần tử. Cú pháp: \$name[index1] [index2]..[indexN];

Một mảng hai chiều có chỉ mục được tạo ra như sau:

Code:

```
$position = $chess_board[5][4];
```

· Mảng đa chiều kết hợp: khá hữu ích trong PHP. Giả sử bạn muốn ghi lại các cặp rượu-thức ăn, không chỉ loại rượu, mà cả nhà sản xuất. Bạn có thể thực hiện như sau:

Code:

```
$pairings["Martinelli"] ["zinfandel"] = "Broiled Veal Chops";

$pairings["Beringer"] ["merlot"] = "Baked Ham";

$pairings["Jarvis"] ["sauvignon"] = "Prime Rib";
```

6. Object: bạn có thể xem object như là một biến mà minh họa một kiểu mẫu template được gọi là class. Khái niệm của đối tượng và lớp được sử dụng nhiều trong ngôn ngữ lập trình hướng đối tượng OOP. Không giống như các kiểu dữ liệu khác trong PHP, object phải được khai báo. Điều quan trọng là phải nhận ra rằng object không hơn gì một minh họa của một lớp, và hoạt động như là một khuôn mẫu cho việc tạo các object có các đặc tính và chức năng cụ thể. Cho nên, lớp(class) phải được định nghĩa trước khi khai báo một object. Để khởi tạo một đối tượng, bạn sử dụng câu lệnh new để minh họa đối tượng với một biến. Ví dụ:

Code:

```
<?php

class foo{

    function do_foo(){

        echo "Doing foo.";

    }

}

$bar = new foo;

$bar->do_foo();

?>
```

7. Resource: là một biến đặc biệt, chứa một tham chiếu đến một resource bên ngoài. Các resource được tạo ra và sử dụng bởi các hàm đặc biệt.

Giải phóng resources: bởi do tham chiếu đếm của hệ thống được giới thiệu trong PHP4 Zend-engine, nó sẽ tự động phát hiện khi một resource không cần thiết cho lâu dài. Khi ở trong trường hợp này, tất cả các resource mà đã được dùng cho resource này được giải phóng bởi "bộ phận thu nhặt rác". Do đó, hiếm khi thật sự cần thiết để giải phóng bộ nhớ thông thường bằng cách sử dụng hàm **free_result()**.

8. NULL: giá trị NULL đặc biệt dùng để thể hiện một biến không có giá trị. Một biến được xem là NULL nếu:

- o Nó được gán giá trị hằng số NULL.
- o Nó chưa được khởi tạo giá trị nào.
- o Nó là hàm **unset()**

Chú thích: unset () là một hàm dùng để hủy bỏ các biến chỉ định.

Cú pháp: chỉ có một loại giá trị của kiểu NULL. Bạn có thể khai báo như ví dụ sau:

Code:

```
$var=NULL;
```

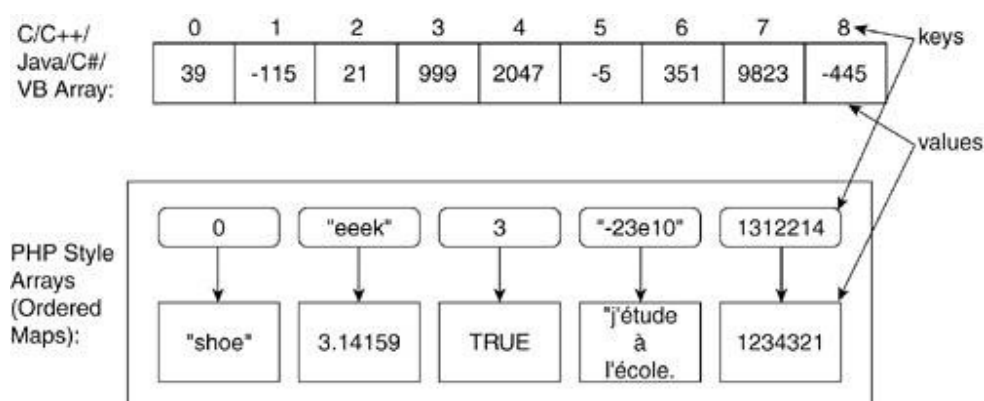
Làm việc với mảng trong PHP

Các vấn đề chính sẽ được đề cập:

- Làm thế nào để tạo 1 mảng trong PHP.
- Cách duyệt qua tất cả các phần tử trong 1 mảng.
- Sơ lược về mảng nhiều chiều

Mở đầu

Mảng là một cách hiệu quả để nhóm một lượng dữ liệu lại với nhau thành một khối duy nhất. Mảng trong PHP cũng như các ngôn ngữ khác (C/Java/VB), nó bao gồm 2 phần: khoá và giá trị (key và value). Nhưng vẫn có sự khác biệt, các khoá và giá trị trong PHP được dùng một cách tự do, không theo một ràng buộc quá mức nào cả. Bạn hoàn toàn có thể dùng một chuỗi để làm khoá, cũng như nhóm các giá trị không cùng kiểu dữ liệu với nhau lại làm thành một mảng. 😊



Sự khác biệt trong cách sử dụng mảng của PHP với các ngôn ngữ khác

Tạo và thêm dữ liệu vào mảng

Mảng được tạo dễ dàng bằng từ khoá `array`, và có thể thêm dữ liệu ngay trong phần nội dung của mảng. Nếu bạn thích dùng một key theo ý thích của mình thì có thể dùng toán tử `=>` để thêm dữ liệu (value) cho key đó. 😊

Hơi khó hiểu, nhưng bạn hãy xem những ví dụ dưới đây để biết rõ hơn về cách tạo mảng cũng như cách lưu trữ giá trị trong mảng của PHP:

PHP Code:

```
<?php
// PHP tự động gán key khi bạn tạo một mảng, bắt đầu từ key 0
```

```
// trong ví dụ dưới, key 0 có giá trị là "Piper",  
// tương tự cho key 1, 2 và 3.  
$airplanes = array("Piper", "Cessna", "Beech", "Cirrus");  
  
// Chúng ta cũng có thể tạo key theo ý thích của chúng ta,  
// không bắt buộc phải là số  
$home = array("size" => 1800, "style" => "ranch",  
             "yearBuilt" => 1955, "numBeds" => 3,  
             "numBaths" => 2, "price" => 150000);  
?>
```

PHP Code:

```
<?php  
  
// key của một mảng không bắt buộc bắt đầu phải là 0.  
// Chẳng hạn, key 123 được dùng làm key bắt đầu trong ví dụ này.  
$noises[123] = "hisssssss";  
  
// và khi khai báo như thế này, key tiếp theo của mảng sẽ là 124  
$noises[] = "gobble gobble";  
  
?>
```

Truy xuất vào 1 phần tử của mảng

Bạn có thể truy xuất phần tử của mảng bằng cách gọi key của nó:

PHP Code:

```
<?php  
// ví dụ về cách gọi key là 1 số  
$breads = array("baguette", "naan", "roti", "pita");  
echo "I like to eat ". $breads[3] . "<br/>\n";  
  
$computer = array("processor" => "Muncheron 6000",  
                 "memory" => 2048, "HDD1" => 80000,  
                 "graphics" => "NTI Monster GFI q9000");  
  
// ví dụ về cách gọi key là 1 chuỗi
```

```
echo "My computer has a " . $computer['processor']  
    ." processor<br/>\n";
```

```
?>
```

Xoá phần tử khỏi mảng

Để xoá 1 phần tử nào đó của mảng, bạn dùng từ khoá unset cho phần tử đó:

PHP Code:

```
<?php  
  
$drinks = array("Coffee", "Café au Lait", "Mocha", "Espresso",  
    "Americano", "Latte");  
unset($drinks[3]); // xoá phần tử "Mocha" khỏi mảng.
```

```
?>
```

Còn muốn xoá toàn bộ phần tử của mảng, bạn cũng dùng từ khoá unset, nhưng cho toàn bộ mảng:

PHP Code:

```
<?php  
  
unset($drinks); // mảng $drinks giờ đã bị xoá sạch dữ liệu
```

```
?>
```

Đếm số phần tử của mảng

Sử dụng từ khoá *count*:

PHP Code:

```
<?php  
  
$drinks = array("Coffee", "Café au Lait", "Mocha", "Espresso",  
    "Americano", "Latte");  
$elems = count($drinks);  
  
// kết quả sẽ là 6.  
echo "The array \$drinks has $elems elements<br/>\n";
```

```
?>
```

II. Duyệt tất cả các phần tử của mảng

Vòng lặp foreach

PHP Code:

```
foreach (array as [key =>] values)
    kh&#7889;i lệnh
```

Vòng lặp này sẽ duyệt qua từng phần tử một trong mảng, nó sử dụng một biến cho trước để tạo một bản copy phần tử mà nó đang duyệt tới và xử lý trên biến đó. Vòng lặp kết thúc khi không còn phần tử nào để duyệt. 🤖

PHP Code:

```
<?php

$drinks = array("Coffee", "Café au Lait", "Mocha", "Espresso",
               "Americano", "Latte");
foreach ($drinks as $drink)
{
    echo "We serve $drink<br/>\n";
}

?>
```

Vòng lặp thông thường (for)

Vòng lặp for hoàn toàn có thể được dùng để duyệt qua tất cả các key của mảng: 😊

PHP Code:

```
<?php

$drinks = array("Coffee", "Café au Lait", "Mocha", "Espresso",
               "Americano", "Latte");

for ($x = 0; $x < count($drinks); $x++)
{
```

```
    echo "We serve '$drinks[$x]'\n";
}

?>
```

III. Mảng nhiều chiều (Multi-Dimensional Arrays)

Rất nhiều trường hợp bạn muốn lưu trữ nhiều mảng trong 1 mảng có sẵn. Khi đó, chúng ta có mảng một chiều. Và rất may mắn là PHP hỗ trợ rất mạnh mẽ và dễ dàng trong việc tạo mảng nhiều chiều. 🍷

Thật vậy, đây là cách mà mảng nhiều chiều được tạo trong PHP 😊:

PHP Code:

```
<?php

$bikes = array();
$bikes["Tourmeister"] = array("name" => "Grande Tour Meister",
    "engine_cc" => 1100,
    "price" => 12999);
$bikes["Slasher1000"] = array("name" => "Slasher XYZ 1000",
    "engine_cc" => 998,
    "price" => 11450);
$bikes["OffRoadster"] = array("name" => "Off-Roadster",
    "engine_cc" => 550,
    "price" => 4295);

?>
```

Còn đây là cách truy xuất vào các phần tử của mảng nhiều chiều:

PHP Code:

```
<?php

$names = array_keys($bikes);

foreach ($names as $name)
{
    print $bikes[$name] . " costs: " . $bikes[$name]["price"]
        . "\n";
}
```

```
}  
?>
```

IV. Lời kết

Mảng là một khai báo rất dễ dàng trong PHP. Việc học về mảng thật ra không có gì khó, chỉ cần đọc qua các ví dụ, bạn cũng hoàn toàn có thể rút ra cho mình được phương thức mà PHP tạo một mảng đơn giản. 😊

Have fun. 😊

Tìm kiếm và thay thế trong chuỗi với Regular Expression

Ở bài trước, chúng ta đã xem xét qua một số hàm thường gặp khi xử lý chuỗi trong PHP. Để tìm kiếm trong chuỗi, ta có thể dùng `strpos()` hoặc `substr()`, nhưng với những hàm này ta chỉ có thể tìm kiếm một cách hết sức hạn chế. Hãy thử tưởng tượng, nếu bạn muốn kiểm tra xem 1 chuỗi có phải là một địa chỉ IP, hay một địa chỉ email đúng đắn hay ko, sẽ phải sử dụng rất nhiều câu lệnh `if`.

Trong những trường hợp như thế này, ta sẽ sử dụng Regular Expression. Regular Expression, viết tắt là RegEx, có rất nhiều định nghĩa. Đây là một trong số các định nghĩa đơn giản nhất:

Regular Expression là một cách thức thể hiện dữ liệu dưới dạng các ký tự đại diện. Nó được dùng trong các thuật toán tìm kiếm, thay thế chuỗi.

Đây là ví dụ về một RegEx dùng để kiểm tra xem một chuỗi có phải là địa chỉ IP đúng đắn hay ko:

Code:

```
(([0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3})\.([0-9]{1,3}))
```

Nhìn thì có vẻ rất phức tạp chứ thực ra RegEx trên rất đơn giản và dễ hiểu một khi bạn đã nắm được cách sử dụng.

Ta sẽ quay trở lại ví dụ trên sau khi đã học được một số điều cần thiết. Giờ hãy dành thời gian vào việc tìm hiểu cú pháp, cách viết 1 RegEx:

1.RegEx CÓ phân biệt ký tự hoa - thường. (case sensitive)

Ví dụ ta có một chuỗi như sau:

Code:

```
Hello, UDS
```

Khi đó RegEx **Hello** sẽ phù hợp với phần đầu của chuỗi nói trên, còn **hello** thì ko.

2. Mọi ký tự trong RegEx đều ứng với một ký tự trong chuỗi cần kiểm tra, kể cả ký tự trắng (dấu cách, dấu tab, dấu xuống dòng).

Ví dụ với chuỗi:

Code:

```
Hello, UDS
```

Thì **Hello, UDS** sẽ phù hợp còn **Hello, UDS ko**.

3. Một số ký tự có ý nghĩa đặc biệt. Ký tự **^** chỉ sự bắt đầu một chuỗi, còn **\$** chỉ sự kết thúc.

Ví dụ: Chuỗi

Code:

```
UDS is UDS
```

^UDS sẽ phù hợp với đoạn UDS đầu chuỗi, trong khi **UDS\$** sẽ phù hợp với đoạn UDS cuối chuỗi.

4. Cũng như trong PHP, ký tự **** được sử dụng để escape một số ký tự đặc biệt. Ví dụ **\\$, \^, \-**

Chuỗi:

Code:

```
$abc$
```

\\$ phù hợp với ký tự \$ đầu chuỗi.

5. Ký tự **.** phù hợp với mọi ký tự

Ví dụ:

RegEx **...** phù hợp với 3 ký tự đầu trong chuỗi

Code:

```
UDS is a great community!!!
```

Dĩ nhiên, để 1 ký tự trong RegEx phù hợp với dấu **.** (thật) thì cần phải escape dấu **.** ấy như thế này **\.**

Ví dụ:

Code:

```
O.K.
```

\. sẽ phù hợp với dấu . thứ nhất sau ký tự O.

6. Một danh sách các ký tự có thể đặt trong dấu ngoặc vuông []. Khi đó bất cứ ký tự nào trong ngoặc vuông được tìm thấy, ký tự đó sẽ được coi là phù hợp. Trật tự các ký tự trong ngoặc là ko quan trọng.

Ví dụ:

Code:

```
How do you do?
```

[oyu] sẽ phù hợp với ký tự o trong từ How

[dH]. sẽ phù hợp với ký tự Ho trong từ How.

7. Một dải (range) các ký tự có thể được thể hiện bằng cú pháp [-]. Có thể có nhiều dải trong một cặp ngoặc [].

Ví dụ:

Code:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
```

[C-K] sẽ phù hợp với ký tự C.

[a-d] sẽ phù hợp với ký tự a.

[C-Ka-d2-6] sẽ phù hợp với ký tự C.

8. Nếu một lớp các ký tự đặt trong dấu [] được mở đầu bằng ký tự ^, những ký tự đó sẽ được coi là ko phù hợp.

Ví dụ:

Code:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz 0123456789
```

[^CDghi45] sẽ ko phù hợp với các ký tự C, D, g, h, i, 4, 5.

9. Các xâu khác nhau có thể được đặt trong dấu () và phân cách bằng ký tự |.

Ví dụ:

Code:

```
Monday Tuesday Friday
```

(on|ues|rida) sẽ phù hợp với đoạn on trong từ Monday, ues trong từ Tuesday,...

10. Có thể chỉ ra số lần ký tự sẽ xuất hiện. Ký tự * phù hợp với "ko hoặc nhiều hơn thế", + phù hợp với "một hoặc nhiều hơn thế", ? phù hợp với "ko hoặc một".

Ví dụ:

Code:

```
aabc abc bc
```

a*b hoặc **a+b** phù hợp với aab.

a?b phù hợp với ab.

11. Dấu ngoặc móc {} được sử dụng để chính xác hóa số lượng ký tự ta mong muốn. Trong đó:

{m} cho biết ký tự xuất hiện **ĐÚNG m lần**

{m,n} cho biết ký tự xuất hiện **ÍT NHẤT m lần và NHIỀU NHẤT n lần.**

{m,} cho biết ký tự xuất hiện **ÍT NHẤT m lần.**

{,n} cho biết ký tự xuất hiện **NHIỀU NHẤT n lần.**

Ví dụ:

Code:

```
One ring to bring them all and in the darkness bind them
```

.{5} sẽ phù hợp với đoạn ký tự *One r*.

[els]{1,3} sẽ phù hợp với ký tự *e*.

[a-z]{3,} sẽ phù hợp với đoạn *ring*.

Cần chú ý: "*", "+", và "?" là trường hợp đặc biệt của luật thứ 11. "*" tương ứng với {0,}, "+" tương ứng với {1,} còn "?" tương ứng với {0,1}.

Vậy, trên đây tớ đã giới thiệu một số luật quan trọng và cần nhớ khi sử dụng Regular Expression để tìm kiếm và thay thế trong xâu. Bài tiếp theo sẽ nói tới một số ví dụ về Regular Expression và việc áp dụng vào PHP.

Các bạn có thể tham khảo thêm về Regular Expression ở Zvon:

RegEx Tutorial: <http://www.zvon.org/other/PerlTutori...put/index.html>

RegEx Reference: <http://www.zvon.org/other/reReference/Output/index.html>

Chính bài viết này cũng đã được dịch từ RegEx Tutorial. 🇻🇳

Tương tác với server bằng form

Việc tương tác với server bằng form có lẽ là một trong những công việc mà các bạn rất hay gặp khi lập trình web. Hãy nghĩ đơn giản, dưới góc độ một user trong UDS, bạn đã phải tương tác với bao nhiêu là form: form register, form login, form post bài nhanh - post bài advance, form để report cho mod... Nhiều quá hóa... chóng mặt 🤔

Các form đó đều có điểm chung: Đều được viết bằng HTML. Hãy cùng nhắc lại đôi chút về cách tạo form với HTML:

Trước hết, bạn hãy dành 30 giây trong quỹ thời gian "ít ỏi" của mình để ngẫm lại: HTML có cấu trúc như thế nào?

Nói một cách ngắn gọn, một file HTML có cấu tạo như sau:

HTML Code:

```
<html>
  <head>
    <title><!-- Tiêu đề --></title>
  </head>
  <body>
    <!-- Nội dung -->
  </body>
</html>
```

Form trong HTML được đặt trong phần <body></body>. Ví dụ như sau:

HTML Code:

```
<form action="process.php" method="GET">
  Tên: <input type="text" name="name" /><br />
  Tuổi: <input type="text" name="age" /><br />
  <input type="submit" value="Submit!" />
</form>
```

Đoạn code HTML trên có tác dụng gì? Rất đơn giản, nó giúp tạo ra 1 form với 2 ô để nhập dữ liệu: Tên và tuổi. Kèm theo đó là 1 nút lệnh mang chữ Submit. Rất giản dị và ko có gì khó hiểu

ở đây cả.

À, có 1 điều này cần giải thích: Ở dòng đầu tiên của đoạn code, thẻ form có 2 thuộc tính: action và method. Hai thuộc tính này có mục đích gì?

1. action="process.php" nghĩa là sau khi bấm submit, dữ liệu sẽ được chuyển qua file process.php nằm cùng thư mục với file HTML này
2. method="GET" nghĩa là phương thức truyền dữ liệu sẽ là GET.

Vậy, bạn đã biết dữ liệu sẽ được chuyển qua file process.php, mà hiển nhiên file process.php này chưa tồn tại, nên một điều rất tự nhiên là ta sẽ tạo file process.php với nội dung sau:

PHP Code:

```
<?php
$name = $_GET["name"];
$age = $_GET["age"];
echo "Tên bạn: $name";
echo "Tuổi của bạn: $age";
?>
```

Hai dòng cuối của ví dụ trên có lẽ chẳng có j` để bàn. Chỉ đơn giản là echo 2 biến \$name và \$age ra màn hình. Vấn đề nằm trong 2 dòng đầu.

Ta lấy dòng thứ 1 để mở xẻ: \$name = \$_GET["name"]; Ở đây ai cũng biết giá trị của \$name được gán từ \$_GET["name"]. Vậy là chỉ còn duy nhất 1 điều cần thắc mắc: \$_GET là gì?

\$_GET là mảng để chứa các giá trị được chuyển từ form sang. Ở đây \$_GET gồm 2 phần tử: \$_GET["name"] và \$_GET["age"].

Mọi vấn đề đến đây được giải quyết.

Giờ ta thử xét form ở Ví dụ ban đầu. Bạn hãy làm một phép thay đổi nhỏ: Đổi method="GET" bằng method="POST".

Khi đó hiển nhiên process.php sẽ ko hoạt động 😞

Nếu suy diễn một cách educated, bạn sẽ ngay lập tức sửa sai bằng cách thay đổi \$_GET bằng \$_POST. Kết quả? Đúng goài 🤖

Đến đây ta có thể sung sướng rút ra kết luận: Để nhận biến từ form chuyển sang, có thể sử dụng mảng \$_GET hoặc \$_POST, tùy vào phương thức truyền dữ liệu của form.

Bài học kết thúc ở đây được rồi nhỉ? 🤖

Chưa đâu bạn ạ. Còn một điều chưa nói đến: Sự khác nhau giữa \$_GET và \$_POST là j`? Hay đúng hơn, sự khác nhau giữ method GET và POST là gì?

Nói đến GET và POST, trước hết ta sẽ nói đến HTTP Request.

Một HTTP Request được gửi tới server có định dạng như sau:

Code:

```
<request-line>
<headers>
<blank line>
[<request-body>]
```

Một HTTP Request được gửi tới trang web www.abc.com từ trình duyệt Firefox sẽ tương tự như ví dụ sau:

Code:

```
GET / HTTP/1.1
Host: www.abc.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)
          Gecko/20050225 Firefox/1.0.1
Connection: Keep-Alive
```

Tương tự, một Request gửi đến trang www.abc.com/def sẽ có dạng như sau:

Code:

```
GET /def/ HTTP/1.1
Host: www.abc.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)
          Gecko/20050225 Firefox/1.0.1
Connection: Keep-Alive
```

Như các bạn có thể thấy, 2 ví dụ trên đều dùng phương thức GET để request dữ liệu.

Giờ hãy tưởng tượng, sau khi điền vào form 1 giá trị name là admin, age là 20, bạn bấm nút submit để gửi dữ liệu đến file process.php.

Trên thanh address sẽ hiện ra url: process.php?name=admin&age=20

Nếu xem xét một cách kỹ lưỡng HTTP Request, ta sẽ thấy nó như sau:

Code:

```
GET /process.php?name=admin&age=20 HTTP/1.1
Host: www.abc.com
```

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)
          Gecko/20050225 Firefox/1.0.1
Connection: Keep-Alive
```

Tóm lại, khi dùng phương thức GET, trình duyệt sẽ gửi Request tới server với các tham số đặt trong dòng đầu tiên (request-line).

Còn với phương thức POST thì sao? Thay nằm trong request-line, những tham số này được đặt trong phần request-body.

Ví dụ ta giữ nguyên form trên, chỉ thay method="GET" bằng method="POST" và bấm Submit. Khi đó đây sẽ là HTTP Request "behind the scene":

Code:

```
POST / HTTP/1.1
Host: www.abc.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.7.6)
          Gecko/20050225 Firefox/1.0.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 40
Connection: Keep-Alive

name=admin&age=20
```

Có thể dễ dàng thấy rằng, ngoài việc thêm một số dòng vào phần header:

Code:

```
Content-Type: application/x-www-form-urlencoded
Content-Length: 40
Connection: Keep-Alive
```

thì phương thức POST đã "cất" các tham số của mình trong phần request-body. Điều này hiển nhiên khác hẳn phương thức GET.

Bài học hôm nay chính thức được kết thúc ở đây <:-P

(và tớ cũng đi xem phim smallville tiếp đây 🍿)

Xâu - Các phép toán thường gặp

Tớ xin nhắc lại một cách ngắn gọn: Xâu là một tập hợp các ký tự. Ví dụ "abc", "sadsks12dsfsdfjkee123" có thể coi là một xâu. Trong PHP, xâu được thể hiện bằng mã ký tự 8 bit (8-bit character code) và được coi là nằm trong bảng mã ISO-8859-1.

Giờ ta hãy xem xét một số phép toán thường gặp trên xâu:

1. strlen - lấy độ dài 1 xâu

Có lẽ các bạn cũng dễ dàng đoán được, strlen là string length - độ dài xâu. Cách sử dụng hàm strlen() rất đơn giản:

PHP Code:

```
<?php
echo strlen("Updatesofts");
?>
```

Kết quả in ra sẽ là 11 - số ký tự trong xâu "Updatesofts".

2. trim - cắt bỏ phần thừa trong xâu

Hàm trim() nhận tham số là 1 xâu và nó sẽ loại bỏ mọi khoảng trắng (whitespace) bắt đầu và kết thúc xâu. Cần chú ý ở đây, khoảng trắng ko chỉ bao gồm dấu cách (" ", mã ASCII 32) mà nó còn gồm:

- Tab ("\t", mã ASCII 9)
- Dấu xuống dòng ("\r" và "\n", mã tương ứng 10 và 13)
- Ký tự NULL ("\0", mã 0)
- Tab dọc (vertical tab - mã 11). Dấu này giờ là "của hiếm". ầu

Ví dụ:

PHP Code:

```
<?php
$str = " \t\t\t \nXâu này lắ m thứ linh tinh thể nhờ \r\n \t \t ";
echo trim($str);
?>
```

Kết quả output ra hoàn toàn có thể đoán được: "Xâu này lắ m thứ linh tinh thể nhờ"

3. ltrim và rtrim

Sau khi biết về hàm trim(), chắc hẳn bạn sẽ đặt câu hỏi: Thế nhờ tớ chỉ muốn cắt bỏ mấy phần linh tinh ở đầu/cuối xâu thôi thì sao?

Trả lời: Hoàn toàn có thể. Hãy dùng ltrim() và rtrim() - bạn sẽ thik ngay mà

ltrim dùng để bỏ các phần linh tinh bắt đầu xâu.

rtrim dùng để bỏ các phần linh tinh kết thúc xâu.

4. strpos - tìm kiếm trong xâu

Hàm strpos() nhận 3 tham số:

1. 1 xâu
2. xâu cần tìm trong xâu trên
3. bắt đầu tìm từ ký tự thứ mấy trong xâu, mặc định là 0

Ví dụ `strpos("Updatesofts", "Update")` sẽ trả về kết quả là 0. `strpos("ABCABC", "A", 2)` sẽ trả về kết quả là 3.

Vậy nếu ko tồn tại xâu cần tìm thì sao? Đơn giản lắm bạn ạ, kết quả trả về sẽ là `FALSE`.

Một điều cần chú ý nữa: Nếu vị trí bắt đầu tìm là số âm (ví dụ -1), PHP sẽ tìm kiếm từ cuối xâu trở lại (ngược với cách tìm mặc định)

Giờ hãy xét 1 ví dụ:

PHP Code:

```
<?
$res = strpos($haystack, $needle);
if ($res == FALSE) {
    echo "Ko thấy!";
} else {
    echo "Thấy òi!";
}
?>
```

Bạn có `$haystack` là "Updatesofts", `$needle` là "Up", hỏi PHP sẽ echo ra cái gì? Bạn đoán là "Thấy òi" đúng ko?

Tiếc là sai rồi bạn ạ Kết quả là "Ko thấy", vì `$needle` được tìm thấy ở ĐẦU `$haystack`, nghĩa là vị trí thứ 0, mà 0 lại đồng nghĩa với `FALSE`

Khá là confusing phải ko hả bạn?

Để xử lý trường hợp này, ta sẽ dùng toán tử so sánh `===` thay vì `==` (bằng bằng bằng thay vì bằng bằng). PHP sẽ kiểm tra cả giá trị và kiểu của biến, do đó 0 và `FALSE` sẽ là 2 khái niệm hoàn toàn khác nhau và vấn đề đã được giải quyết gọn ghẽ.

5. substr - tách (extract) 1 phần trong xâu

Cú pháp của hàm `substr()` như sau:

`substr(xâu, vị trí bắt đầu, [số ký tự - nếu cần])`

Ví dụ: `substr($str, 1)` trả về xâu bắt đầu từ ký tự thứ 1. `substr($str, 3, 2)` trả về 2 ký tự của xâu bắt đầu từ ký tự thứ 3.

Cũng như `strpos`, tham số thứ 2 có thể là âm. Khi đó PHP sẽ xử lý ngược từ cuối.

Trên đây là 5 hàm thường gặp khi xử lý xâu trong PHP. Hiển nhiên việc liệt kê tất cả các hàm là impossible, vì vậy nếu bạn cần thêm về các hàm xử lý xâu trong PHP, hãy tìm đến PHP.net:

<http://www.php.net/manual/en/ref.strings.php>

Coming Up Next: Sử dụng Regular Expression để tìm kiếm và thay thế trong xâu.

TẠO FORM ĐỂ UPLOAD FILE

Form để upload file cần thoã mãn các điều kiện sau:

- * method là POST
- * enctype là multipart/form-data

Mã HTML của form sẽ tựa tựa như sau:

```
<form method="POST" enctype="multipart/form-data" action="process_upload.php">
<input type="hidden" name="MAX_FILE_SIZE" value="30000">
<input type="file" name="file_upload" size="20">
<input type="submit" value="Upload">
</form>
```

Đoạn code trên sẽ tạo 1 form với 1 nút Browse... để bạn chọn file cần upload, và 1 nút Upload để bạn submit form. Form sẽ được submit tới file process_upload.php nằm cùng thư mục với file chứa form.

Một số browser support MAX_FILE_SIZE sẽ kiểm tra dung lượng file trước khi form được submit, tuy nhiên không phải browser nào cũng vậy. Cho nên bạn đừng nên tin tưởng tuyệt đối vào server! Ở ví dụ trên, nếu browser hỗ trợ, nhưng file có dung lượng lớn hơn 30000 byte sẽ được browser thông báo lỗi khi submit form.

XỬ LÝ DỮ LIỆU ĐƯỢC SUBMIT LÊN SERVER

Bây giờ ta hãy xem xét tới phần xử lý dữ liệu được submit lên server trong file process_upload.php. PHP lưu thông tin về file được upload lên server trong biến global \$_FILES. Với form ở ví dụ trên, PHP sẽ truyền cho script process_upload.php các thông tin sau:

- * \$_FILES['file_upload']['name']: tên file gốc trên máy client. Tùy vào browser, tên file có thể được truyền lên server ở dạng C:\folder\filename.ext hoặc chỉ là filename.ext. Chương trình phải tự kiểm tra và trích ra tên file nếu cần thiết.
- * \$_FILES['file_upload']['type']: kiểu của file, được lưu ở dạng MINE (Ví dụ: image/gif, audio/wav).
- * \$_FILES['file_upload']['size']: dung lượng của file tính theo byte.
- * \$_FILES['file_upload']['tmp_name']: sau khi upload, server sẽ lưu file vào một file tạm trên server, biến này cho ta biết đường dẫn và tên của file tạm đó. Chương trình sẽ đọc file tạm này để lấy nội dung của file được upload.
- * \$_FILES['file_upload']['error']: mã lỗi, chương trình nên kiểm tra biến này để bảo đảm rằng quá trình upload không xảy ra lỗi.
 - o UPLOAD_ERR_OK (= 0): không có lỗi, quá trình upload thành công.
 - o UPLOAD_ERR_INI_SIZE (= 1): dung lượng file upload vượt quá giới hạn được chỉ định trong file php.ini.
 - o UPLOAD_ERR_FORM_SIZE (= 2): dung lượng file upload vượt quá giới hạn được chỉnh định bởi MAX_FILE_SIZE.
 - o UPLOAD_ERR_PARTIAL (= 3): file chỉ được upload 1 phần (có thể là do lỗi đường truyền trong quá trình upload).
 - o UPLOAD_ERR_NO_FILE (= 4): không có file nào được upload (có thể là file ở client không

tồn tại).

Khi đã có toàn bộ các thông tin cần thiết, xử lý file như thế nào là quyền định của bạn. Bạn có thể đọc nội dung của file và lưu vào database, hoặc di chuyển file và lưu vào thư mục upload của bạn. Sau đây là 1 ví dụ của file process_upload.php.

Đầu tiên, kiểm tra xem tác vụ có phải là upload hay không:

```
if ( $_SERVER["REQUEST_METHOD"] != "POST" ) {
//thông báo lỗi không phải là method POST
//và thoát
exit(-1);
} //end if
```

Tiếp theo kiểm tra xem quá trình upload có lỗi gì không:

```
if ( !isset($_FILES["file_upload"]["error"]) ||
$_FILES["file_upload"]["error"] != 0 ) {
//thông báo lỗi dựa vào giá trị của $_FILES["file_upload"]["error"]
//và thoát
exit(-1);
} //end if
```

```
//ta cũng có thể kiểm tra xem dung lượng file có vượt quá giới hạn
//của chương trình hay không
if ( $_FILES["file_upload"]["size"] > $MAX_FILE_SIZE ) {
//thông báo lỗi
//và thoát
exit(-1);
}
```

Tách tên file từ client:

```
$temp = preg_split('/[\\\/]+/', $_FILES["file_upload"]["name"]);
$filename = $temp[count($temp)-1];
```

```
//ta cũng có thể kiểm tra phần mở rộng của file nếu cần thiết
if ( !preg_match('/\.(gif|jpg)$/i', $filename) ) {
//thông báo lỗi file upload không phải là dạng GIF hoặc JPG
//và thoát
exit(-1);
} //end if
```

Và cuối cùng, lưu file được upload vào nơi cần thiết:

```
$upload_dir = "/home/nbthanh/public_html/uploads/";
$upload_file = $upload_dir . $filename;
if ( move_uploaded_file($_FILES["file_upload"]["tmp_name"], $upload_file) ) {
//file đã được upload và copy sang thư mục lưu trữ thành công
} else {
//có lỗi xảy ra
} //end if
```

CÁC HÀM PHP ĐƯỢC DÙNG TRONG VÍ DỤ

- * `exit`: dừng/thoát chương trình ngay lập tức.
- * `isset`: kiểm tra xem biến có tồn tại hay không. Trong ví dụ của bài viết, ta dùng hàm `isset` để kiểm tra xem biến `$_FILES["file_upload"]["error"]` có tồn tại hay không.
- * `preg_split`: tách một chuỗi thành từng phần nhỏ theo regular expression. Trong ví dụ của bài viết, ta dùng hàm này để tách tên file cùng đường dẫn ra thành từng phần nhỏ (phân cách nhau bằng ký tự `\` hoặc `/`, ta không biết chắc được client là Windows hay Linux nên ta tách theo trường hợp tổng quát). Sau khi tách, phần tử cuối cùng sẽ là tên file. Một cách khác để lấy tên file là dùng hàm `basename`. Tuy nhiên sử dụng hàm này sẽ có một số vấn đề nảy sinh, bạn tham khảo thêm ở đây: <http://www.php.net/manual/en/function.basename.php>.
- * `count`: đếm số lượng phần tử trong mảng. `$a[count($a)-1]` sẽ truy cập tới phần tử cuối cùng của mảng `$a`.
- * `preg_match`: sử dụng regular expression để tìm xem chuỗi con có xuất hiện trong chuỗi mẹ hay không. Trong ví dụ của bài viết, ta dùng hàm này để kiểm tra xem tên của của có được kết thúc bằng `.gif` hoặc `.jpg` hay không.
- * `move_uploaded_file`: di chuyển file được upload từ client đến 1 thư mục khác trên server.

TÀI LIỆU THAM KHẢO

- * PHP Manual: <http://www.php.net/manual/en/index.php>
 - o Regular Expression Functions (Perl-Compatible): <http://www.php.net/manual/en/ref.pcre.php>
 - o Handling file uploads: <http://www.php.net/manual/en/features.file-upload.php>
- * Từ Google: từ khoá php tutorial upload file

Source from [DDTH](#)