CHƯƠNG 3

Các cấu trúc điều khiển, Vòng lặp, Hàm tự tạo, Mảng

Các chủ đề chính

Mục tiêu

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

- Trình bày được bản chất cũng như sự khác nhau giữa các câu lệnh điều kiện, vòng lặp.
- Vận dụng được các câu lệnh điều kiện và vòng lặp vào bài toán thực tế.
- Vận dụng được cách chèn file bằng câu lệnh include hoặc require trong PHP.
- Xây dựng được các hàm tự tạo và ứng dụng trong các bài toán cụ thể.
- Biết cách tạo mảng và truy xuất các phần ử của mảng.
- Vận dụng được các hàm sắp xếp trong mảng khi mảng được đánh chỉ mục tuần tự hoặc không tuần tự

Câu hỏi kiểm tra mở đầu

Trả lời các câu hỏi sau

1. Để giải quyết bài toán tính `n!` theo bạn chúng ta nên sử dụng?
   a. Câu lệnh `if`
   b. Câu lệnh `switch`
   c. Vòng lặp `for`
   d. Vòng lặp `while`
2. Theo bạn, đoạn mã sau hiển thị ra cái gì?

```
$i = 5;
Do
{
     echo $i;
     $i--;
}
```

```
While $i > 5;
```

a. Không hiển thị ra gì cả

b. 5, 4, 3, 2, 1

c. 5

d. 5, 4, 3, 2, 1, 0

3. Biến toàn cục là?
   a. Biến có phạm vi toàn chương trình
   b. Biến có phạm vi trong hàm
   c. Không có biến toàn cục
4. Trong C++ mảng có số chỉ mục bắt đầu là?

   a. 0

   b. 1

   c. Tùy ý
5. Để duyệt mảng, thông thường chúng ta sử dụng
   a. Vòng lặp `while`
   b. Vòng lặp `do ... while`
   c. Vòng lặp `for`

1.Các cấu trúc điều khiển

1.1 Câu lệnh điều kiện

Câu điều kiện cho phép chúng ta xác định khối mã sẽ được thực hiện chỉ khi một số điều kiện được đáp ứng. PHP cung cấp hai loại câu lệnh điều kiện. Đầu tiên là `if ... elseif ... else`, cho phép chúng ta kiểm tra một số biểu thức và thực thi câu lệnh theo giá trị của chúng. Nếu chúng ta muốn kiểm tra một biểu thức duy nhất đối với một số giá trị, PHP cũng cung cấp một câu lệnh `switch ... case`.

Câu lệnh if

Câu lệnh `if` là một trong những tính năng quan trọng nhất của hầu hết các ngôn ngữ lập trình. Nó cho phép một trong những lựa chọn của các dòng mã lệnh được thực thi chỉ khi điều kiện được xác định là đúng. Ví dụ:

```
// Canada sẽ hiển thị nếu $country là ca
if ($country == "ca") echo ("Canada");
```

Nếu có nhiều hơn một câu lệnh được thực thi khi điều kiện đúng, dấu ngoặc nhọn { } được sử dụng để xác định những dòng thuộc bên trong khối `if`:

```
// Canada sẽ hiển thị nếu $country là ca
if ($country == "ca") {
     echo ("Canada");
     echo (" Ottawa");
}
```

Điều kiện nhánh

Nếu điều kiện được kiểm tra trả về `sai`, PHP cho phép chúng ta xác định khối mã lệnh khác để thực thi bằng cách sử dụng từ khóa

else. Mỗi điều kiện thực thi các khối mã được biết đến như là một nhánh, và mỗi nhánh phải được đặt trong vòng dấu ngoặc nhọn, nếu nó chứa nhiều hơn một dòng mã:

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

PHP cũng cung cấp từ khóa elseif để kiểm tra các điều kiện luân phiên nếu điều kiện trong phần if là sai. Bất kỳ một câu lệnh elseif nào đều có thể được sử dụng trong câu lệnh if. Nhánh else kết thúc cuối cùng để cho phép chúng ta xác định mã lệnh sẽ được thực thi nếu không có điều kiện nào của if hoặc elseif là đúng.

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

Nó là có thể, và thậm chí phổ biến để kiểm tra các điều kiện hoàn toàn khác nhau khi sử dụng elseif:

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

Chú ý rằng nếu tất cả các điều kiện là đúng, chỉ có nhánh đầu tiên sẽ được thực thi.

Nó cũng phổ biến để tạo các câu lệnh `if` trong câu lệnh `if` khác.

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

Đoạn mã trên tương đương với:

```
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 thay thế cho câu lệnh `if`: `if: ... endif;`

```
    if ($country == "ca"):
        echo ("Canada");
    elseif ($country == "cr"):
        echo ("Costa Rica");
    elseif ($country == "de"):
        echo ("Germany");
    elseif ($country == "uk"):
        echo ("the United Kingdom");
    else:     // Phải là "us"
        echo ("the United States");
    endif;
```

Lưu ý là dấu ngoặc nhọn không được sử dụng và các điều kiện kiểm tra được theo sau bởi dấu hai chấm. Câu `endif` cuối cùng được sử dụng để biểu hiện sự kết thúc của khối thay thế cho việc đóng cặp }. Cú pháp thay thế này đặc biệt hữu ích nếu chúng ta muốn tạo khối HTML, Javascript, CSS bên trong câu lệnh `if` của PHP:

```
<?php if ($country == "ca"):  ?>
<TABLE>
   <CAPTION>Canada</CAPTION>
   <TR>
      ...
   </TR>
</TABLE>
<?php elseif ($country == "cr"):  ?>
<TABLE>
   <CAPTION>Costa Rica</CAPTION>
   <TR>
      ...
   </TR>
</TABLE>
<?php endif;  ?>
```

Trong ví dụ trên, bảng `Canada` sẽ được tạo ra nếu mã đất nước là "`ca`" và bảng `Costa Rica` nếu mã đất nước là "`cr`".

Switch

Chúng ta sẽ tạo ra Form Hồ sơ xin việc gồm có một biến `$country` với ràng buộc chỉ gồm 2 ký tự. Giả sử chúng ta muốn kiểm tra biến này và hiển thị ra tên đầy đủ của đất nước, chúng ta có thể sử dụng câu lệnh `if…elseif…else` như trên:

```
    if ($country == "ca") {
         echo ("Canada");
    } elseif ($country == "cr") {
         echo ("Costa Rica");
    } elseif ($country == "de") {
         echo ("Germany");
    } elseif ($country == "uk") {
         echo ("the United Kingdom");
```

```
    } else {     // Must be "us"
         echo ("the United States");
    }
```

Trong ví dụ trên, chúng ta kiểm tra liên tục giá trị của biến $country, mặc dù nó không thay đổi từ dòng tiếp theo. Chúng ta sử dụng câu lệnh switch để tránh sự hạn chế trên. Switch được sử dụng khi một biến duy nhất đang được kiểm tra với nhiều giá trị:

```
switch ($country) {
   case "ca":
      echo ("Canada");
      break;
   case "cr":
      echo ("Costa Rica");
      break;
   case "de":
      echo ("Germany");
      break;
   case "uk":
      echo ("the United Kingdom");
      break;
   default:     // Phải là "us"
      echo ("the United States");
}
```

Trong ví dụ trên, nếu $contry bằng “ca”, “us”, hoặc “cr”, North America sẽ được hiển thị. Nếu nó bằng “uk” hoặc “de”, Europe sẽ hiển thị. Chú thích “fall through” chỉ ra rằng chúng ta không sử dụng break. Đây là một dạng lập trình tốt bằng cách đưa những chú thích này vào để chứng minh rằng lỗi sẽ không được thực hiện. Khi câu lệnh switch được sử dụng trong hàm, người ta thường sử dụng return để dừng việc thực thi thay cho break.

Người lập trình quen thuộc với các ngôn ngữ lập trình khác nên lưu ý rằng switch trong PHP linh hoạt hơn nhiều so với hầu hết các ngôn ngữ lập trình khác. Không giống C, Java và thậm chí JavaScript, các giá trị có thể là trường hợp của bất kỳ loại vô hướng nào, bao gồm tất cả các số và chuỗi và chúng thậm chí có thể là biến.

```
$val = 6;

$a = 5;
$b = 6;
$c = 7;

switch ($val) {
    // In JavaScript, it would be illegal to use a variable as a case label.
   // Not in PHP!
   case $a:
      echo ("five");
      break;
   case $b:
      echo ("six");
      break;
   case $c:
      echo ("seven");
      break;
   default:
      echo ("$val");
}
```

1.2 Vòng lặp

Lập trình sẽ là một nghề khá khó chịu nếu không có vòng lặp. Vòng lặp là một phương tiện để thực hiện một khối mã lệnh nếu điều kiện đúng, hoặc cho đến khi một điều kiện nhất định được đáp ứng. PHP có hai loại vòng: vòng lặp while kiểm tra các điều kiện trước hoặc sau mỗi lần lặp và tiếp tục lặp lại khi điều kiện còn đúng. Các loại khác của vòng lặp là vòng lặp for; trong trường hợp này, số lặp là cố định trước khi vòng đầu tiên, và không thể thay đổi.

Vòng lặp while

Vòng lặp while là lệnh lặp đơn giản nhất. Cú pháp khá giống với câu lệnh if:

```
while (condition) {
   // statements
}
```

Vòng lặp while đánh giá một biểu thức logic. Nếu biểu thức là sai, các mã lệnh trong dấu móc nhọn sẽ được bỏ qua. Nếu đúng, các mã lệnh trong dấu móc nhọn sẽ được thực thi. Khi dấu móc nhọn đóng } được đọc, điều kiện sẽ được tái kiểm tra lại và nếu điều kiện còn đúng, các mã lệnh trong vòng lặp được thực thi lại. Quá trình này tiếp tục cho đến khi điều kiện được đáp ứng.

```
$i = 11;
while (--$i) {
   if (my_function($i) == "error") {
      break;  // Dừng vòng lặp!
   }
   ++$num_bikes;
}
```

Trong ví dụ trên, nếu hàm ảo my_function không trả về bất kỳ lỗi nào, vòng lặp sẽ lặp lại 10 lần và dừng khi biến $i bằng 0. (Nhớ

rằng 0 được đánh giá là sai (false).) Nếu my_function trả ra lỗi, câu lệnh break được thực thi và kết thúc vòng lặp. Trong một số tình huống chúng ta mong muốn việc kết thúc chỉ lặp lại trong vòng lặp hiện thời, không phải toàn bộ vòng lặp. Để làm điều này, chúng ta sử dụng continue.

```
$i = 11;
while (--$i) {
   if (my_function($i) == "error") {
      continue;  // Skip ahead to next iteration.
                 // Don't increment $num_bikes
   }
   ++$num_bikes;
}
```

Đoạn mã trên cũng lặp 10 lần nếu không có lỗi được trả về từ `my_function`. Tuy nhiên, nếu một lỗi xảy ra, nó sẽ bỏ qua để thực hiện vòng lặp tiếp theo, không tăng biến đếm `$num_bikes`. Giả sử `$i` vẫn còn lớn hơn 0, vòng lặp sẽ tiếp tục như bình thường.

Giống như câu lệnh `if, while` thường là một cú pháp luân phiên, nó có ích cho việc nhúng các khối mã HTML.

```
<?php
   $i = 0;
   while ($i <= 5):
?>
<TR><TD><INPUT type=text></TD></TR>
<?php
   ++$i;
   endwhile;
?>
```

Vòng lặp Do…While

Câu lệnh `do … while` giống với câu lệnh `while`, trong câu lệnh này điều kiện được kiểm tra lúc kết thúc mỗi lần lặp chứ không phải lúc bắt đầu. Điều này có nghĩa là vòng lặp sẽ luôn được thực thi ít nhất một lần.

```
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 này, 0 sẽ luôn xuất hiện như một tùy chọn trong phần tử `<SELECT>`, thậm chí nếu `$total_parts` bằng `0`.

Câu lệnh `while` và `do … while` thường được sử dụng để tăng hoặc giảm các tác toán tử điều khiển khi bắt đầu và kết thúc như trong ví dụ trên. Biến được sử dụng cho mục đích này là thỉnh thoảng tham chiếu tới các biến điều khiển lặp. Thông thường sử dụng câu lệnh while để đọc các bản ghi từ việc truy vấn cơ sở dữ liệu, các dòng từ một tệp hoặc các phần tử của mảng. Những chủ đề này sẽ được đề cập trong phần sau.

Vòng lặp for

Cú pháp của vòng lặp for là khá phức tạp, các vòng lăp for thường tiện lợi hơn các vòng lặp while.

```
for ($i = 1; $i < 11; ++$i) {
   echo ("$i <BR> \n");  // Prints from 1 to 10
}
```

Câu lệnh for yêu cầu 3 biểu thức trong dấu ngoặc đơn của nó, cách nhau bởi dấu chấm phẩy “;”. Đầu tiên là một câu lệnh gán để

khởi tạo biến điều khiển lặp. Câu lệnh này được thực thi duy nhất một lần, trước lần lặp đầu tiên của vòng lặp. Thứ hai là một biểu thức logic để đánh giá lúc bắt đầu của mỗi lần lặp. Nếu biểu thức là đúng, vòng lặp được xử lý. Nếu sai, vòng lặp kết thúc. Thứ ba là một câu lệnh để thực thi việc kết thúc của mỗi lần lặp. Nó thường được sử dụng để tăng hoặc giảm biến điều khiển lặp.

Biểu thức giữa thường kiểm tra biến điều khiển lặp dựa vào giá trị đã định nghĩa trước đó, nhưng đây đây không phải là một trường hợp. Một vòng lặp sau đây là hoàn toàn hợp lệ.

```
for ($i = 1; my_function($i) != "error"; ++$i) {
   // Do something with $i until my_function returns
an error
}
```

Tuy nhiên, đoạn mã này sẽ đơn giản hơn nếu chúng ta dùng vòng lặp while.

```
$i = 1;
while (my_function($i) != "error") {
// Thực hiện các công việc với $i cho tới khi
my_function trả về một lỗi.
   ++$i;
}
```

1.3 require và include files trong PHP

PHP cung cấp hai câu lệnh require và include, cả hai câu lệnh này đọc và thực thi mã từ file xác định. Điều này cho phép chúng ta viết những hàm, các biến và các mã khác để tái sử dụng và lưu trữ chúng trong một file để có thể sau đó truy cập bằng bất kỳ kịch bản nào khác của chúng ta. Câu lệnh require thay thế toàn bộ nội dung của file được chỉ định. Điều này thích hợp cho mọi hoàn cảnh mà chúng ta muốn tạo ra các hàm và các biến dùng chung tới kịch bản của chúng ta.

```
<?php
   define ("COMPANY", "Phop's Bicycles");
   define ("NL", "<BR>\n");
?>
```

Bây giờ chúng ta có thể thiết lập file common mới của chúng ta, chúng ta có thể tham chiếu tới nó từ bất kỳ các files khác của chúng ta bằng cách thêm vào:

```
     require ("common.php");
     echo (COMPANY . NL);
```

Khi câu lệnh require được thay thế với nội dung của file common.php, các hằng của chúng ta được thừa nhận trong câu lệnh echo. Một bất lợi của hành động thay thế này là require không sử dụng được trong vòng lặp để gọi file khác trong mỗi lần lặp của vòng lặp. Đây là nơi để chúng ta đưa include vào.

Câu lệnh include cũng truy cập mã của một file bên ngoài, nhưng nó đánh giá và thực thi mã trong file bên ngoài mỗi lần để câu lệnh include được bắt gặp, hơn là chỉ thay thế mã bên ngoài của nó một lần lúc bắt đầu thực thi. Giả sử chúng ta có ba files có tên là file1.php, file2.php và file3.php, chúng ta muốn đưa vào trong một trang PHP. Với include, chúng ta có thể làm điều này:

```
for ($i = 1; $i <= 3; ++$i) {
   include("file" . $i . ".php");
}
```

Nếu chúng ta thử làm điều này với require, nội dung của `file1.php` sẽ thay thế câu lệnh `require` trong lần lặp đầu tiên của vòng lặp và đoạn mã đó cúng được thực thi lại trong lần lặp kế tiếp.

Cả `require` và `include` đều thừa nhận file bên ngoài là HTML, vì vậy nội dung của các files bên ngoài sẽ được xử lý như

HTML và không PHP, trừ khi chúng ta tránh HTML khi chúng ta đã làm trong `common.php` với cặp thẻ `<?php ... ?>`.

PHP cũng cung cấp các chỉ dẫn `auto_prepend_file` và auto_append_file để có thể thiết lập trong file php.ini. Những chỉ dẫn này cho phép chúng ta tự động require một file bên ngoài lúc bắt đầu hoặc kết thúc mọi file PHP được phục vụ.

1.4 Thoát một trang PHP

Nếu một lỗi nghiêm trọng xảy ra (ví dụ, nếu chúng ta thất bại trong việc kết nối đến một cơ sở dữ liệu), Nó có thể không thực hiện phần còn lại của tập lệnh PHP của chúng ta chạy. Trong những trường hợp này, chúng ta có thể muốn hiển thị một thông báo lỗi, dừng thực thi trang PHP của chúng ta ngay lập tức và thoát khỏi trang. Để làm điều này, PHP cung cấp câu lệnh exit. Đây là một trong những câu lệnh đơn giản để làm chủ trong PHP. Nó chỉ đơn giản dừng tất cả việc thực thi, giống như câu lệnh break cho toàn bộ tài liệu. Bất kỳ mã - PHP, HTML, Javascript, hoặc mã khác - xuất hiện sau exit sẽ không được thực hiện:

```
if (my_function ($i) == "error") {
   echo ("<BR><B>Có lỗi xảy ra.</B><BR>\n" .
         "không thể kết thúc việc nạp tài liệu.<BR>\n");
   exit;
}
```

2. Hàm tự tạo

Hàm là một khối mã mà có thể được định nghĩa một lần và sau đó được gọi từ các phần khác của chương trình. Điển hình, hàm lấy một đối số hoặc tập các đối số, thực hiện một tập các thao tác được định nghĩa trước và trả về một giá trị kết quả. Các hàm cho phép chúng ta viết nhiều môđum, các ứng dụng được cấu trúc một cách hợp lý.

Bằng cách viết và kiểm tra các hàm tái sử dụng, chúng ta có thể lưu một lần và làm giảm số lỗi trong mã của chúng ta. PHP có nhiều hàm được xây dựng sẵn, chẳng hạn như gettype() và isset(). Trong phần này, chúng ta sẽ học làm thế nào để tạo ra các hàm do người dùng tự định nghĩa.

2.1 Chúng làm việc như thế nào

Hàm được khai báo với câu lệnh function. Ví dụ, tính lũy thừa 3 của một số:

```
     // Khai báo và định nghĩa hàm
     function cube($num) {
          return $num * $num * $num;
     // Trả về $num thành lũy thừa 3
     }
// Gọi hàm cube():
echo (cube(6));                    // In ra 216
```

Dòng đầu tiên của mã trên có dạng:

```
function Tên_hàm(parameters) {
   Thân hàm
}
```

Tên của hàm (“cube” trong trường hợp này) đi sau từ khóa function và các tham số (nếu có) xuất hiện trong dấu ngoặc đơn và cách nhau bởi dấu phẩy. Thân hàm phải được đặt giữa cặp móc nhọn.

Để gọi một hàm, chỉ cần sử dụng tên của nó theo sau là cặp ngoặc đơn và bao gồm các đối số trong đó nếu có.

Gọi hàm `cube(6)` trong ví dụ trên cho kết quả là một biểu thức với giá trị là `216`. Giá trị mà hàm trả về được quyết định bởi câu lệnh `return` bên trong than hàm. Khi `return` được thực thi, hàm dừng và sự thực thi lại tiếp tục với những dòng gọi hàm, giá trị được trả về thay thế cho gọi hàm. Do vậy, dòng `echo cube(6);` tương đương với `echo (216);` .

Hàm không nhất thiết phải có giá trị trả về. Hàm sau đây không sử dụng câu lệnh return:

```
function js_alert($msg) {
  // Tạo một thông báo JavaScript sử dụng $msg
  echo (
    "\n<SCRIPT LANGUAGE='JavaScript'>\n" .
    "  <!-- \n" .
    "  alert (\"$msg\");\n" .
    "  // --> \n" .
    "</SCRIPT>\n"
  );
}
// Gọi js_alert():
js_alert ("Password bạn nhập không hợp lệ.");
```

Chúng ta cũng có thể sử dụng return để tạm dừng sự thi hành của một hàm, thậm chí nếu không có giá trị được trả về. Dưới đây, hàm js_alert () đã được sửa đổi để tạm dừng nếu $msg chứa một chuỗi rỗng:

```
function js_alert($msg) {
  // Create a JavaScript alert using $msg
```

```
  if ($msg == "") return;        // Halt execution
  echo (
    "\n<SCRIPT LANGUAGE='JavaScript'>\n" .
    "  <!-- \n" .
    "  alert (\"$msg\");\n" .
    "  // --> \n" .
    "</SCRIPT>\n"
  );
}
// Invoke js_alert():
js_alert("The password that you entered is not valid.");
js_alert (""); //This one won't generate the JavaScript code
```

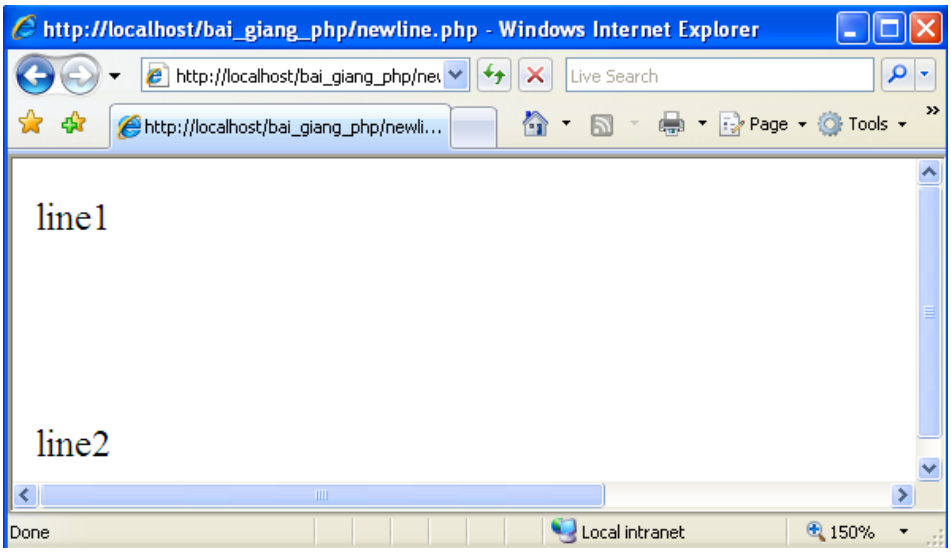
2.2 Truyền đối số

Đối số cung cấp một cách để truyền dữ liệu đầu vào cho hàm. Khi chúng ta viết mã cube(6), "6" là đối số. Đối số có sẵn trong hàm như tham số $num. Tham số này lấy về giá trị của tham số tương ứng được truyền cho nó trong lúc gọi hàm. Hàm cube () sau đó sử dụng giá trị này để tính giá trị trả lại.

Ví dụ newline() sau đây nhận một đối số, nó được sử dụng để quyết định xem có bao nhiêu dòng được in ra. Trong hàm, tham số $x mô tả khoảng cách dòng khi có dòng mới (trong trường hợp này là 5).

```
function newline($x) {
   // Prints <BR> $x times
   for ($i = 0; $i < $x; ++$i) {
      echo ("<BR>\n");
   }
}
```

```
echo ("line1");
newline(5);
echo ("line2");
```



Mặc định, đối dố được truyền bằng giá trị. Điều này có nghĩa là các biến tham số trong phạm vi hàm giữ một bản sao của giá trị truyền cho nó. Nếu thay đổi giá trị của tham số, nó sẽ không thay đổi giá trị của một biến trong câu lệnh gọi. Xem ví dụ sau đây:

```
function print_double($n) {
   $n = $n * 2;
   echo ($n);
}
$a = 5;
echo ("$a <BR>\n");          // In ra 5
print_double($a);            // In ra 10
echo ($a);                   // In ra  5
```

Giá trị của $a không thay đổi, ngay cả sau khi truyền tham số cho print_double (). Ngược lại, khi một đối số được truyền bởi tham chiếu, các thay đổi vào biến tham số nào dẫn đến sự thay đổi kết quả khi gọi

biến của câu lệnh. Một ký hiệu (&) được đặt trước tên của tham số để cho biết rằng đối số sẽ được truyền bởi tham biến:

```
function raise(&$salary, $percent) {
   // Increases $salary by $percent
   $salary += $salary * $percent/100;
}
$sal = 50000;
echo ("Pre-raise salary:  $sal<BR>\n");// In ra 50000
raise ($sal, 4);
echo ("Post-raise salary:  $sal<BR>\n");//In ra 52000
```

Chúng ta cũng có thể thiết lập một giá trị mặc định cho tham số, điều này được thực hiện bằng cách gán một giá trị cho tham số trong khai báo hàm:

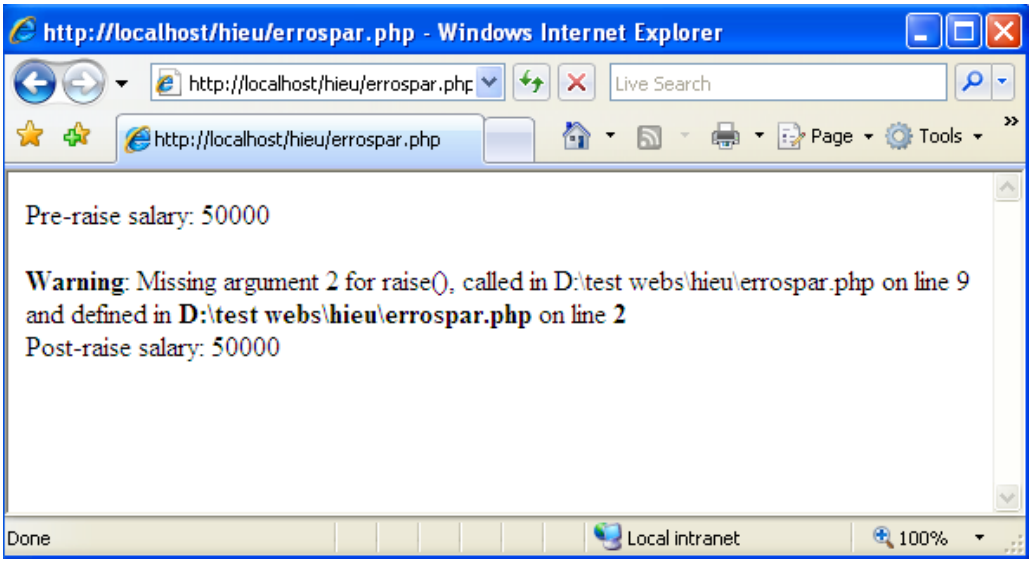
```
function newline($x = 1) {
   // Prints <BR> $x times
   for ($i = 0; $i < $x; ++$i) {
      echo ("<BR>\n");
   }
}
echo ("line1");
newline(); // Prints <BR> once
echo ("line2");
newline(2); // Prints <BR> twice
```

Trong khai báo hàm, quan trọng nhất là vị trí của các tham số có giá trị mặc định phải bắt đầu từ bên phải. Đoạn mã sau sẽ tạo ra lỗi:

```
function raise ($percent = 4, &$salary) {
   // Increases $salary by $percent
   $salary += $salary * $percent/100;
}
$sal = 50000;
```

```
echo ("Pre-raise salary:  $sal<BR>\n");
raise($sal);
echo ("Post-raise salary:  $sal<BR>\n");
```

Đoạn mã trên sẽ cho ra kết quả cùng với thông báo sau:



PHP biên dịch `$sal` là tham số đầu tiên hoặc `$percent`. Rõ rang điều này là sai. Việc đặt bất kỳ các đối số tùy chọn ở cuối sẽ tránh những nhầm lẫn này:

```
function raise(&$salary, $percent = 4) {
   // Tăng $salary bởi $percent
   $salary += $salary * $percent/100;
}

$sal = 50000;
echo ("Pre-raise salary:  $sal<BR>\n");
raise($sal);                  // Assume a 4% raise
echo ("Post-raise salary:  $sal<BR>\n");
```

2.3 Phạm vi và vòng đời biến

Với việc đưa các hàm vào chương trình của chúng ta, chúng ta gặp phải vấn đề về phạm vi biến cho lần đầu tiên. Phạm vi của một

biến xác định các phần của chương trình có thể truy cập vào nó. Hãy xem xét ví dụ sau:

```
$position = "m";
function change_pos() {
     $position = "b";
}
change_pos();
echo ("$position");          // In ra "m"
```

Đoạn mã trên in ra "m". Biến $position trong hàm là biến cục bộ (local) và đó chính là biến khác so với biến $position xuất hiện bên ngoài của hàm. Biến bên ngoài của hàm có phạm vi toàn cục (global), bởi vì nó có thể được truy cập và thay đổi bởi bất kỳ mã trong kịch bản chung mà không phải là trong một hàm. Để truy cập vào một biến toàn cục từ bên trong một hàm, sử dụng câu lệnh global. Câu lệnh global cho PHP biết rằng chúng ta không tạo ra một biến mới (biến $position cục bộ) nhưng thay vì sử dụng biến $position để được tham chiếu đến các nơi khác trong trang.

```
$position = "m";
function change_pos() {
     global $position;
     $position = "b";
}
change_pos();
echo ("$position");          // In ra "b"
```

Bây giờ, đoạn mã trên của chúng ta sẽ in ra "b" bởi vì $position tham chiếu cùng biến bên trong và bên ngoài của hàm.

Ngoài ra, chúng ta có thể sử dụng bằng cách xây dựng mảng $GLOBAL . Mảng này giữ tất cả các biến toàn cục của kịch bản.

```
    $position = "m";
    function change_pos() {
        $GLOBALS["position"] = "b";
    }
    change_pos();
    echo ("$position");          // In ra "b"
```

Lưu ý rằng ký tự $ không xuất hiện trước từ position khi sử dụng mảng này. Ngoài phạm vi biến, chúng ta cần phải xem xét vòng đời của một biến. Có thể có lần khi chúng ta muốn biến cục bộ của hàm giữ lại giá trị của nó từ một lời gọi của các hàm khác. Trong đoạn mã sau, biến cục bộ được tạo và thiết lập là 0 mỗi lần hàm được gọi:

```
function counter () {
   $counter = 0;
   ++$counter;
}
```

Điều này sẽ là không hữu ích trong ngữ cảnh này. Chúng ta cần tạo ra biến $counter tĩnh. **Biến tĩnh** giữ lại giá trị trước mỗi lần một hàm được gọi:

```
function counter () {
   static $counter = 0;
   ++$counter;
}
```

Khi chúng ta khai báo biến sử dụng static, chúng ta cho PHP biết rằng chúng ta muốn giá trị của nó được giữ lại giữa các ngữ cảnh của hàm. Lần đầu tiên hàm được gọi, biến tĩnh sẽ được tạo và được giá trị

được xác định, trong trường hợp này là 0, nhưng sau đó nó sẽ giữ lại giá trị của nó giữa các lần gọi hàm. Nó sẽ không tái khởi tạo biến. Tuy nhiên, giá trị này sẽ chỉ được ghi nhớ trong quá trình thực thi kịch bản. Nếu người sử dụng tải lại các trang web, ví dụ, qua đó tái thực hiện các kịch bản PHP, biến một lần nữa sẽ tái khởi động lần đầu tiên mà hàm được gọi để thực thi kịch bản.

2.4 Gán hàm tới biến

PHP cho phép các biến tham chiếu tới các hàm. Điều này có thể hữu ích khi các điều điều kiện động sẽ quyết định hàm nào nên được gọi trong hoàn cảnh cụ thể. Khi một biến tham chiếu tới một hàm, hàm có thể được gọi bởi các thành phần trong dấu ngoặc đơn chứa các đối số (nếu có) sau tên biến.

Trong ví dụ sau, chúng ta không biết trong phần tùy chọn, hàm nào nên được gọi để nạp trang, do vậy chúng ta kiểm tra giá trị của biến khác `($browser_type)` và gán hàm thích hợp tới `$loading_function`.

```
switch ($browser_type) {
  case "NN":                // Netscape Navigator
    $loading_function = "load_nn";
    break;
  case "IE":                // Internet Explorer
    $loading_function = "load_ie";
    break;
  default:                  // Các trình duyệt khác
    $loading_function = "load_generic";
}
//Bây giờ gọi hàm nạp thích hợp:
$loading_function($URL);
```

3. Mảng

Trong chương 1 chúng ta đã được tìm hiểu về luật của các biến trong PHP. Khi chúng ta đã xem, chúng ta thấy rằng một biến có thể chứa một giá trị đơn. Mảng chứa một số các giá trị. Một mảng gồm có một số các phần tử, mỗi phần tử có một giá trị - dữ liệu được lưu trữ trong phần tử mảng – và một khóa (key) hoặc chỉ mục (index) để các phần tử có thể tham chiếu tới nó. Thông thường một chỉ mục (index) sẽ là một số nguyên (integer). Mặc định, phần tử đùu tiên của mảng có chỉ mục là 0. Tuy nhiên, khi chúng ta sẽ thấy trong các phần sau, một chỉ mục có thể cũng là một chuỗi ký tự (string).

3.1 Các mảng đơn giản

Dạng đơn giản nhất của mảng bao gồm một dãy các phần tử với các chỉ mục bắt đầu là 0 và tăng liên tục. Ví dụ, nếu chúng ta có mảng tên là $contries, mỗi phần tử của mảng chứa 2 ký tự mã đất nước, nó có thể trông giống thế này:

$countries[0]	$countries[1]	$countries[2]	$countries[3]	$countries[4]
"ca"	"cr"	"de"	"uk"	"us"

Chú ý rằng, không giống với mảng trong C, mảng PHP có thể bao gồm các phần tử có số các kiểu dữ liệu khác nhau. Trong C, tất cả các phần tử của mảng phải có kiểu dữ liệu khác nhau, nhưng trong PHP có sự linh hoạt hơn. Mỗi phần tử có thể thuộc kiểu dữ liệu bất kỳ, không cần quan tâm đến các kiểu dữ liệu của các phần tử khác trong mảng.

3.2 Khởi tạo giá trị ban đầu cho mảng

Có một số cách để khởi tạo giá trị cho một mảng. Cách đơn giản nhất là gán các giá trị tới biến mảng. Mỗi lần thực hiện điều này, phần tử khác được thêm vào mảng:

```
$countries[] = "cr";
$countries[] = "de";
$countries[] = "us";
```

Đoạn mã trên tạo ra một mảng gồm 3 phần tử. Khi chúng ta không xác định rõ ràng các chỉ mục trong cặp ngoặc vuông, các phần tử sẽ lấy các chỉ mục chuẩn là 0, 1 và 2. Đoạn mã trên cũng có thể được viết với các chỉ mục rõ ràng:

```
$countries[0] = "cr";
$countries[1] = "de";
$countries[2] = "us";
```

Mảng thường được lập trình để gán các chỉ mục trong dãy sắp xếp khi chúng ta thực hiện ở trên. Nhưng nếu cần thiết, chúng ta có thể gán chỉ mục là số nguyên bất kỳ:

```
$countries[50] = "cr";
$countries[20] = "de";
$countries[10] = "us";

echo ("$countries[20]"); // In ra de
```

Mảng mới này cũng chứa 3 phần tử nhưng với các chỉ mục là 10, 20 và 50.

Nếu chúng ta cần biết có bao nhiêu phần tử trong mảng, chúng ta có thể sử dụng hàm `count()`. Hàm này sẽ trả về kết quả là một số nguyên số các phần tử trong mảng. Trong ví dụ trên, `count($countries)` sẽ trả về 3.

Một cách khác để tạo giá trị ban đầu cho một mảng là dùng cấu trúc `array()`. Chúng ta đưa vào `array()` các giá trị mà chúng ta muốn gán vào mảng mới của chúng ta:

```
$countries = array ("cr", "de", "us");
echo ("$countries[2]");     // In ra "us"
```

Nếu chúng ta muốn ghi đè các chỉ mục mặc định, toán tử cho phép chúng ta gán các chỉ mục xác định tới các phần tử của chúng ta. Trong ví dụ trên, $countries có 3 phần tử với các chỉ mục 0, 1 và 2. Nếu chúng ta muốn chỉ mục của mảng bắt đầu là 1, chúng ta có thể viết:

```
$countries = array (1 => "cr", "de", "us");
echo ("$countries[2]"); // Prints de
```

Toán tử có thể được sử dụng trước bất kỳ phần tử nào của mảng. Trong ví dụ sau, "cr" sẽ có chỉ mục là 0, "de" sẽ có chỉ mục là 7 và "us" sẽ có chỉ mục là 8:

```
$countries = array ("cr", 7 => "de", "us");
```

3.3 Lặp mảng

Một trong những tính năng hữu ích nhất của mảng là nó có thể lặp và thực hiện quá trình lặp đi lặp lại trên các phần tử riêng lẻ. Có một số cách để thực hiện điều này, phụ thuộc vào mảng được chỉ mục tuần tự hoặc các chỉ mục của mảng là không dự đoán được.

Các mảng được chỉ mục tuần tự

Cách đơn giản nhất để lặp mảng là sử dụng count() để quyết định số các phần tử trong một mảng và sau đó sử dụng vòng lặp for():

```
$countries = array ("cr", "de", "us");
$num_elements = count ($countries);
// $num_elements có giá trị là 3

for ($idx = 0; $idx < $num_elements; ++$idx) {
   // In mỗi phần tử trên 1 dòng:
```

```
   echo ("$countries[$idx] <BR>\n");
}
```

Chúng ta khởi tạo giá trị ban đầu `$idx` với một giá trị 0 bởi vì phần tử đầu tiên của mảng của chúng ta có một chỉ mục 0. Thông thường, nếu chỉ mục thấp nhất là một số khác lớn hơn 0, chúng ta có thể gán gián giá trị ban đầu `$idx` với giá trị đó. Sau đó chúng ta tăng $idx lên 1 với mỗi lần lặp của vòng lặp. lần lặp cuối cùng xảy ra khi `$idx` là số bé hơn gần nhất với tổng số phần tử. (Trong mảng có 3 phần tử bắt đầu bởi chỉ mục là 0, phần tử cuối cùng có chỉ mục là 2.)

Chúng ta có thể không cần biến `$num_elements` trong ví dụ trên bằng cách gán luôn hàm `count()` trong vòng lặp `for`.

```
$countries = array ("cr", "de", "us");
for ($idx = 0; $idx < count ($countries); ++$idx) {
   // In mỗi phần tử trên một dòng mới:
   echo ("$countries[$idx] <BR>\n");
}
```

Có hai lý do để lờ điều này. Một là nó có rất ít hiệu quả khi count() được gọi mọi lần trong suốt vòng lặp.

Các mảng được chỉ mục không tuần tự

Một mảng đã được xây dựng trong một con trỏ . Trong mảng này con trỏ giữ bản ghi của mỗi phần tử hiện là rõ ràng. Đối với một mảng vừa được tạo ra, con trỏ là phần tử đầu tiên. Chúng ta có thể quyết định giá trị của phần tử hiện tại bằng cách sử dụng hàm `current()` và các chỉ mục của phần tử hiện thời sử dụng hàm `key()`. Để minh họa điều này, các mã lệnh sau khởi tạo một mảng và in ra giá trị và chỉ số của phần tử hiện tại:

```
    $countries[50] = "cr";
    $countries[20] = "de";
    $countries[10] = "us";
```

```
    $countries[] = "uk";   // Có chỉ mục '51'
    $key = key ($countries);
    $value = current ($countries);
    echo ("Element $key equals $value");
```

Bởi vì mảng đó vừa được tạo ra (phần tử hiện thời là đầu tiên) do vậy mã này sẽ in ra `Element 50 equals cr`. Lưu ý rằng `“cr"` là phần tử đầu tiên trong mảng này (mặc dù `"de"` và `"us"` có chỉ mục thấp hơn) bởi vì nó là phần tử đầu tiên được gán cho mảng. Mảng không được sắp xếp trừ khi chúng ta gọi một cách cụ thể một hàm để sắp xếp mảng, chúng ta sẽ thấy làm thế nào để làm việc này trong phần sau.

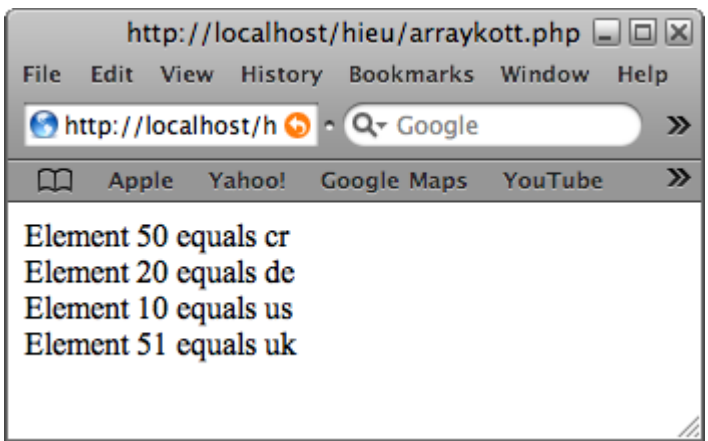
Hai hàm `each()` và `list()` có thể được sử dụng với nhau để lặp một mảng, dù các chỉ số là không tuần tự.

```
reset ($countries);
while (list ($key, $value) = each ($countries)) {
   echo "Element $key equals $value<BR>\n";
}
```

Trong phần “Các mảng chỉ mục chuỗi”, chúng ta sẽ nghiên cứu kỹ lưỡng hơn về nó làm việc như thế nào. Bây giờ, nghĩ đến dòng sau:

```
while (list ($key, $value) = each ($countries)) {
```

Nghĩa là “mỗi phần tử trong mảng, thiết lập `$key` bằng khóa của phần tử (hoặc chỉ mục) và `$value` bằng giá trị của phần tử.” (Một số ngôn ngữ thực thi điều này với một cấu trúc `for each`.) Hàm `reset()` thiết lập con trỏ nội bộ tới phần tử đầu tiên. Thông thường, điều này không cần thiết nếu con trỏ đã sẵn sàng ở phần tử đầu tiên, nhưng nó có thể là



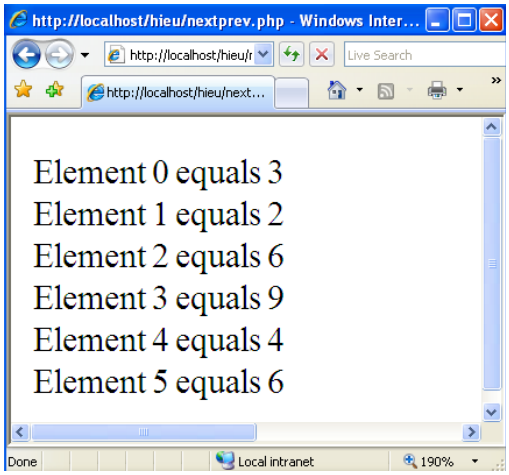
thói quen hữu ích để đưa `reset()` trước lúc duyệt mảng để tạo ra một sự tin tưởng chúng ta khởi động lúc bắt đầu. Hàm `each()` di chuyển một phần tử con trỏ mảng quay lại mỗi lần nó được gọi. Do vậy, nếu vòng lặp của chúng ta được lồng, `reset()` sẽ cần được lưu trữ lại con trỏ tới phần tử đầu tiên. Sử dụng mảng `$countries` được gán trước đây, mã lệnh phía trên sẽ được hiển thị là:

Một vài lệnh về next() và prev()

PHP cung cấp hai hàm khác để định hướng mảng. Hàm `next()` nhận một mảng là đối số của nó. Nó di chuyển một phần tử con trỏ nội bộ của mảng này sang phải và trả về giá trị của phần tử mới (hoặc một giá trị sai nếu nó đã đọc tới phần tử cuối cùng). Cũng như vậy, chúng ta mong muốn `prev()` cũng làm được giống thế này nhưng theo hướng ngược lại. Xem xét đoạn mã sau:

```
// Khai báo một mảng:
$arr = array (3, 2, 6, 9, 4, 6);
// lặp:
do {
 $k = key ($arr);
 $val = current ($arr);
echo ("Element $k equals $val<BR>\n");
} while (next($arr));
```

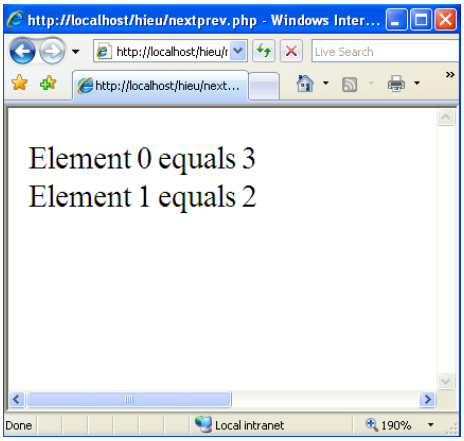
Đoạn mã này sẽ được hiển thị như sau:



Nhớ rằng, `next()` trả về giá trị của phần tử tiếp theo. Nếu giá trị này được đánh giá là `false`, vòng lặp sẽ dừng thực thi. Xem xét những cái xảy ra nếu chúng ta thay đổi phần tử thứ 3 của mảng chúng ta thành 0:

```
// Khai báo mảng:
$arr = array (3, 2, 0, 9, 4, 6);

// lặp:
do {
 $k = key ($arr);
 $val = current ($arr);
echo ("Element $k equals $val<BR>\n");
} while (next ($arr));
```

Đoạn mã này sẽ hiển thị:



Đây có thể là một vấn đề khó khăn để gỡ lỗi, vì nó đòi hỏi kiến thức về các dữ liệu liên quan (các nội dung của mảng). Giải pháp của chúng ta là sử dụng `next()` giống như thế này:

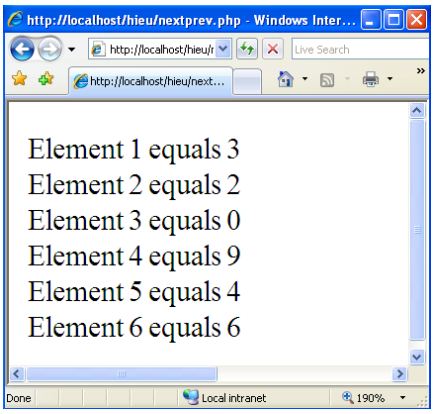
```
$arr = array (3, 2, 0, 9, 4, 6);
// Sử dụng vòng lặp khác:
for (reset ($arr); $k = key ($arr); next ($arr)) {
  $val = current ($arr);
  echo ("Element $k equals $val<BR>\n");
}
```

Đoạn mã trên chúng ta sử dụng `next()` chỉ để di chuyển con trỏ, chúng ta loại bỏ giá trị trả về của nó. Điều này sử lý vấn đề trước đó. Nhưng nó nảy sinh một vấn đề mới: Vòng lặp này sẽ không bao giờ thực thi! Bây giờ nó là `key()` mà đang trả về một vòng lặp với giá trị sai không rõ ràng. Do `$arr` là mảng có chỉ số đầu là 0, lần gán đầu tiên trả về một giá trị là 0. Do vậy, đánh giá là sai và dừng thực thi vòng lặp trước khi nó bắt đầu. Vấn đề này sẽ xảy ra cho tất tất cả các mảng có chỉ số đầu là 0 hoặc bất kỳ mảng nào có chỉ mục được định giá là 0 (Chẳng hạn một mảng được chỉ mục là chuỗi ký tự với một khóa chuỗi ký tự rỗng). Chúng ta có thể sử lý vấn đề mới này bằng cách tránh một mảng có chỉ mục là 0:

```
$arr = array (1 => 3, 2, 0, 9, 4, 6);
// lặp:
```

```
for (reset ($arr); $k = key ($arr); next ($arr)) {
  $val = current ($arr);
  echo ("Element $k equals $val<BR>\n");
}
```

Sẽ in ra:



Array_walk()

Trong một số tình huống, hàm `array_walk()` có thể cung cấp một sự lựa chọn giữa hai hay nhiều khả năng để xây dựng một vòng lặp duyệt mảng. Hàm này cho phép chúng ta áp dụng một hàm mà chúng ta đã viết tới mọi thành phần của một mảng:

```
    function println ($s) {
        echo "$s<BR>\n";
    }

    $countries = array ("ca", "cr", "de", "us");
    array_walk ($countries, println);
```

Đối số đầu tiên được lấy bởi `array_walk()` là mảng và đối số thứ hai là tên của hàm được áp dụng. Mã lệnh phía trên sẽ in mỗi phần tử của mảng trên mỗi dòng.

3.4 Mảng được đánh chỉ mục là chuỗi ký tự

Tất cả các mảng mà chúng ta đã khảo sát đều có các chỉ mục là số nguyên. Tuy nhiên, do chúng ta đã đề cập lúc bắt đầu của phần

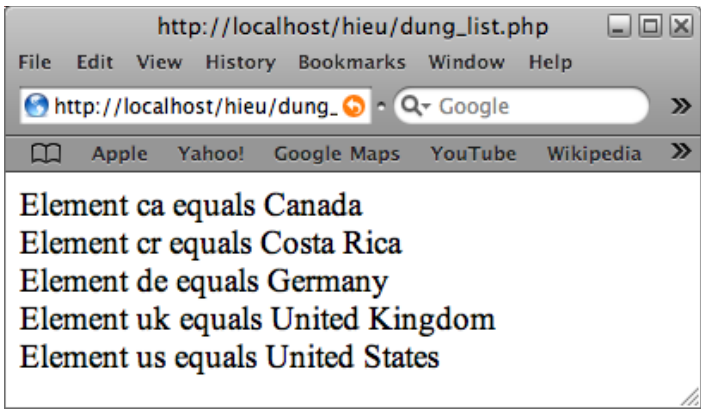
Mảng, mảng cũng có thể sử dụng các chuỗi ký tự là các chỉ mục của chúng:

```
    $countries["ca"] = "Canada";
    $countries["cr"] = "Costa Rica";
    $countries["de"] = "Germany";
    $countries["uk"] = "United Kingdom";
    $countries["us"] = "United States";

    // In ra 'Germany':
    echo ("$countries[de]");
```

Mảng giống nhau có thể được gán với `array()` và toán tử (trong các ví dụ trên). Trong ví dụ này, chúng ta lại sử dụng `list()` và `each()` để duyệt mảng:

```
$countries = array ("ca" => "Canada",
                    "cr" => "Costa Rica",
                    "de" => "Germany",
                    "uk" => "United Kingdom",
                    "us" => "United States");

while (list ($key, $val) = each ($countries)) {
   echo "Element $key equals $val<BR>\n";
}
```

Sẽ in ra:



Bây giờ chúng ta xem xét thêm về each() làm việc như thế nào. Hư chúng ta đã biết, hàm each() nhận một mảng là đối số của nó. Trong trường hợp này, chúng ta đang truyền cho nó (mảng $contries). Nó trả về các giá trị này trong dạng của mảng 4 phần tử với các chỉ số 0, 1, "key" và "value". Các phần tử 0 và "key" bao gồm cả chỉ mục của phần tử hiện thời của $countries. Phần tử 1 và "value" bao gồm cả chỉ mục của phần tử hiện thời của $countries. Do vậy trong ví dụ này:

```
$countries = array ("ca" => "Canada",
                    "cr" => "Costa Rica",
                    "de" => "Germany",
                    "uk" => "United Kingdom",
                    "us" => "United States");
$arr = each ($countries);
```

Bây giờ $arr sẽ là mảng 4 phần tử với các khóa và các giá trị sau:

1. Phần tử đầu tiên có chỉ mục 0 và giá trị ca
2. Phần tử thứ hai có chỉ mục 1 và giá trị Canada
3. Phần tử thứ ba có chỉ mục "key" và giá trị ca
4. Phần tử thứ tư có chỉ mục "value" và giá trị Canada

Trên thực tế list() không phải là một hàm nhưng PHP là ngôn ngữ cấu trúc. Nó được dùng để gán giá trị phần tử của một mảng tới các biến được chỉ định. Thay vì gán, mảng được trả về bởi each() tới biến $arr. Chúng ta có thể sử dụng list() để giữ lại các giá trị này trong biến:

```
// Quay lại phần tử đầu tiên của $countries
reset ($countries);
list ($key, $val) = each ($countries);
echo ("$key<BR>\n");   // Prints 'ca'
```

```
echo ("$val<BR>\n");   // In ra 'Canada'
```

3.5 Mảng nhiều chiều

Lúc bắt đầu phần mảng, chúng ta đã định nghĩa một mảng là chứa nhiều giá trị. Không có lý do là tại sao các giá trị này không thể là các mảng của chúng. Điều này là kết quả của mảng hai chiều. Về cơ bản, chúng ta có thể tạo một mảng mà các phần tử của nó chứa mảng. Nếu các phần tử mảng được lồng cũng chứa mảng, chúng ta kết thúc với mảng ba chiều, và do vậy.

Giả định chúng ta có một mảng được chỉ mục là chuỗi ký tự được gọi là $continents. Chúng ta có thể lồng array() do vậy mỗi phần tử của mảng chứa một mảng các đất nước:

```
$continents = array ("Europe" => array ("de", "uk"),
            "North America" => array ("ca", "cr", "us"));

echo ($continents["Europe"][1]);        // In ra "uk"
echo ($continents["North America"][2]); // In ra "us"
```

Đoạn mã trên tạo r một mảng hai chiều với cấu trúc như sau:

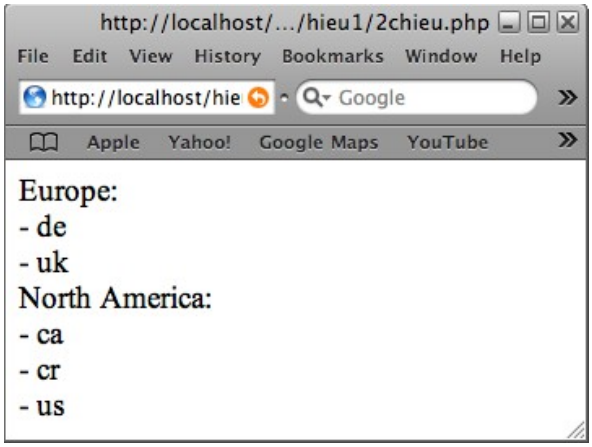
$continents["Europe"]		$continents["America"]		
[0]	[1]	[0]	[1]	[2]
"de"	"uk"	"ca"	"cr"	"us"

Tất nhiên, chúng ta cũng có thể sử dụng một vòng lặp lồng nhau để duyệt các mảng lồng nhau:

```
$continents = array ("Europe" => array ("de", "uk"),
            "North America" => array ("ca", "cr", "us"));
while (list ($key1) = each ($continents)) {
    echo ("$key1:<BR>\n");      // In tên châu lục:
   // Danh sách các nước cho châu lục đó:
```

```
            while    (list    ($key2,    $val)    =    each
($continents["$key1"])) {
      echo ("- $val<BR>\n");
   }
}
```

Và đây là kết quả:

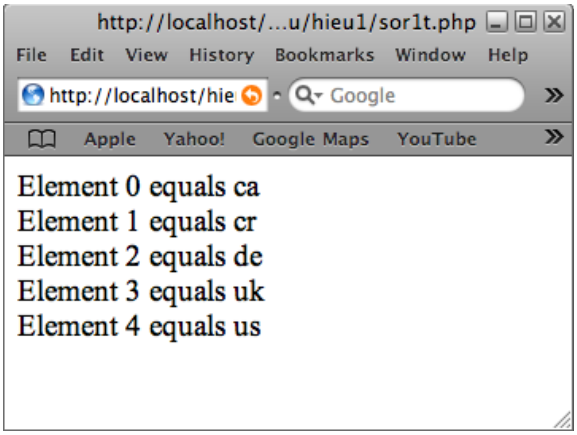


3.6 Các hàm sắp xếp

PHP cung cấp các hàm để sắp xếp các mảng. Hàm đơn giản nhất là sort(). Hàm này tái sắp xếp các phần tử theo thứ tự số và thứ tự chữ cái. (Các số đầu tiên, sau đó là dấu chấm câu, sau cùng là các chữ cái) Nó gán lại các chỉ số của mảng để tạo ra một thứ tự sắp xếp mới.

```
$countries = array ("us", "uk", "ca", "cr", "de");
sort ($countries);

while (list ($key, $val) = each ($countries)) {
   echo "Element $key equals $val<BR>\n";
}
```

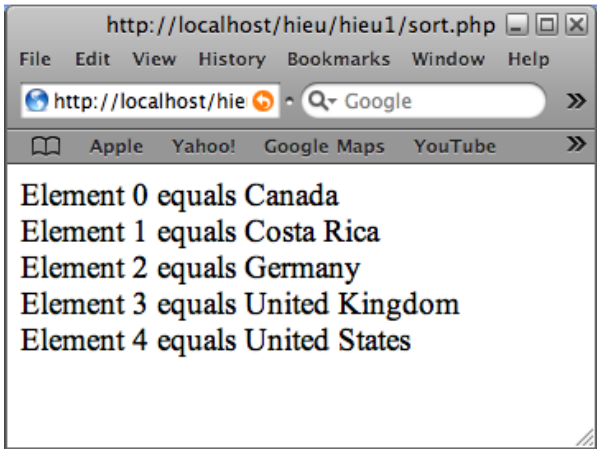
Mã trên tái sắp xếp lại mảng sao cho các giá trị trong thứ tự chữ cái và các chỉ mục này cũng được sắp xếp theo thứ tự này:



Chú ý: Khi ta sử dụng hàm sắp xếp này, các chỉ mục được gán dứt khoát:

```
$countries = array ("us" => "United States",
                    "uk" => "United Kingdom",
                    "ca" => "Canada",
                    "cr" => "Costa Rica",
                    "de" => "Germany");

sort ($countries);

while (list ($key, $val) = each ($countries)) {
   echo "Element $key equals $val<BR>\n";
}
```

Khi chúng ta chạy mã này, nó sẽ in ra:

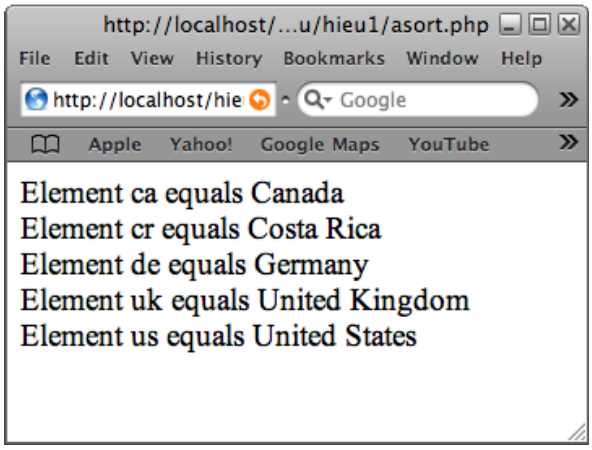


Các chỉ mục chuỗi ký tự của chúng ta được thay thế bởi các chỉ mục số! Biểu thức `$countries["ca"]` bây giờ sẽ dẫn đến kết

quả tương đương. Vấn đề này được thêm bởi hàm `asort()`, hàm này thay đổi thứ tự của các phần tử mà không thay đổi các chỉ mục:

```
$countries = array ("us" => "United States",
                    "uk" => "United Kingdom",
                    "ca" => "Canada",
                    "cr" => "Costa Rica",
                    "de" => "Germany");

asort ($countries); // Preserve keys

while (list ($key, $val) = each ($countries)) {
   echo "Element $key equals $val<BR>\n";
}
```

Sẽ in ra:

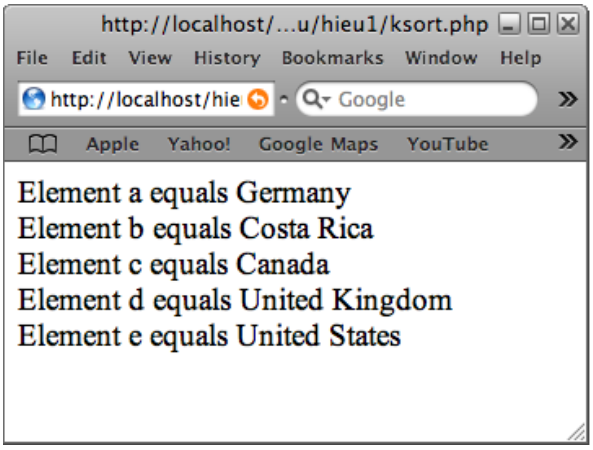


Hàm rsort() và arsort() lần lượt giống với sort() và asort() trừ việc chúng sắp xếp các mảng trong thứ tự duyệt. hàm ksort() sắp xếp các mảng bởi khóa:

```
$countries = array ("e" => "United States" ,
                    "d" => "United Kingdom",
```

```
                    "c" => "Canada",
                    "b" => "Costa Rica",
                    "a" => "Germany");

ksort ($countries);

while (list ($key, $val) = each ($countries)) {
   echo "Element $key equals $val<BR>\n";
}
```

Sẽ in ra:



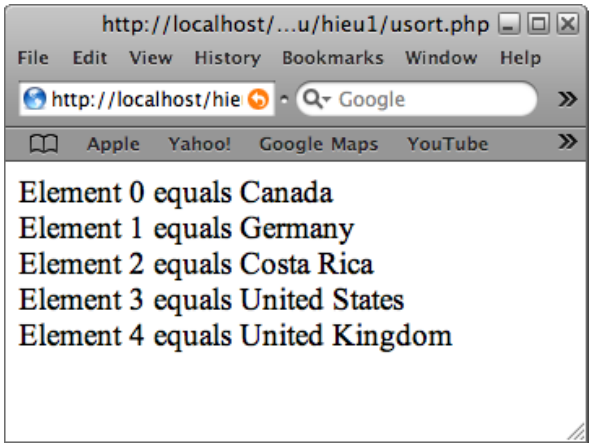
Không có một hàm nào sắp xếp thứ tự duyệt bởi khóa của mảng. Để làm được điều này chỉ có thể sử dụng `asort()` sau đó dùng `ksort()`.

Hàm `usort()` là khá phức tạp. Nó nhận một mảng là một đối số (giống như tất cả các hàm sắp xếp khác) nhưng nó cho phép nhận một đối số thứ hai. Đối số thứ hai này là một hàm mà chúng ta có thể định nghĩa để cho `usort()` biết thực hiện quá trình sắp xếp như thế nào. Ví dụ sau sắp xếp một mảng theo độ dài của các chuỗi ký tự đã bao gồm các phần tử trong nó. Hàm `strlen()` trả về độ dài của một chuỗi ký tự.

```
function by_length ($a, $b) {
```

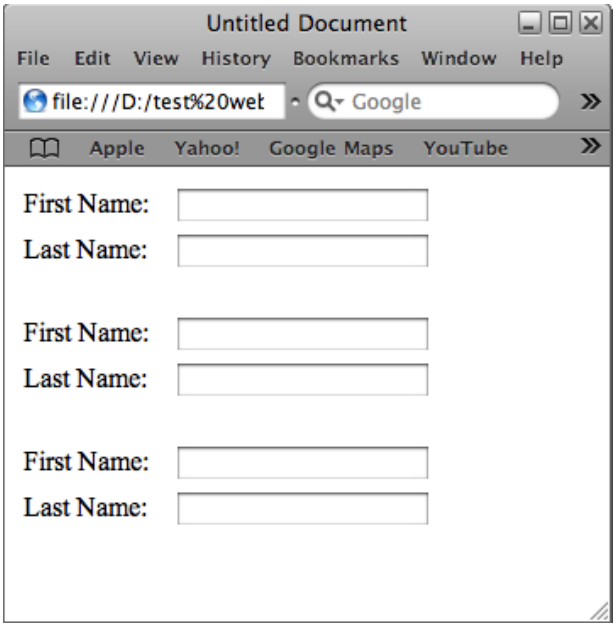
```
   $l_a = strlen ($a);
   $l_b = strlen ($b);
   if ($l_a == $l_b) return 0;
   return ($l_a < $l_b) ? -1 : 1;
}

$countries = array ("e" => "United States" ,
                    "d" => "United Kingdom",
                    "c" => "Canada",
                    "b" => "Costa Rica",
                    "a" => "Germany");
usort ($countries, by_length);

while (list ($key, $val) = each ($countries)) {
   echo "Element $key equals $val<BR>\n";
}
```

Đoạn mã trên sẽ in ra tên của 5 đất nước đã được sắp xếp theo độ dài của chúng:



3.7 Sử dụng mảng với các phần tử Form

Mảng là đặc biệt hữu ích khi chia bảng (giống như dữ liệu Form trong HTML). Giả sử chúng ta muốn cho phép một người dùng nhập tên vào cơ sở dữ liệu. Chúng ta có thể tạo một Form giống như thế này:



Mã HTML cho các ô textbox có thể giống như thế này:

```
<INPUT NAME="first1" TYPE=TEXT>
<INPUT NAME="last1" TYPE=TEXT>

<INPUT NAME="first2" TYPE=TEXT>
<INPUT NAME="last2" TYPE=TEXT>

<INPUT NAME="first3" TYPE=TEXT>
<INPUT NAME="last3" TYPE=TEXT>
```

Mỗi ô textbox có một tên định danh (“first1”, “last1”, “first2”, vv). Khi Form được submit, nó sẽ trả về một biến PHP cho mỗi ô text-box trên Form ($first1, $last1, $first2, vv.). Tuy nhiên, nếu số hàng có thể thay đổi (rất có khả năng sẽ như thế khi đưa vào cơ sở dữ liệu), nó hữu ích hơn để mô tả

mỗi trường là một mảng dữ liệu. Trong HTML, điều này được thực hiện bằng cách đặt các dấu ngoặc vuông sau tên của phần tử. Chúng ta có thể sử dụng vòng lặp `for` để đặt số điều kiện cần thiết của các ô text-box trên Form:

```
<?php
   $names = 3;
?>
<FORM ACTION="submit.php" METHOD=POST>
   <?php for ($i = 1; $i <= $names; $i++): ?>
   First name:
   <INPUT NAME="first[]" TYPE=TEXT><BR>
   Last name:
   <INPUT NAME="last[]" TYPE=TEXT><BR><BR>
   <?php endfor ?>
   <BR><INPUT TYPE=SUBMIT>
</FORM>
```

Khi Form được submit, PHP sẽ tạo ra một mảng được gọi là `$first` và `$last` với mỗi phần tử bao gồm giá trị của một ô `text-box`. Điều này mang lại cho chúng ta lập trình để xử lý việc `submit` dữ liệu đơn giản trong kịch bản PHP của chúng ta. Giả sử rằng chúng ta muốn thêm các tên được `submit` tới một bảng cơ sở dữ liệu. Chúng ta có thể xây dựng một câu lệnh `INSERT` trong SQL cho mỗi cặp tên một cách dễ dàng bằng cách lặp mảng:

```
<!-- submit.php -->
<?php
   $numrows = count ($first);
   for ($i = 0; $i < $numrows; ++$i) {
      $sql = "INSERT INTO Names ('First', 'Last') " .
             "VALUES ('$first[$i]', '$last[$i]')";
```

```
      // Code to execute query goes here.
      // ...
      // ...
   }
?>
```

4.Tổng kết

Các hàm làm cho nó có thể viết môdum, tái sử dụng mã. Hàm có thể nhận các đối số và trả về một giá trị.

Đối số thường được truyền bởi giá trị, nó có nghĩa rằng một bản sao dữ liệu được gửi tới hàm. Các thay đổi tới bản sao của biến không ảnh hưởng đến biến gốc. Các đối số có thể được truyền bởi tham chiếu, trong trường hợp đó hàm không làm việc với bản sao dữ liệu, đúng hơn là biến gốc của nó. Do vậy, các thay đổi tới biến tiếp tục tồn tại bên ngoài hàm. Các đối số có thể được tạo tùy ý bằng việc gán giá trị cho chúng trong khai báo hàm.

Các biến trong hàm thường có phạm vi cục bộ, nghĩa là chúng tồn tại duy nhất trong hàm và sẽ không can thiệp với các biến ngoài hàm cho dù chúng có tên giống nhau. Các hàm có thể truy cập các biến toàn cục bằng cách sử dụng câu lệnh global. Các biến cục bộ trong hàm được khởi tạo lại mỗi lần hàm được gọi trừ khi câu lệnh static được sử dụng. Trong trường hợp này, nó sẽ giữ lại giá trị cũ của nó từ lời gọi trước đó.

Trong chương này chúng ta cũng đã thảo luận về một công cụ lập trình không thể thiếu: Mảng (array). Mảng là một danh sách các giá trị. Mỗi giá trị được lưu trữ trong một phần tử của mảng và chúng ta có thể tham chiếu tới từng phần tử riêng lẻ bằng chỉ mục hoặc khóa của nó. Mặc dù thông thường hầu hết các chỉ mục sẽ là các số nguyên và sẽ là dãy tăng dần, đây không phải là một trường hợpvà chúng có thể là các giá trị số hoặc chuỗi ký tự.

Một tính năng hữu ích đặc biệt của mảng là khả năng lặp mảng. Điều này cho phép chúng ta thực thi thao tác giống nhau trên mỗi phần

tử mảng mà không phải viết mã cho mỗi phần tử. Mặc dù PHP cung cấp một số cách để làm được điều này, nó vẫn là cách tốt nhất để sử dụng `list()` và `each()` trong một vòng lặp hoặc hàm `array_walk()`. Các hàm `next()` và `prev()` thông thường nên được tránh khi duyệt mảng vì chúng thường dẫn đến hỏng vòng lặp.

Các phần tử của mảng có thể chứa các mảng. Đây là kết quả của mảng nhiều chiều. Mảng nhiều chiều là điển hình cho cách duyêth sử dụng các vòng lặp lồng nhau.

PHP cung cấp một số hàm để sắp xếp mảng, cho phép chúng ta sắp xếp bởi nhiều tiêu chí khác nhau. Mảng cũng có hàm `usort()` để cho phép chúng ta chỉ định một hàm so sánh tùy chọn.

Mảng rất hữu ích để xử lý bảng (giống như form dữ liệu HTML). Các phần tử form có thể được gán tới một mảng bằng cách đặt dấu ngoặc vuông sau tên của phần tử.

Câu hỏi trắc nghiệm kết chương

Trả lời các câu hỏi sau:

1. Cách chính xác để tạo ra một hàm trong PHP?
   a. create myFunction()
   b. new_function myFunction()
   c. function myFunction()
2. Nếu chúng ta viết if (7 == $i) thì câu lệnh trên sẽ?
   a. So sánh biến $i với 7
   b. Báo lỗi
3. Để khai báo một biến mảng, chúng ta sử dụng câu lệnh?
   a. $tên_biến = create array()
   b. $tên_biến = array()
   c. $tên_biến = int array[]
   d. $tên_biến = array[]
4. Cách đơn giản nhất để sắp xếp mảng trong PHP là sử dụng hàm?
   a. ksort()
   b. sort()
   c. asort()
   d. rsort()
5. Để duyệt mảng hai chiều chúng ta sử dụng?
   a. Vòng lặp for
   b. Vòng lặp for lồng nhau
   c. Vòng lặp while
   d. Vòng lặp while lồng nhau