เรียนรู้การเขียนเว็บเพจด้วยภาษา PHP ด้วยตนเองภายใน 5.5 สัปดาห์

PHP in 5.5 Week

Credit :
http://www.bcoms.net/php/

Table of Contents

สัปดาห์ที่ 1

บทที่ 1 ประวัติความเป็นมาของภาษา PHP

PHP เป็นภาษาจำพวก scripting language คำสั่งต่างๆจะเก็บอยู่ในไฟล์ที่เรียกว่าสคริปต์ (script) และเวลาใช้งานต้องอาศัยตัวแปลชุดคำสั่ง ตัวอย่างของภาษาสคริปก็เช่น JavaScript, Perl เป็นต้น ลักษณะของ PHP ที่แตกต่างจากภาษาสคริปต์แบบอื่นๆ คือ PHP ได้รับการพัฒนาและออกแบบมา เพื่อใช้งานในการสร้างเอกสารแบบ HTML โดยสามารถ สอดแทรกหรือแก้ไขเนื้อหาได้โดยอัตโนมัติ ดังนั้นจึงกล่าวว่า PHP เป็นภาษาที่เรียกว่า server-side หรือ HTML-embedded scripting language เป็นเครื่องมือที่สำคัญชนิดหนึ่ง ที่ช่วยให้เราสามารถสร้างเอกสารแบบ Dynamic HTML ได้อย่างมีประสิทธิภาพและมีลูกเล่นมากขึ้น

ถ้าใครรู้จัก Server Side Include (SSI) ก็จะสามารถเข้าใจการทำงานของ PHP ได้ไม่ยาก สมมุติว่า เราต้องการจะแสดงวันเวลาปัจจุบันที่ผู้เข้ามาเยี่ยมชมเว็บไซต์ในขณะนั้น ในตำแหน่ง ใดตำแหน่งหนึ่งภายในเอกสาร HTML ที่เราต้องการ อาจจะใช้คำสั่งในรูปแบบนี้ เช่น <!--#exec cgi="date.pl"--> ไว้ในเอกสาร HTML เมื่อ SSI ของ web server มาพบคำสั่งนี้ ก็จะกระทำคำสั่ง date.pl ซึ่งในกรณีนี้ เป็นสคริปต์ที่เขียนด้วยภาษา perl สำหรับอ่านเวลาจากเครื่องคอมพิวเตอร์ แล้วใส่ค่าเวลาเป็นเอาพุท (output) และแทนที่คำสั่งดังกล่าว ลงในเอกสาร HTML โดยอัตโนมัติ ก่อนที่จะส่งไปยังผู้อ่านอีกทีหนึ่ง

อาจจะกล่าวได้ว่า PHP ได้รับการพัฒนาขึ้นมา เพื่อแทนที่ SSI รูปแบบเดิมๆ โดยให้มีความสามารถ และมีส่วนเชื่อมต่อกับเครื่องมือชนิดอื่นมากขึ้น เช่น ติดต่อกับคลังข้อมูลหรือ database เป็นต้น

PHP ได้รับการเผยแพร่เป็นครั้งแรกในปีค.ศ.1994 จากนั้นก็มีการพัฒนาต่อมาตามลำดับ เป็นเวอร์ชั่น 1 ในปี 1995 เวอร์ชั่น 2 (ตอนนั้นใช้ชื่อว่า PHP/FI) ในช่วงระหว่าง 1995-1997 และเวอร์ชั่น 3 ช่วง 1997 ถึง 1999 จนถึงเวอร์ชั่น 4 ในปัจจุบัน

PHP เป็นผลงานที่เติบโตมาจากกลุ่มของนักพัฒนาในเชิงเปิดเผยรหัสต้นฉบับ หรือ OpenSource ดังนั้น PHP จึงมีการพัฒนาไปอย่างรวดเร็ว และแพร่หลายโดยเฉพาะอย่างยิ่งเมื่อใช้ร่วมกับ Apache Webserver ระบบปฏิบัติอย่างเช่น Linux หรือ FreeBSD เป็นต้น ในปัจจุบัน PHP สามารถใช้ร่วมกับ Web Server หลายๆตัวบนระบบปฏิบัติการอย่างเช่น Windows 95/98/NT เป็นต้น

รายชื่อของนักพัฒนาภาษา PHP ที่เป็นแก่นสำคัญในปัจจุบันมีดังต่อไปนี้

- Zeev Suraski, Israel
- Andi Gutmans, Israel
- Shane Caraveo, Florida USA
- Stig Bakken, Norway
- Andrey Zmievski, Nebraska USA
- Sascha Schumann, Dortmund, Germany
- Thies C. Arntzen, Hamburg, Germany
- Jim Winstead, Los Angeles, USA
- Rasmus Lerdorf, North Carolina, USA

เนื่องจากว่า PHP ไม่ได้เป็นส่วนหนึ่งของตัว Web Server ดังนั้นถ้าจะใช้ PHP ก็จะต้องดูก่อนว่า Web server นั้นสามารถใช้สคริปต์ PHP ได้หรือไม่ ยกตัวอย่างเช่น PHP สามารถใช้ได้กับ Apache WebServer และ Personal Web Server (PWP) สำหรับระบบปฏิบัติการ Windows 95/98/NT

ในกรณีของ Apache เราสามารถใช้ PHP ได้สองรูปแบบคือ ในลักษณะของ CGI และ Apache Module ความแตกต่างอยู่ตรงที่ว่า ถ้าใช้ PHP เป็นแบบโมดูล PHP จะเป็นส่วนหนึ่งของ Apache หรือเป็นส่วนขยายในการทำงานนั่นเอง ซึ่งจะทำงานได้เร็วกว่าแบบที่เป็น CGI เพราะว่า ถ้าเป็น CGI แล้ว ตัวแปลชุดคำสั่งของ PHP ถือว่าเป็นแค่โปรแกรมภายนอก ซึ่ง Apache จะต้องเรียกขึ้นมาทำงานทุกครั้ง ที่ต้องการใช้ PHP ดังนั้น ถ้ามองในเรื่องของประสิทธิ ภาพในการทำงาน การใช้ PHP แบบที่เป็นโมดูลหนึ่งของ Apache จะทำงานได้มีประสิทธิภาพมากกว่า

ต่อไปเราจะมาทำความรู้จักกับภาษา PHP และทำความเข้าใจการทำงาน รวมถึงคำสั่งพื้นฐานต่าง ๆ

บทที่ 2 การสอดแทรกคำสั่งภาษา PHP ในเอกสาร HTM

เพื่อเป็นการบ่งบอกให้รู้ว่า ส่วนใดเป็นคำสั่ง PHP ที่อยู่ภายในเอกสาร HTML จึงได้มีการกำหนดสัญลักษณ์ไว้ดังนี้ ซึ่งสามารถทำได้หลายรูปแบบ เช่น

1. <? ... ?> (SGML style)
2. <?php ... ?> (XML style)
3. <script language="php"> ... </script> (JavaScript style)
4. <% ... %> (ASP style)

ที่นิยมก็คือแบบแรก โดยเริ่มต้นด้วย <? และจบด้วย ?> และตรงกลางจะเป็นคำสั่งในภาษา PHP

เราสามารถวางคำสั่ง PHP ไว้ภายในเอกสาร HTML ตามที่ต้องการได้ อาจจะสลับกับ Tag ของภาษา HTML ก็ได้ ตัวอย่างเช่น

```
<HTML>
<HEAD><TITLE> My  Homepage </TITLE></HEAD>
<BODY BGCOLOR=#FFFFFF>
<H1><? echo "Hello World"; ?></H1>
Your web browser is <? echo $HTTP_USER_AGENT; ?>.
</BODY>
</HTML>
```

คำสั่งแรกที่ง่ายที่สุดสำหรับการเรียนรู้ ก็คือคำสั่ง echo แล้วตามด้วยข้อความหรือสตริงค์ (string)
ข้อความในภาษา PHP จะเริ่มต้นและจบด้วย double quote (") เหมือนในภาษาซี

ตัวอย่าง แสดงข้อความลงในเอกสาร HTML

```
<?
```

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

โปรดสังเกตว่า คำสั่งแต่ละคำสั่งในภาษา PHP จะจบท้ายคำสั่งด้วย semicolon (;) เหมือนในภาษาซี ซึ่ง คำสั่งหรือฟังก์ชันในภาษา PHP นั้นจะเขียนด้วยตัวพิมพ์เล็กหรือใหญ่

บทที่ 3 การใช้ตัวแปรในภาษา PHP

สำหรับการเขียนโปรแกรมสำหรับภาษาคอมพิวเตอร์ระดับสูง สิ่งที่จะขาดเสียมิได้คือ การกำหนดและใช้ตัวแปร (variable) ตัวแปรในภาษา PHP จะเหมือนกับในภาษา Perl คือเริ่มต้นด้วยเครื่องหมาย dollar ($) โดยเราไม่จำเป็นต้องกำหนดแบบของข้อมูล (data type) อย่างเจาะจงเหมือนในภาษาซี เพราะว่า ตัวแปลภาษาจะจำแนกเองโดยอัตโนมัติว่า ตัวแปรดังกล่าว ใช้ข้อมูลแบบใด ในช่วงเวลานั้นๆ เช่น ข้อความ จำนวนเต็ม จำนวนที่มีเลขจุดทศนิยมตรรก เป็นต้น ตัวอย่างการใช้งาน เช่น

```
$mystring = "Hello World!";
$myinteger = 1031;
$myfloat = 3.14;
```

ถ้าเราต้องการจะแสดงค่าของตัวแปร ก็อาจจะใช้คำสั่ง echo ได้ ตัวอย่างเช่น

```
echo "$mystring\n";
echo "$myinteger\n";
echo "$myfloat\n";
```

สัญลักษณ์ \n หมายถึงการขึ้นบรรทัดใหม่ เป็น escape character ตัวหนึ่ง (สำหรับตัวอื่นๆ โปรดดูในตาราง) เมื่อพิมพ์ข้อความเป็นเอาพุต และโปรดสังเกตว่า สำหรับการใช้งานภายในเอกสาร HTML การขึ้นบรรทัดใหม่โดยใช้ \n จะแตกต่างจากการขึ้นบรรทัดโดยใช้
 ใน HTML

```
<?
$mystring = "Hello World!";
$myinteger = 1031;
$myfloat = 3.14;

echo "$mystring<BR>\n";
echo "$myinteger<BR>\n";
echo "$myfloat<BR>\n";
?>
```

Escaped characters

\n	newline

\r	carriage
\t	horizontal □ab
\\	backslash
\$	dollar sign
\"	double-quote
%%	percent

ตัวแปรตัวหนึ่ง อาจจะมีข้อมูลหลายแบบในช่วงเวลาที่ต่างกัน แต่การจะใช้งานบ้างครั้งจะต้องดูด้วยว่า เมื่อไหร่จะใช้เป็นตัวเลขเท่านั้น และไม่ใช้กับข้อความเป็นต้น ตัวอย่างเช่น

```
<?

$x = 10;
$y = $x + 15.5;
echo "$x, $y \n";
$x = "abc";
echo "$x \n";
$z = $x + 15.5;
echo "$x, $z \n";
echo ("100.5" - 16);
echo (0xef + 007);

?>
```

ในกรณีนี้ เรากำหนดในตอนแรกว่า $x ให้เก็บค่า 10 ซึ่งเป็นจำนวนเต็ม ถ้าเรานำมาบวกกับ 15.5 ผลที่ได้ก็จะเป็น 25.5 ซึ่งกลายเป็นเลขทศนิยม แล้วเก็บไว้ในตัวแปร $y ต่อมากำหนดให้ตัวแปร $x เก็บสตริงค์ที่เก็บข้อความ "abc" ถ้าเรานำมาบวกกับ 15.5 กรณีนี้ก็จะให้ผลที่ได้ไม่ถูกต้อง เนื่องจากไม่สามารถนำข้อความมาบวกกับตัวเลขได้

แต่ PHP อนุญาตให้เราทำเช่นนั้นได้ในบางกรณี สมมุติว่า สตริงค์มีเฉพาะตัวเลขและสามารถเปลี่ยนเป็น เลขจำนวนเต็ม หรือจำนวนจริงได้โดยอัตโนมัติ เราก็นำสตริงค์ นี้มาบวกลบคูณหรือหารกับตัวแปรที่เก็บเป็นตัวเลขได้ ค่าคงที่สำหรับเลขจำนวนเต็ม อาจจะอยู่ในรูปของเลขฐานแปดหรือสิบหกก็ได้ ถ้าเป็นเลขฐานแปดจะมีเลข

ศูนย์นำ ถ้าเป็นเลขฐานสิบหกจะมี 0x นำหน้า

บทที่ 4 การอ่านและแปลงแบบข้อมูลในตัวแปรหรือค่าคงที่แบบเจาะจง

เราสามารถแปลงแบบข้อมูลจากแบบหนึ่งไปยังอีกแบบหนึ่ง (type casting) เช่น แปลงจากข้อความที่มีเฉพาะตัวเลขให้กลายเป็นเลขจำนวนเต็ม (int) หรือทศนิยม (double), (float), (real) หรืออาจจะใช้คำสั่ง settype() ทำได้ตามตัวอย่างต่อดังนี้

```
<?

$x = ((double)"100.1") + 0.3e+3;
echo $x," <BR>\n";
echo ($x=(int)$x)," <BR>\n";
$x = "P".$x."\n";
echo $x," <BR>\n";

$x= ceil(13.45);  /* get integer part */
echo $x," <BR>\n";
if (!settype( $x, "integer") ) {
  echo "error\n";
}
echo $x," $x%5=",($x%5)," <BR>\n";


?>
```

บทที่ 5 การอ่านแบบข้อมูลของตัวแปรหรือค่าคงที่

ถ้าต้องการเช็คดูว่า ตัวแปรมีข้อมูลแบบใด เราสามารถใช้คำสั่ง gettype() ได้ ค่าที่ได้จากฟังก์ชันก็จะเป็น "integer" "double" หรือ "string" เป็นต้น

```
<?

echo gettype(0),"\n";
echo gettype(1.1),"\n";
echo gettype(""),"\n";
echo gettype((1==1)),"\n";

$var="abc";
if ( gettype($var)=="string" ) {
 echo "this is a string\n";
}

?>
```

เราอาจจะไม่ใช้ gettype() ก็ได้ แต่เลือกใช้ฟังก์ชัน is_long() สำหรับเช็คค่าที่เป็นเลขจำนวนเต็ม, is_string() สำหรับเช็คค่าที่เป็นสตริงค์, is_double() สำหรับค่าที่เป็นเลขทศนิยม, is_array() สำหรับค่าที่เป็นอาร์เรย์ หรือ is_object() สำหรับค่าที่เป็นออปเจคจากคลาสแทน ซึ่งจะให้ค่าเท่ากับ true (1) ถ้าตัวแปรมีแบบข้อมูล ตรงตามที่กำหนด

```
<?

unset($a);
$a="hello";
if (is_string($a) == true) {
  echo "\$a is a string <BR>\n";
```

```
}

unset($a);
$a[]="red";
$a[]="green";
$a[]="blue";

if (is_array($a) == true) {
  echo "\$a is an array of size ",count($a),"<BR>\n";
}

?>
```

บทที่ 6 ตัวอย่างการใช้ echo เพื่อแสดงข้อความ (เพิ่มเติม)

การพิมพ์ค่าใดๆที่เก็บอยู่ในตัวแปร ถ้าชื่อของตัวแปรอยู่ในสตริงค์ระหว่าง double quote เวลาสร้างเอาพุตแล้วจะอ่านค่าของตัวแปรนั้นก่อนแล้วจึงแทนที่ลงในข้อความ แต่ถ้านำหน้าด้วย backslash (\) ก็จะไม่มีการอ่านค่าของตัวแปร เช่น "\$a" จะให้ผลต่างจาก "$a" สังเกตได้จากตัวอย่างต่อไปนี้

```
<?
$a=1;
echo "\$a=$a <BR>\n";

$test = "test";
echo "$test$test$test<BR>\n";
echo $test,$test,$test,"<BR>\n";

$a = 1;
$b = 2;
echo $a,"+",$b,"=□","$a+$b","<BR>\n";
echo $a,"+",$b,"=",$a+$b,"<BR>\n";
?>
```

สำหรับข้อความในภาษา PHP เราอาจจะใช้ single qoute แทน double quote ได้ แต่เวลาใช้งานร่วมกับ echo หรือ print() จะให้ผลต่างกัน ซึ่งสังเกตได้จากตัวอย่างต่อไปนี้

```
<?
 $a = "aaa";
 $b = 'bbb';
 ech□ "$a  $b<BR>\n";
 echo '$a  $b<BR>\n';
 ?>
```

ตัวแปลคำสั่งจะมองข้ามชื่อตัวแปรและรวมถึงพวก escape sequence ต่างๆด้วยที่อยู่ในข้อความที่ใช้ single quote

บทที่ 7 คำอธิบาย (หมายเหตุ) ในภาษา PHP

ถ้าเราต้องการเขียนคำอธิบายในส่วนใดๆก็ตามของสคริปต์ เราก็จะสามารถทำได้โดยใช้ /* ... */ เหมือนในภาษาซี หรือ // เหมือนในภาษาจาวา หรือ # เหมือน shell script โปรดสังเกตว่า // ใช้เขียนนำคำอธิบายในภายบรรทัดหนึ่งๆ เท่านั้น ส่วน # ใช้เริ่มต้นของบรรทัดที่เขียนคำอธิบาย

```
<?
# comment
$a = 41;  // set $a to 41.
$b =10;  /□ set $b to 10.
$b += $a; /* add $a to $b */
echo $b," \n";
?>
```

สัปดาห์ที่ 2

บทที่ 8 คำการใช้คำสั่งสำหรับคำนวณเลขคณิต

- บวก (+) เช่น $x + $y
- ลบ (-) เช่น $x - $y
- คูณ (*) เช่น $x * $y
- หาร (/) เช่น $x / $y
- หาเศษจากการหาร (%) หรือโมดูลัส เช่น $x % $y การเศษจากการหารโดยปรกติจะใช้กับเลขจำนวนเต็มเท่านั้น ถ้าใช้กับเลขมีจุดทศนิยม จะมีการปัดทิ้งเป็นจำนวนเต็มก่อน

กำหนดให้ $x มีค่าเท่ากับ 7 และ $y มีค่าเท่ากับ 4

$x + $y	11
$x - $y	3
$x * $y	28
$x / $y	1.75
$x % $y	3

กำหนดให้ $x มีค่าเท่ากับ 2.5 และ $y มีค่าเท่ากับ 4

$x + $y	6.5
$x - $y	-1.5
$x * $y	1.0
$x / $y	0.615
$x % $	2

บทที่ 9 การเพิ่มหรือลดค่าของตัวเลขในตัวแปรทีละหนึ่งตามแบบภาษาซีหรือจาวา

- $x++ เพิ่มค่าขึ้นอีกหนึ่ง
- ++$x เพิ่มค่าขึ้นอีกหนึ่ง
- $x-- ลดค่าลงอีกหนึ่ง
- --$x ลดค่าลงอีกหนึ่ง ความแตกต่างของการวาง ++ หรือ -- ไว้ข้างหน้าหรือข้างหลัง คือดูว่า จะอ่านค่าของตัวแปรก่อน (ในกรณีที่มีการอ่านค่าของตัวแปร) หรืออ่านค่าหลังจากการเพิ่มหรือลด

โปรดลองทำตามตัวอย่างแล้วสังเกตผลลัพธ์ที่ได้ในแต่ละกรณี

```
<?
$x=3;
echo $x++,"<BR>\n";
echo $x,"<BR>\n";

$x=3;
echo ++$x,"<BR>\n";
echo $x,"<BR>\n";

$x=3;
echo $x--,"<BR>\n";
echo $x,"<BR>\n";

$x=3;
echo --$x,"<BR>\n";
echo $x,"<BR>\n";
?>
```

บทที่ 10 การกำหนดค่าของตัวแปรที่เป็นตัวเลขหรือสตริงค์โดยใช้ assignment operators

การกำหนดค่า (assignment) หรือเปลี่ยนแปลงค่าให้แก่ตัวแปร จะใช้โอเปอร์เรเตอร์ (assignment operators) ได้ในหลายๆรูปแบบ เหมือนอย่างที่ใช้ในภาษาซี ตามตัวอย่างต่อไปนี้

```
$x=0;
$x += 1;  // the same as $x = $x + 1;
$x--;  // the same as $x = $x - 1;
$x *= 3;  // the same as $x = $x * 3;
$x /= 2; //  the same as $x = $x / 2;
$x %= 4; // the same as $x = $x % 4;

$x="";
$x .= 'A'; // append char to an existing string
$x .= "BC"; // append string  to an existing string
```

จากตัวอย่างข้างบน ในกรณีของการต่อสตริงค์ เราจะใช้จุด (.) เป็นโอเปอร์เรเตอร์

บทที่ 11 การใช้ตัวแปรเป็นชื่อของตัวแปร

ภาษา PHP เปิดโอกาสให้เราสามารถเลือกหรือเปลี่ยนชื่อของตัวแปรได้ ตัวอย่างเช่น

```
<?

$a = "var1";
$$a = 10.3;
echo "$a ${$a} $$a <BR>\n";
echo "$var1 <BR>\n";

?>
```

จากตัวอย่างข้างบน เรากำหนดให้ตัวแปร $a เก็บสตริงค์ "var1" และจะใช้เป็นชื่อของตัวแปรอีกตัวหนึ่ง โดยทางอ้อม $$a เป็นการอ้างถึงตัวแปรที่มีชื่อเดียวกับค่าของตัวแปร $a (ในกรณีนี้คือ var1) ดังนั้นถ้าเราเขียนว่า $$a หรือ $var1 ก็หมายถึงตัวแปรตัวเดียวกัน ถ้าต้องการแสดงค่าของ $$a โดยใช้คำสั่ง echo โดยอยู่ในสตริงค์ (ระหว่าง double quotations) เราจะต้องเขียน ${$a} ไม่ใช่ $$a เพราะว่า ถ้าเขียนตามแบบหลัง ตัวแปลคำสั่งจะอ่านค่า $a ก่อนแล้วแทนที่ลงในข้อความ ซึ่งจะได้ $var1แทนที่จะเป็นการอ่านค่าของ $var1

เทคนิคนี้ยังสามารถใช้ได้กับฟังก์ชัน ตัวอย่างเช่น

```
<?
function foobar()  {
   echo "foobar<BR>\n";
 }

 function callFunc ($f)  {
  if ( is_string($f) == true) {
     $f();
  }
```

```
 }

callFunc("foobar");

?>
```

ตัวอย่างข้างบนอาจจะทำให้เกิดปัญหาถ้าสมมุติว่า $f เป็นชื่อของฟังก์ชันที่ไม่มีอยู่จริง วิธีตรวจสอบคือ การใช้ฟังก์ชัน function_exists() ดังต่อไปนี้

```
<?
function MyFunc() {
 print ("ok..<BR>\n");
}

$f="myFunc";
if ( function_exists($f) ) {
  $f();
}
else {
  echo "$f does not exist!";
}

?>
```

บทที่ 12 การกำหนดค่าคงที่

ในภาษา PHP มีการทำสัญลักษณ์ให้เก็บค่าคงที่ เช่น อาจจะเป็นสตริงค์หรือตัวเลขก็ได้ สามารถทำได้โดยใช้คำสั่ง DEFINE() สัญลักษณ์ที่กำหนดโดยคำสั่ง DEFINE() จะเหมือนกันตัวแปรทั่วๆไป แต่แตกต่างตรงที่ว่า เมื่อนิยามแล้วจะเปลี่ยนแปลงค่าอีกไม่ได้

```
<?

define(PI, 3.141592654);
define(YES, true);
define(NO, false);
define("AUTHOR", "RWS");

echo (PI/3),"<BR>\n";
echo "AUTHOR=".AUTHOR."<BR>\n";
echo "YES=".YES."<BR>\n";

?>
```

นอกจากสัญลักษณ์ที่ผู้ใช้นิยามขึ้นมาได้เองแล้วยังมีสัญลักษณ์กลุ่มหนึ่งที่ได้มีการนิยามไว้ก่อนแล้วในภาษา PHP ตัวอย่างเช่น

__FILE__	เก็บชื่อของไฟล์สคริปต์
__LINE__	เก็บเลขบรรทัดภายในสคริปต์ในตอนที่ใช้
TRUE	มีค่าเป็นจริง
FALSE	มีค่าเป็นเท็จ
PHP_VERSION	เก็บเวอร์ชั่นของ PHP
P□P_OS	เก็บชื่อระบบปฏิบัติการที่ใช้ เช่น Linux

บทที่ 13 การทำขั้นตอนซ้ำหรือวนลูป

การวนลูปหรือสร้างลูปเพื่อทำงานซ้ำเป็นส่วนประกอบสำคัญของโปรแกรมคอมพิวเตอร์ ในภาษา PHP ก็จะใช้โครงสร้างเหมือนภาษาซี ดังต่อไปนี้

- while-do loop
- do-while loop
- for-loop

ตัวอย่างการใช้ while-do loop เพื่อคำนวณค่า เลขยกกำลังสอง ซึ่งมีเลขฐานตั้งแต่ 1 ถึง 10

```
<?

$x = 1;
while ($x <= 10) {
  echo $x*$x,"\n";
  $x++;
}

?>
```

เริ่มต้นด้วยการกำหนดตัวแปร $x ให้มีค่าเป็นหนึ่ง ซึ่งในกรณีนี้ เราใช้เป็นเลขฐาน ในการคำนวณเลขยกกำลังสอง เมื่อเข้าสู่การวนลูปแบบ while-do จะมีการตรวจดูเงื่อนไข ของการวนลูปในแต่ละครั้งว่า เงื่อนไขเป็นจริงอยู่หรือไม่ ในกรณีนี้ เรากำหนดเงื่อนไขในการวนลูปไว้ว่า ถ้าค่าของ $x มีค่าน้อยกว่าหรือเท่ากับ 10 ก็ให้ทำคำสั่งที่อยู่ภายในลูป ซึ่งก็คือ echo $x*$x,"\n"; โดยจะพิมพ์ค่าของผลคูณซึ่งหมายถึงเลขยกกำลังสองนั่นเอง หลังจากนั้น ก็ให้เพิ่มค่าของ $x ทีละหนึ่งในการวนลูปแต่ละครั้ง ค่าของ $x จะเพิ่มขึ้นเรื่อยๆจนมีค่ามากกว่า 10 เมื่อถึงเวลานั้น ก็จะเป็นการจบการวนลูป เพราะว่า เราจะได้ว่า เงื่อนไข ($x <= 10) มีค่าเป็นเท็จ

สมมุติว่า ถ้าเปลี่ยนจาก $x++ เป็น $x-- ปัญหาก็จะเกิดตามมาเวลาใช้งาน คือ แทนที่จะวนลูปแค่สิบครั้ง ก็กลับกลายเป็นว่า เป็นการวนลูปนับครั้งไม่ถ้วน เพราะว่า ค่าของ $x จะลดลงเรื่อยๆในการวนลูปแต่ละครั้ง คือเป็นลบ

และค่าเป็นลบจะน้อยกว่า 10 เสมอ (ยกเว้นแต่ว่า เมื่อถึงจุดเวลาหนึ่งค่าเป็นลบมากๆ จะกระโดดกลับเป็นบวก)

ตัวอย่างการใช้ do-while loop เพื่อคำนวณค่าเลขยกกำลังสอง ซึ่งมีเลขฐานตั้งแต่ 1 ถึง 10

```
<?

$x = 1;
do {
  echo $x*$x,"<BR>\n";
  $x++;
} while ($x < 10);

?>
```

โปรดสังเกตความแตกต่างระหว่างการใช้ while-do และ do-while โดยเฉพาะตรงเงื่อนไข ในการจบการวนลูป ในกรณีของ do-while เราจะกระทำขั้นตอนในลูปก่อนหนึ่งครั้ง แล้วค่อยตรวจดูว่า เงื่อนไขในการวนลูปเป็นจริงหรือไม่ ความแตกต่างนี้ เราสามารถจำได้ง่ายๆ คือว่า ถ้าใช้ do-while จะต้องมีการทำคำสั่ง ภายในลูปหนึ่งครั้งเสมอ แม้ว่าเงื่อนไขโดยเริ่มต้นจะเป็นเท็จก็ตาม ซึ่งแตกต่างจาก while-do ถ้าเงื่อนไขเป็นเท็จตั้งแต่เริ่ม ก็จะไม่มีการทำคำสั่งที่อยู่ในลูป

อีกแบบหนึ่งสำหรับการวนลูปคือใช้ for-loop ทำได้ตามตัวอย่างต่อไปนี้

```
<?

for ($x = 1; $x <=10; $x++) {
  echo $x*$x,"<BR>\n";
}

?>
```

ในบรรทัดที่เริ่มต้นด้วย for ระหว่างวงเล็บเปิดและปิด จะถูกแบ่งเป็นสามส่วนโดยเครื่องหมาย semicolon (;) ใน

ส่วนแรกเราสามารถใส่คำสั่งที่ต้องการจะกระทำก่อนเข้าลูป ส่วนแรกนี้จะมีหรือไม่มีก็ได้ ในส่วนที่สองจะเป็นเงื่อนไขสำหรับการทำ loop และในส่วนที่สามจะคำสั่งที่จะต้องทำเป็นการจบท้ายลูปในแต่ละครั้ง หลักการทำงานของ for-loop จะคล้ายกับ while-do-loop

การใช้งาน for-loop และวางตำแหน่งส่วนต่างๆ อาจจะไม่จำเป็นต้องทำเหมือนกันแต่ให้ผลเหมือนกัน เช่น

```
<?

$x=1;
for ( ; $x <=10; $x++) {
  echo $x*$x,"<BR>\n";
}

$x=1;
for ( ; $x <=10; )  {
  echo $x*$x,"<BR>\n";
  $x++;
}

?>
```

จากตัวอย่างข้างบนที่ผ่านๆมา เป็นการวนลูปจะใช้การนับเลขเพิ่มขึ้นทีละหนึ่ง เรายังสามารถเขียนใหม่โดยเป็นการนับเลขลดลง ยกตัวอย่างเช่น เราต้องการจะพิมพ์ตัวเลขเรียงลำดับจาก 10,9,8...,1 ก็อาจจะเขียนคำสั่งได้ดังนี้

```
<?

for ($x=10 ; $x >0; $x--)  {
    echo $x,"<BR>\n";
}

?>
```

การใช้งาน for-loop ก็จะเหมือนกับเวลาใช้ในภาษาซี ในหลายๆเรื่อง เช่น เราสามารถใส่คำสั่งได้ มากกว่าหนึ่ง โดยใช้เครื่องหมาย (,) เป็นตัวแยก ตัวอย่างเช่น

```
<?

for ($x=1, $y=0 ; $x < 10; $x++, $y--)  {
   echo "$x $y<BR>\n";
}

?>
```

บทที่ 14 การแบ่งสายงานโดยจำแนกตามเงื่อนไขแบบ if-else

ในบางครั้งมีความจำเป็นต้องจำแนกเงื่อนไขในการทำงาน โดยแต่ละเงื่อนไขจะกำหนดกรณี เพื่อทำคำสั่งหรือกลุ่มของคำสั่ง ซึ่งอาจจะแตกต่างจากคำสั่งในกรณีอื่น ในภาษา PHP จะใช้ โครงสร้าง if หรือ if-else ในการจำแนกกรณีตามเงื่อนไข

```
<?

if ($x == 0)
  echo $x,"  is zero<BR>\n";
else if ($x > 0)
  echo $x,"  is positive<BR>\n";
else
  echo $x,"  is negative<BR>\n";

?>
```

จากตัวอย่าง ถ้า $x มีค่าเป็นศูนย์ตามเงื่อนไข ก็จะทำคำสั่ง echo $x," is zero
\n"; ถ้าเงื่อนไขแรกเป็นเท็จ ก็จะเงื่อนไขที่สองว่า $x มีค่ามากกว่าศูนย์หรือไม่ ถ้าใช้ ก็ทำคำสั่ง echo $x," is positive
\n"; ถ้าเงื่อนที่สองเป็นเท็จอีก ก็ให้ทำคำสั่งในกรณีสุดท้ายคือ $x จะต้องมีค่าเป็นลบ

ถ้าในแต่ละกรณีต้องมีการทำคำสั่งมากกว่าหนึ่ง คือ เป็นกลุ่มคำสั่ง จะต้องใช้ { } มากำหนดขอบเขต (scope) เช่น

```
<?

if ($x == 0) {
  echo $x;
  echo "  is zero.<BR>\n";
}
else if ($x > 0) {
```

```
 echo $x;
 echo "  is positive.<BR>\n";
}
else  {
 echo $x;
 echo "  is negative.<BR>\n";
}

?>
```

โปรดสังเกตว่า { } ไม่ต้องมีเครื่องหมาย ; ต่อท้าย

ในภาษา PHP มีการกำหนด elseif (เงื่อนไข) ขึ้นมาใช้ ซึ่งไม่มีอะไรแตกต่างจาก else if (เงื่อนไข)

โครงสร้างแบบ (เงื่อนไข) ? นิพจน์ : นิพจน์ แบบที่ใช้กันในภาษาซีนั้น ก็ใช้ได้เช่นกัน ตัวอย่างเช่น

```
<?

$x= -0.1035;
echo (($x < 0) ? -$x : $x),"<BR>\n";

?>
```

สัปดาห์ที่ 3

บทที่ 15 การใช้ break และ continue ภายในลูป

คำสั่ง break และ continue ภายในลูปอย่างที่ใช้กันในภาษาซี ก็นำมาใช้กับภาษา PHP ได้ ตัวอย่างเช่น

```
<?

unset($a);
$a[]=1;
$a[]=2;
$a[]=3;
$a[]="red";
$a[]="green";
$a[]="blue";
$a[]="none";

$i=0;
$found="not found";
for ($i=0; $i < count($a); $i++) {
  if ( is_long($a[$i]) ) { // skip all integer elements
     continue;
  }
  if ($a[$i] == "blue") {
     $found=$a[$i];
     break;
  }
}
```

```
echo $found,"<BR>\n";

?>
```

คำสั่ง continue บังคับให้ไปเริ่มต้นทำขั้นตอนในการวนลูปครั้งต่อไป ส่วน break นั้นส่งผลให้หยุดการทำงานของลูป

บทที่ 16 การแบ่งสายงานโดยจำแนกตามเงื่อนไขแบบ switch-case

นอกเหนือจากการใช้ if-else ในการจำแนกกรณีตามเงื่อนไขแล้ว เรายังสามารถใช้โครงสร้างแบบ switch-case ได้ ตัวอย่างเช่น

```
switch ($day) {
 case 1 :
  echo "Monday<BR>\n";
  break;
 case 2 :
  echo "Tuesday<BR>\n";
  break;
 case 3 :
  echo "Wednesday<BR>\n";
  break;
 case 4 :
  echo "Thurday<BR>\n";
  break;
 case 5 :
  echo "Friday<BR>\n";
  break;
 case 6:
  echo "Saturday<BR>\n";
  break;
 case 7 :
  echo "Sunday<BR>\n";
  break;
 default :
  echo "error<BR>\n";
```

```
}
```

ถ้าตัวแปร $day มีค่าที่อยู่ระหว่าง 1 ถึง 7 ก็จะพิมพ์ชื่อวันเป็นภาษาอังกฤษ ถ้าตัวแปรมีค่านอกเหนือจากนั้น ซึ่งในกรณีจะเป็น default ในโครงสร้างแบบ switch-case ก็จะพิมพ์คำว่า error เพื่อให้ผู้ใช้ทราบ
โปรดสังเกตว่า ในแต่ละกรณี จะต้องจบด้วยคำสั่ง break; ยกเว้นแต่ของ default ซึ่งจะมีหรือไม่ก็ได้ ถ้าเราไม่ได้ใส่คำสั่ง break; เอาไว้ โปรแกรมก็จะกระทำคำสั่งทุกคำสั่งในกรณีที่อยู่ถัดมา

การจำแนกกรณีไม่จำเป็นต้องอาศัยเฉพาะตัวแปรที่เก็บค่าจำนวนเต็มเท่านั้น ข้อมูลแบบอื่นก็ใช้ได้ เช่น ใช้ข้อความเป็นตัวจำแนกกรณี เช่น

```
switch ($answer) {
 case "yes" :
   echo "The user said 'yes'□\n";
   break;
 case "no" :
   echo "The user said 'no'.\n";
   break;
 default:
  echo "The user said neither 'yes' nor 'no'.\n";
}
```

โปรดสังเกตว่า การจำแนกโดยใช้ข้อความนี้ จะดูความแตกต่างระหว่างตัวพิมพ์เล็กหรือใหญ่ด้วย

ในบางครั้งเราอาจจะไม่จำเป็นต้องใส่ break; ก็ได้ ตัวอย่างเช่น

```
switch ($answer) {
 case "yes" :
 case "no" :
```

```
    echo "The user said '",$answer,"'.\n";
    break;
  default:
    echo "The user said neither 'yes' nor 'no'.\n";
}
```

บทที่ 17 การเปรียบเทียบตัวเลขสำหรับสร้างเงื่อนไข

==	เท่ากับ
>	มากกว่า
>=	มากกว่าหรือเท่ากับ
<	น้อยกว่า
<=	น้อยกว่าหรือเท่ากับ
!=	ไม่เท่ากับ

เราสามารถสร้างเงื่อนไขจากการเปรียบเทียบมากกว่าน้อยกว่านี้ได้ซับซ้อนมากขึ้นโดยใช้ "และ" "หรือ" "ไม่" มาประกอบ ตัวอย่างเช่น

($x == -1) \|\| ($x==1)	ถ้า $x มีค่าเท่ากับ -1 หรือ 1 จะได้เงื่อนไขเป็นจริง นอกเหนือจากนั้นเป็นเท็จ
($x < 10) && ($x >1)	ถ้า $x มีค่าน้อยกว่า 10 และ มากกว่า 1 ก็จะได้เงื่อนไขที่เป็นจริง นอกเหนือจากนั้นเป็นเท็จ
! ($x==0)	ถ้า $x ไม่เท่ากับศูนย์ ก็ได้เงื่อนไขเป็นจริง นอกเหนือจากนั้นเป็นเท็จ

การใช้ || และ && มีลักษณะการทำงานเหมือนในภาษาซี อย่างกรณีของ ($x || $y) ถ้า $x เป็นจริงจะไม่มีการพิจารณา $y และสำหรับ ($x && $y) ถ้า $x เป็นเท็จแล้วจะไม่มีการพิจารณา $y ต่อ

บทที่ 18 การคำนวณเลขคณิตในระดับบิต

การคำนวณแบบบิตที่ใช้ในภาษาซี ก็ใช้ได้กับภาษา PHP ตามตารางข้างล่างนี้

$x & $y	AND
$x | $y	OR
$x ^ $y	XOR
~ $x	NOT
$x << $y	SHIFT LEFT
$x >> $y	SHIFT RIGHT

บทที่ 19 การใช้อาร์เรย์ (Array)

อาร์เรย์ในภาษา PHP นั้นจะแตกต่างจากอาร์เรย์ในภาษาซีหรือจาวาตรงที่ว่า อาร์เรย์ในภาษา PHP มีขนาดที่เปลี่ยนแปลงได้ หรือจะเรียกว่า dynamic array หรือ vector (สำหรับอาร์เรย์มิติเดียว) เริ่มต้นอาจจะแจ้งใช้ตัวแปรแบบอาร์เรย์ พร้อมเจาะจงขนาดเริ่มแรก เช่น มีขนาดเป็นศูนย์ก็ได้

```
$myarray[]=3;
$myarray[]=1.1;
$myarray[]="abc";
```

แต่เมื่อใช้อาร์เรย์ไป ขนาดของมันจะปรับเปลี่ยนได้ คือขยายจำนวนข้อมูลที่เก็บอยู่ภายในอาร์เรย์ ตามจำนวนข้อมูลที่เราใส่เพิ่มเข้าไป จากตัวอย่างข้างบน ในกรณีที่เรามิได้กำหนดเลขดัชนี (index) ก็หมายความว่า จะมีการขยายขนาดของอาร์เรย์เพิ่มขึ้นอีกหนึ่งโดยอัตโนมัติ ทุกครั้งที่เราใส่ข้อมูลที่อยู่ทางขวา และค่าที่เรากำหนดจากทางขวามือ และจะเก็บไว้ในที่ใหม่ของอาร์เรย์ เราไม่ต้องคำนึงถึงเรื่องการจอง หรือ ปลดปล่อยหน่วยความจำของอาร์เรย์ เหมือนอย่างในกรณีของอาร์เรย์ แบบไดนามิกในภาษาซี

นอกจากนั้นข้อมูลแต่ละตัวในอาร์เรย์ไม่จำเป็น ต้องเป็นข้อมูลชนิดเดียวกัน เช่น อาจจะมีทั้งจำนวนเต็ม เลขทศนิยม และข้อความ ปะปนกันไป ตัวอย่างเช่น

```
<?

$myarray[0] = 1;
echo "number of elements =".count($myarray)."<BR>\n";

$myarray[1] = "abc";
echo "number of elements =".count($myarray)."<BR>\n";

$myarray[2] = 1.3;
echo "number of elements =".count($myarray)."<BR>\n";

$myarray[]= 13+10;  // the same as $myarray[3]= 13+10;
```

```
echo "number of elements =".count($myarray)."<BR>\n";

for ($i=0; $i < 4; $i++) {
    echo $myarray[$i]," \n";
}

?>
```

ถ้าเราต้องการจะทราบจำนวนของข้อมูลที่มีอยู่ในอาร์เรย์เราจะใช้คำสั่ง count()

เทคนิคหนึ่งที่ใช้ในการสร้างอาร์เรย์ที่เก็บหลายๆข้อความหรือสตริงค์ คือ แทนที่เราจะกำหนดค่าของสมาชิก ในอาร์เรย์ทีละตัว เราจะสร้างได้โดยอัตโนมัติ โดยเก็บสตริงค์เหล่านั้นไว้ในสตริงค์เพียงอันเดียวโดยมีสัญลักษณ์ | เป็นตัวแยก และก็แล้วใช้ฟังก์ชันเป็นตัวแบ่งเพื่อสร้างอาร์เรย์อีกที ตามตัวอย่าง

```
<?
// create empty array
$a=array();

// define string containing color names separated by | (pipe)
$color_names="red|green|blue";

// create array from string
$a=explode("|",$color_names);
while ( $color=each($a) ) {
  echo "$color[1]<BR>\n";  // note: $color[0] contains the index (0,1,2,...)
}
?>
```

ลองดูอีกตัวอย่างหนึ่งที่ใช้ฟังก์ชัน explode() สร้างอาร์เรย์โดยอัตโนมัติสำหรับใส่ไว้ใน FORM ในส่วนของ SELECT เป็นเมนูให้เลือก

```
<?
// create selection list from a given string
function str2select($str,  $delim)  {
  $options = explode($delim,$str);
  $num = count($options);
  for( $i=0; $i < $num;$i++) {
    echo "<option> $options[$i]</option>\n";
  }
}

$select_str="10 บาท|20 บาท|30 บาท|40 บาท|50 บาท|100 บาท|200 บาท|500 บาท|1000
บาท";

?>

<FORM>
<SELECT NAME="testform">
<? str2select($select_str,"|"); ?>
</SELECT>
</FORM>
```

บทที่ 20 การใช้อาร์เรย์สองมิติ (2 Dimension Array)

ถ้าเราต้องการจะใช้อาร์เรย์แบบสองมิติ (หรือมากกว่า) ก็ทำได้เช่นกัน คือชื่อตัวแปรแล้วตามด้วย [..][..] ตัวอย่างเช่น

```
<?

$dim = 3;
for ($row=0; $row <= $dim; $row++) {
  for ($column=0; $column <= $dim; $column++) {
    $myarray2[$row][$column] = 4*$row + $column;
    echo $myarray2[$row][$column]," ";
  }
  echo "<BR>\n";
}

?>
```

สังเกตว่า สำหรับการใช้งานตัวแปรที่เป็นอาร์เรย์ เราไม่จำเป็นต้องแจ้งใช้ตัวแปรที่เป็นอาร์เรย์ พร้อมกำหนดขนาดก่อนการใช้งาน

อาร์เรย์แบบเชื่อมโยงหรือ associative array

การเก็บข้อมูลในอาร์เรย์แบบนี้จะใช้กับข้อมูลที่จัดเก็บเป็นคู่ๆไป ซึ่งแตกต่างจากอาร์เรย์แบบแรกที่เราได้ทำความรู้จัก ตัวอย่างเช่น ใช้ทำ lookup table เช่น สมมุติว่า "red" ให้แทนค่า 0xff0000 "green" ให้แทนค่า 0x00ff00 และ "blue" 0x0000ff โดยเก็บไว้ในอาร์เรย์ชื่อ $color_table ตามตัวอย่างต่อไปนี้

```
$color_table["red"]   = 0xff0000;
$color_table["green"] = 0x00ff00;
$color_table["blue"]  = 0x0000ff;

$color_name= "red";
echo "value = ".$color_table[ $color_name]."<BR>\n";
```

หรืออีกรูปแบบหนึ่งที่เขียนสร้างอาร์เรย์ดังกล่าวได้ โดยใช้คำสั่ง array()

```
$color_table = array(
    "red"   => 0xff0000,
    "green" => 0x00ff00,
    "blue"  => 0x0000ff
);
```

เราอาจจะสร้างอาร์เรย์เป็นสองมิติก็ได้ เช่น

```
<?
$countries = array (
  "thailand"  => array ( "zone" => "Asia", "D_NAME" => ".th"),
  "malasia"  => array ( "zone" => "Asia", "D_NAME" => ".my"),
  "india"       => array ( "zone" => "Asia", "D_NAME" => ".in"),
  "holland"  => array ( "zone" => "Europe", "D_NAME" => ".nl"),
  "france"    => array ( "zone" => "Europe", "D_NAME" => ".fr")
);

echo "domain name=".$countries[ "thailand"]["D_NAME"]."<BR>\n";
```

```
?>
```

บทที่ 21 การใช้คำสั่ง each และ list สำหรับ associative array

ถ้าเราต้องการจะเข้าถึงข้อมูลแต่ละคู่ที่ถูกเก็บอยู่ใน associative array เราอาจจะใช้วิธีเรียกผ่านฟังก์ชัน each() และ list() ตามตัวอย่างต่อไปนี้

```
<?
unset($a);
$a = array( "a" => 10, "b" => 20, "c" => 30 );

while (list($key,$value) = each($a)) {
  echo "$key=$value <BR>\n";
}
?>
```

ฟังก์ชัน each() จะอ่านข้อมูลทีละคู่จากอาร์เรย์แบบเชื่อมโยงมาแล้วส่งไปยังฟังก์ชัน list() ซึ่งจะทำหน้าที่แยกเก็บ ซึ่งในกรณีก็คือ เก็บไว้ในตัวแปร $key และ $value หลังจากนั้น เราก็สามารถนำค่าของตัวแปร ไปใช้งานตามที่ต้องการได้

สัปดาห์ที่ 4

บทที่ 22 การนิยามและสร้างฟังก์ชันโดยผู้ใช้ (User-defined functions)

ถ้าเราต้องการสร้างฟังก์ชันขึ้นมาใช้งานเองก็ทำได้ โดยเฉพาะอย่างยิ่งในกรณีที่เราต้องการจะ ใช้ชุดคำสั่งเหล่านั้นบ่อยครั้ง เราก็จัดเก็บเป็นฟังก์ชัน เพื่อให้เรียกใช้ได้สะดวก และยังช่วยให้การเขียนโปรแกรมง่ายขึ้นด้วย

การสร้างฟังก์ชันขึ้นใช้เองทำได้โดย ใช้โครงสร้าง

```
function function_name ($arg1, $arg2, .., $argN) {
  ....
}
```

และฟังก์ชันจะให้ค่ากลับคืนหรือไม่ก็ได้ ถ้าต้องการให้ค่ากลับคืนจากการทำงานของฟังก์ชัน ก็จะใช้คำสั่ง return นอกจากนั้น PHP ยังสนับสนุน default parameter ด้วย

ตัวอย่างเช่น การหาค่าสัมบูรณ์ของตัวเลข

```
<?

function myabs ($x) {
  if ($x < 0)
    return -$x;
}

echo myabs(-6),"<BR>\n";
echo myabs(-4+2.034),"<BR>\n";

?>
```

การหาค่าดังกล่าวของตัวเลขใดๆ เราสามารถใช้ฟังก์ชัน abs() หรือเราเขียนขึ้นเองก็ได้ตามตัวอย่างข้างบน

บทที่ 23 การหาค่ามากกว่าและน้อยกว่าจากตัวเลขสองตัวและสลับที่กัน

สมมุติว่า เรามีตัวแปรอยู่สองตัว และเราต้องการจะตรวจดูว่า ตัวแปรตัวแรกมีค่าน้อยกว่าตัวแปรอีกตัวหรือไม่ ถ้าไม่ ก็ให้สลับที่กัน ปัญหานี้เราสามารถแก้ไขได้โดยเขียนฟังก์ชันดังนี้

```
<?

function minmax (&$a,&$b)  {
  if ($a > $b) {
    $t=$a; $a=$b; $b=$t;
  }
}

$x=10;
$y=3;
echo "x=",$x,",y=",$y,"<BR>\n";
minmax($x,$y);
echo "x=",$x,",y=",$y,"<BR>\n";

?>
```

ฟังก์ชัน minmax() เป็นตัวอย่างของฟังก์ชันที่ใช้หลักการของ call-by-reference โปรดสังเกตที่เครื่องหมาย & ที่วางอยู่หน้าตัวแปรที่เป็นอาร์กิวเมนต์ของฟังก์ชัน การเรียกใช้ฟังก์ชันแบบ call-by-reference ช่วยให้เราสามารถผ่านตัวแปรไปยังฟังก์ชัน และให้ฟังก์ชันสามารถเปลี่ยนแปลงแก้ไขค่าของตัวแปรนั้นได้

บทที่ 24 การสลับค่าของตัวแปรสองตัว swap()

ถ้าเราต้องการสลับค่าระหว่างสองตัวแปร เราก็เขียนฟังก์ชัน swap() ขึ้นมา

```
<?

function swap(&$a, &$b) {
  $t = $a;
  $a = $b;
  $b = $t;
}

$x=10;
$y=3;
echo "x=",$x,",y=",$y,"\n";
swap($x,$y);
echo "x=",$x,",y=",$y,"\n";

?>
```

ตัวอย่างข้างบน ก็แสดงให้เห็นวิธีการใช้ call-by-reference อีกเช่นกัน

มีข้อสังเกตอยู่ว่า การใช้ call-by-reference ไม่จำเป็นต้องทำตอนนิยามฟังก์ชันเท่านั้น แต่อาจจะทำตอนผ่านตัวแปรเมื่อเรียกใช้งานจริง ตัวอย่างเช่น

```
<?

function swap($a, $b) {
  $t = $a;
  $a = $b;
```

```
  $b = $t;
}

$x=10;
$y=3;
echo "x=",$x,",y=",$y,"\n";
swap(&$x, &$y);
echo "x=",$x,",y=",$y,"\n";

?>
```

จากตัวอย่างนี้ เราแก้ไขฟังก์ชัน swap() ทำให้ไม่สนับสนุน call-by-reference ดังนั้นเพื่อจะใช้งานได้อย่างถูกต้อง เราก็จะต้องใช้ reference ของตัวแปรเป็นอาร์กิวเมนต์ของฟังก์ชัน ในเวลาที่เรียกใช้ ซึ่งก็คือ swap(&$x,&$y) ถ้าเราไม่ทำอย่างนี้ เช่น เขียนว่า swap($x,$y) ก็จะไม่มีการสลับค่าของตัวแปรทั้งสอง เนื่องจากว่า เมื่ออยู่ภายในฟังก์ชัน swap() แล้ว เราไม่สามารถเปลี่ยนแปลงค่าของตัวแปรเหล่านั้นได้ คืออ่านได้แต่ไม่สามารถกำหนดค่าใหม่ได้

บทที่ 25 การใช้ฟังก์ชันเพื่อสร้างตัวเลขแบบสุ่ม

การใช้ฟังก์ชันเพื่อสร้างตัวเลขแบบสุ่ม หรือ random number generator จะคล้ายกับของภาษาซี คือ เริ่มต้นด้วย srand () โดยจะต้องผ่านค่าที่เรียกว่า seed ซึ่งเป็นเลขจำนวนเต็มใดๆก็ได้ก่อน โดยทั่วไปจะใช้ค่าของเวลาในหน่วยวินาที หรือ Time Stamp ซึ่งสามารถอ่านได้จากฟังก์ชัน date("s") (s หมายถึง second หรือหน่วยวินาที) โดยผ่านค่านี้เป็นค่าของ seed จากนั้นจึงค่อยเรียกใช้ rand()

ตัวอย่างการใช้งาน

```
<?

srand( date("s") );
for ($i=0; $i < 10; $i++) {
  $x = rand() % 10;
  echo $x,"  ";
}

?>
```

คำสั่งนี้จะสร้างตัวเลขโดยการสุ่มเลือกเป็นจำนวน 10 ตัวเลข และพิมพ์ออกทางเอาพุต

ตัวอย่างการใช้งานเพิ่มเติมในรูปของฟังก์ชัน

```
<?

function randInt($low,$high) {
  srand ( date("s") );
  $range = $high - $low;
  $num = (rand() % $range) + $low;
  return $num;
```

```
}

?>
```

ตัวอย่างนี้จะสร้างตัวเลขโดยสุ่มที่อยู่ระหว่างเลขจำนวนเต็มสองค่า และเงื่อนไขของการใช้ฟังก์ชันนี้คือ $low จะต้องมีค่าน้อยกว่า $high และทั้งสองต้องเป็นเลขจำนวนเต็ม

ตัวอย่างการใช้งานเพิ่มเติมในรูปของฟังก์ชันเพิ่มเติม

```
<?
function randStr($len) {
  srand ( date("s") );
  for ($i=0; $i < $len; $i++) {
    $ret_str .= chr( (rand() % 26)+97 );
  }
  return $ret_str;
}
echo randStr(40);

?>
```

ตัวอย่างนี้จะสร้างสตริงค์แบบสุ่มที่มีความยาวตามที่กำหนดและสร้างขึ้นจากตัวอักขระภาษาอังกฤษ

บทที่ 26 การสร้างฟังก์ชันแบบเรียกตัวเอง (recursive function)

ตัวอย่าง การหาค่าแฟลทอเรียล n!

```
<?
function factorial ($n) {
  if ( ($n == 0) || ($n == 1) )
   return 1;
  else
    return $n*factorial($n-1);
}
echo  factorial(4);

?>
```

เงื่อนไขก็ใช้ฟังก์ชัน factorial() จากตัวอย่างข้างบน คือ $n จะต้องเป็นตัวแปรที่เก็บค่าที่เป็นเลขจำนวนเต็ม และไม่เป็นลบ ถ้าเราต้องการจะเขียนฟังก์ชันให้มีความปลอดภัยในการใช้งาน เราก็อาจจะเพิ่มเงื่อนไข เพื่อตรวจเช็คดูก่อนว่า ผู้ใช้ผ่านค่าของตัวแปรที่ตรงตามต้องการหรือไม่ เช่น ไม่ผ่านค่าที่เป็นสตริงค์ หรือเป็นเลขทศนิยม หรือค่าที่เป็นลบ เป็นต้น

ตัวอย่าง การค้นหาข้อมูลแบบ Binary Search ในอาร์เรย์ที่มีการเรียงข้อมูลจากน้อยไปมาก

```
<?

function binSearch(&$key,&$array, $left, $right)
{
  $mid = ceil( ($left + $right) / 2 );

  if ($left > $right)
    return -1;
  if ($array[$mid] == $key)
```

```
    return $mid;
  else if ($key < $array[$mid])
    return binSearch($key, $array, $left, $mid-1);  // recursive call
  else
    return binSearch($key, $array, $mid+1, $right); // recursive call
}

$num=100;
$key = randInt(0, $num);
for($i=0; $i < $num; $i++) {
  $sorted_array[$i] = $i+1;
}

echo binSearch(13, $sorted_array, 0, $num);

?>
```

ตัวอย่าง การสร้างสตริงค์แบบสุ่มอีกแบบหนึ่งซึ่งอาจจะนำไปใช้ในการสร้าง one-time password (OTP)

```
<?

function randomToken($len) {
  srand( date("s") );
  $chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
  $chars.= "1234567890!@#$%^&*()";
  $ret_str = "";
  $num = strlen($chars);
  for($i=0; $i < $len; $i++) {
    $ret_str.= $chars[rand()%$num];
  }
  return $ret_str;
```

```
}

echo randomToken(13)," ";

?>
```

หมายเหตุ: การกำหนดค่า seed สำหรับฟังก์ชัน srand() นอกจะใช้ date("s") เป็นตัวกำหนดค่าแล้ว เราอาจจะใช้ฟังก์ชันอื่นก็ได้ เช่น srand((double)microtime()*1000000);

บทที่ 27 การใช้ตัวแปรแบบ global ภายในฟังก์ชัน

บางครั้งเราไม่ต้องการที่จะผ่านตัวแปรเป็นอาร์กิวเมนต์ของฟังก์ชัน เพื่อนำไปใช้ภายในฟังก์ชันเหล่านั้น ก็จะทำได้โดยการแจ้งใช้ตัวแปรที่มีชื่อเหมือนตัวแปรภายนอกที่เราต้องการใช้ ให้เป็น global หรือใช้ผ่านตัวแปรที่เป็นอาร์เรย์ของ PHP ที่มีชื่อว่า $GLOBALS ดังตัวอย่างต่อไปนี้

```
<?
$a = 10;
$b = 20;

function getMin ( ) {
 global $a, $b;

 if ($a < $b)
  return $a;
 else
  return $b;
}

function getMin2 () {
 if ($GLOBALS["a"] < $GLOBAL["b"])
  return $GLOBALS["a"];
 else
  return $GLOBALS["b"];
}

echo getMin()."<BR>\n";
echo getMin2()."<BR>\n";
?>
```

ในกรณีนี้เราต้องการจะใช้ตัวแปร $a และ $b ซึ่งอยู่นอกฟังก์ชัน getMin() เพื่อเช็คดูว่า ค่าของตัวแปรใดมีค่าน้อยกว่ากัน ถ้าเราไม่แจ้งใช้ global $a, $b; ตามตัวอย่างแล้ว $a และ $b จะกลายเป็นตัวแปรภายในแม้ว่าจะชื่อเหมือนกันตัวแปรภายนอกที่มีอยู่แล้วก็ตาม ทำให้ได้ผลการทำงานไม่ถูกต้องตามที่ต้องการ

ฟังก์ชัน getMin() อีกรูปแบบหนึ่ง โดยไม่ใช้ตัวแปรแบบ global ภายในฟังก์ชัน และใช้วิธีผ่านค่าแทน

```
<?
$a = 10;
$b = 20;

function getMin ($a, $b) {
  if ($a < $b)
     return $a;
  else
     return $b;
}

echo getMin($a, $b)."<BR>\n";
?>
```

บทที่ 28 การตัวแปรแบบ static ภายในฟังก์ชัน

สมมุติว่า เราต้องการจะใช้ตัวแปรภายในฟังก์ชัน และสามารถเก็บค่าไว้ได้ตลอดเวลา โดยไม่สูญหายไปทุกครั้งที่มีการเรียกใช้ฟังก์ชัน ในกรณีนี้เราจะแจ้งใช้ตัวแปรให้เป็นแบบ static ตามตัวอย่างต่อไปนี้

```
function MyFunc() {
  static $num_func_calls = 0;
  echo "my function\n";
  return ++$num_func_calls;
}
```

ทุกครั้งที่มีการเรียกใช้ฟังก์ชันดังกล่าว ตัวแปรชื่อ $num_func_calls ซึ่งมีค่าเริ่มต้นเป็นศูนย์ในตอนแรก จะเพิ่มค่าที่เก็บขึ้นทีละหนึ่ง

สัปดาห์ที่ 5

บทที่ 29 การผ่านค่ากลับคืนมากกว่าหนึ่งจากฟังก์ชัน

โดยปรกติแล้วเราไม่สามารถผ่านค่ากลับคืนจากฟังก์ชันได้มากกว่าหนึ่ง แต่อย่างไรก็ตาม ยังมีวิธีการหนึ่งที่ช่วยแก้ปัญหาดังกล่าวได้ วิธีนี้คือ เก็บค่าต่างๆที่ต้องการจะใช้เป็นค่ากลับคืนไว้ใน array แล้วใช้ array นั้นเป็นค่ากลับคืน และผู้เรียกใช้ฟังก์ชันสามารถใช้ฟังก์ชัน list() อ่านค่าเหล่านั้นได้ ตัวอย่างเช่น

```
<?
function foobar() {
   return array ("foo", "bar", 0xff);
}

list ($foo, $bar, $num) = foobar();
echo "$foo $bar $num <BR>\n";
?>
```

จากตัวอย่าง ฟังก์ชัน foobar() จะให้ค่ากลับคืนเป็น array ประกอบด้วยสามสมาชิก ค่าที่ได้จากฟังก์ชันนี้ก็จะส่งไปยังฟังก์ชัน list() เพื่อให้เก็บแยกลงในตัวแปรตามชื่อที่กำหนดคือ $foo, $bar และ $num ตามลำดับ

บทที่ 30 การอ่านตัวแปรจากภายนอกที่ได้จากการ Web browser โดยวิธี GET หรือ POST

สมมุติว่า เรามีฟอร์มสำหรับให้ผู้ใช้ป้อนชื่อ (login) และรหัสผ่าน (password) จากนั้นก็ส่งมายัง Webserver และใช้สคริปต์ PHP เป็นตัวจัดการกับข้อมูลที่ส่งมาโดยวิธีการแบบ POST ตามตัวอย่าง

```
<form action="login.php3" method="post">
<table>
<tr><td>login:</td>
<td><input type="text" name="login"></td>
</tr><br>
<tr><td>password:</td>
<td><input type="text" name="password"></td>
</tr><br>
</table>
<p><input type="submit">
</form>
```

ภายในสคริปต์ login.php3 เราสามารถอ่านข้อมูลที่ส่งมาได้ ในกรณีนี้ ที่เราสนใจคือ ค่าจาก login และ password ที่อยู่ในฟอร์ม และสามารถจะอ่านข้อมูลเหล่านั้นได้ เพราะ PHP จะเก็บข้อมูลไว้ในตัวแปรชื่อ $login และ $password ตามลำดับ ตัวอย่างเช่น login.php3 อาจจะเป็นสคริปต์ง่ายๆดังนี้

ตัวอย่างไฟล์ login.php3

```
<HTML>
<HEAD><TITLE> Result </TITLE></HEAD>
<BODY>
<P> Your login = <? echo "$login" ?>
<BR> Your password = <? echo "$password"; ?>
</BODY>
</HTML>
```

เราสามารถอ่านข้อมูลที่ได้จากการส่งแบบ GET ได้เช่นกัน ตัวอย่างเช่น

```
<a href="print.php3?ID=103543564&mode=plaintext"> click </a>
```

เมื่อผู้ใช้คลิ้กที่ link ก็จะติดต่อกับสคริปต์ที่ชื่อว่า print.php3 โดยผ่านข้อมูลสองตัวคือ ID และ mode ภายในสคริปต์ เราก็ใช้ $ID และ $mode สำหรับอ่านค่าของข้อมูลที่ส่งมา ในตัวอย่างนี้ $ID จะได้ค่าเป็น "103543564" และ $mode ได้ค่า "plaintext"

บทที่ 31 การตรวจดู web browser ของผู้มาเยือนว่าเป็นตัวไหน

อีกตัวอย่างหนึ่งที่แสดงให้เห็นการใช้ตัวแปรแบบ global ซึ่งเป็นตัวแปรที่ตัวแปลชุดคำสั่ง PHP ได้สร้างขึ้น ทุกครั้งที่ทำงาน หนึ่งในตัวแปรนั้นคือ $HTTP_USER_AGENT

```
<?
 function getBrowserName() {
  global $HTTP_USER_AGENT;
  $browser=strtoupper($HTTP_USER_AGENT);
  if (strstr($browser,"MSIE."))
    return "MS Internet Explorer";
  else if (strstr($browser,"MOZILLA"))
    return "Netscape";
  else
    return "";
 }

 $name = getBrowserName();
 if ($name != "") {
   echo "Your browser is ".$name.".<BR>";
 }

?>
```

จากตัวอย่าง เราสามารถใช้ตัวแปรดังกล่าวในการตรวจดูว่า ผู้ใช้ได้ใช้ web browser ตัวไหน เช่น ระหว่าง IE (Microsoft Explorer) หรือ Mozilla (Netscape)

บทที่ 32 การสร้างและใช้งานคลาส (class) และออปเจค (object)

ภาษาแบบ scripting language ในปัจจุบันหลายๆภาษาก็สนับสนุนการเขียนโปรแกรมเชิงวัตถุด้วย ตัวอย่างเช่น Perl และ PHP ก็รวมอยู่ในนั้นด้วย แม้ว่าจะไม่ซับซ้อนเหมือนอย่างภาษาซีพลัสพลัสหรือจาวาก็ตาม

คลาสคือโครงสร้างที่ประกอบด้วยสมาชิก (class members) หรือคุณสมบัติ (properties) ตามแต่จะเรียก และฟังก์ชันสมาชิก (member functions) การนิยามคลาสขึ้นมาใช้งานจะเริ่มด้วย class { ... } โดยข้างในจะมีส่วนของตัวแปรสมาชิก และฟังก์ชันสมาชิกตามลำดับ ฟังก์ชันที่มีชื่อเดียวกับคลาสจะเรียกว่า class constructor ทุกครั้งที่มีการสร้างออปเจคจากคลาสโดยใช้คำสั่ง new ฟังก์ชันที่ทำหน้าที่เป็น class constructor ก็จะถูกเรียกมาทำงานก่อนทุกครั้ง ประโยชน์ของการใช้งานก็เช่น ใช้กำหนดค่าเริ่มต้น หรือเตรียมพร้อมก่อนที่จะเริ่มใช้ออปเจค

ลองดูตัวอย่าง การเขียนคลาสสำหรับแบบข้อมูลเชิงนามธรรม (Abstract Data Type) ที่เรียกว่า stack การทำงานของ stack ก็เป็นดังนี้ ถ้าเราใส่ข้อมูลเข้าไป ข้อมูลเหล่านั้นก็จะถูกเก็บไว้เสมือนกับว่า วางซ้อนกันจากข้างล่างขึ้นข้างบน ถ้าเราจะดึงข้อมูลออกมาใช้ก็จะได้ข้อมูลที่อยู่ข้างบนสุด ซึ่งก็คือข้อมูลที่เราใส่เข้าไปครั้งล่าสุดนั่นเอง หน้าที่ของ stack ที่สำคัญก็มีเช่น

push()	ใส่ข้อมูลไว้ใน stack
pop()	ดึงข้อมูลออกมา
is_empty()	ตรวจดูว่า stack มีข้อมูลอยู่อีกหรือไม่
get_size()	หาจำนวนของข้อมูลที่ถูกเก็บไว้ใน stack

ตัวอย่างการสร้างคลาส stack ในภาษา PHP ทำได้ดังตัวอย่างต่อไปนี้

```
<?

class stack {
  var $arrays;
```

```
  var $size;

  function stack() { // class constructor
    $this->size = 0;
    unset($this->arrays);
  }

  function push($elem) {  // put an element on stack
    $this->arrays[$this->size] = $elem;
    $this->size++;
  }

  function get_size() {  // get number of elements stored
    return $this->size;
  }

  function is_empty() { // is stack empty ?
    return ($this->size == 0) ? true : false;
  }

  function pop() { // retrieve an element from the top of stack
    if ( $this->is_empty() == false ) {
      $this->size--;
      return $this->arrays[$this->size];
    }
    else
      return 0;
  }
}
```

```
$inst = new stack; // create an object from stack class
echo "initial stack size=".($inst->get_size()),"<BR>\n";

for ($i=0; $i < 10; $i++) {
  $inst->push( ($i*7)%11 );
}
echo "current stack size=".($inst->get_size()),"<BR>\n";

while (! $inst->is_empty() ) {
  echo "pop ".$inst->pop(),"<BR>\n";
}

echo "stack is ".($inst->is_empty() ? "empty." : "not empty.")."<BR>\n";

$inst = 0; // unuse this instance of class stack
?>
```

โปรดสังเกตว่า ตัวแปร $this ที่ปรากฏอยู่ในคลาสจะเหมือน this ที่เราใช้ในภาษาซีพลัสพลัส และการนิยามและสร้างฟังก์ชันสมาชิกจะทำภายในคลาสทั้งหมด (เหมือนในภาษาจาวา)

PHP ยังสนับสนุนการสืบทอดคุณสมบัติของคลาส (inheritance) ทำให้เราสามารถสร้างคลาสขึ้นมาใหม่ โดยใช้คลาสที่มีอยู่เดิมและเพื่อส่วนขยายเข้าไป การสืบสอดคุณสมบัติจากคลาสหนึ่งไปยังอีกคลาสหนึ่ง จะใช้คำสั่ง extends คล้ายกับของภาษาจาวา ตามตัวอย่างดังนี้

```
<?

class stack {
   var $arrays;
   var $size;
```

```
  function stack() {
    echo "DEBUG> stack constructor<BR>\n";
    $this->size = 0;
    unset($this->arrays);
  }

  function push($elem) {
    $this->arrays[$this->size] = $elem;
    $this->size++;
  }

  function get_size() {
    return $this->size;
  }

  function is_empty() {
    return ($this->size == 0) ? true : false;
  }

  function pop() {
    if ( $this->is_empty() == false ) {
      $this->size--;
      return $this->arrays[$this->size];
    }
    else
      return 0;
  }
}
```

```
// class LimitedStack is derived from class stack.

class LimitedStack extends stack {
  var $max_size;

  function LimitedStack ($capacity = 10)  {
    $this->stack(); // call stack's constructor explicitly.
    echo "DEBUG> LimitedStack constructor<BR>\n";
    $this->max_size = $capacity;
  }

  function is_full() {
   return ($this->max_size <= $this->size) ? true : false;
  }

  function push($elem) {
   if ($this->is_full() == false) {
     $this->arrays[$this->size] = $elem;
     $this->size++;
   }
   else {
     echo "stack is full!\n";
   }
  }
}

$inst = new LimitedStack(5);
echo "initial stack size=".($inst->get_size()),"<BR>\n";
for ($i=0; $i < 10; $i++) {
 if (! $inst->is_full() ) {
```

```
    $inst->push( ($i*7)%11 );
  }
  else  break;
}
echo "current stack size=".($inst->get_size()),"<BR>\n";
echo "stack is ".($inst->is_empty() ? "empty." : "not empty.")."<BR>\n";
```

คลาส LimitedStack นี้มีคุณสมบัติที่ได้รับมาจากคลาส stack แต่แตกต่างตรงที่ว่า เราได้กำหนดความจุ ของ LimitedStack เอาไว้ โดยตัวแปร $max_size ผู้ใช้จะต้องกำหนดขนาดความจุของออปเจคจากคลาส LimitedStack ก่อนใช้ ถ้าไม่กำหนดก็จะใช้ค่า 10 เป็นค่าความจุโดยอัตโนมัติตามตัวอย่าง (เป็น default parameter)

เมื่อมีการกำหนดความจุก็จะต้องมีการเขียนฟังก์ชันสมาชิกเพิ่มขึ้นอีก ชื่อ is_full() เพื่อตรวจสอบดูว่าจำนวนของข้อมูลใน stack เท่ากับความจุที่กำหนดไว้แล้วหรือไม่

โปรดสังเกตว่า PHP สนับสนุนการนิยามฟังก์ชันทับฟังก์ชันเดิมของคลาสที่ได้รับคุณสมบัติมา และสิ่งที่จะลืมไม่ได้คือ constructor จากคลาสลูก (child class) จะไม่เรียกใช้ constructor จากคลาสแม่ (parent class) จะต้องมีการเรียกใช้อย่างเจาะจง

ในกรณีที่เราสร้างอาร์เรย์สำหรับเก็บออปเจค เวลาจะใช้ออปเจคแต่ละตัว จะต้องใช้ตัวแปรช่วยตามตัวอย่างต่อไปนี้

```
<?
 // array of objects

class MyObj {
 var $id;

 function MyObj( $set_id) {
```

```
  // $id = $set_id; <-- this does't work if you forget to use $this
  $this->id = $set_id;
 }

 function show() {
  echo "hello world $this->id<BR>\n";
 }
}

// can create the array of objects
$obj_array = array();
$obj_array[] = new MyObj(1);
$obj_array[] = new MyObj(2);
$obj_array[] = new MyObj(3);

// To access each object we must use help variable like follows:
for($i=0; $i < count($obj_array); $i++) {
  $tmp = $obj_array[$i];
  $tmp->show();
}

?>
```

จากตัวอย่างเราใช้ตัวแปร $tmp ในการเข้าถึงออปเจคแต่ละตัวในถูกเก็บไว้ในอาร์เรย์ $obj_array เนื่องจากเราไม่สามารถเรียกใช้ฟังก์ชันของออปเจคได้โดยตรงถ้าออปเจคอยู่ในอาร์เรย์ เช่น $obj_array[0]->show();

บทที่ 33 การอ่านค่าวันและเวลาปัจจุบัน

การอ่านค่าสำหรับบ่งบอกวันเดือนปีและเวลาในปัจจุบัน เราสามารถใช้คำสั่ง date () ตัวอย่างเช่น แสดงวันเดือนปีของวันนี้

```
<?
$today = date("Y-m-d");
print "<CENTER>Today is: $today.</CENTER>";
?>
```

"Y-m-d" หมายถึงสตริงค์ที่กำหนดรูปแบบ (formatted string) ของการแสดงวันที่ ในกรณีนี้คือ ปีค.ศ.-เดือน-วัน ตามลำดับ จริงๆแล้วฟังก์ชัน date() จะต้องการอาร์กิวเมนต์สองตัวคือ สตริงค์ที่กำหนดรูปแบบ เช่น "Y-m-d" และค่าของ TimeStamp (integer) ในหน่วยเป็นวินาที นับตั้งแต่ 1 มกราคม 1970 ในกรณีที่เราไม่ได้กำหนด TimeStamp ก็จะหมายถึง TimeStamp เวลาในปัจจุบัน

ถ้าเราต้องการแสดงทั้งเวลาและวันเดือนปี ก็ต้องกำหนดรูปแบบของสตริงค์ใหม่ เช่น "D d F Y h:i:s"
ซึ่งตัวอักษรแต่ละตัวจะมีความหมายและเป็นตัวบ่งบอกหน้าที่ เช่น d ใช้แทนที่วันในหนึ่งเดือน D ใช้แทนชื่อวันแบบย่อในเจ็ดวัน F ใช้แทนชื่อเดือนในทั้งหมด 12 เดือน Y แทนที่ปีค.ศ. เป็นเลขสี่หลัก
h i s ใช้แทนชั่วโมง นาที และวินาทีตามลำดับ

```
<?
$today = date("D d F Y h:i:s");
print "<CENTER>Today is: $today.</CENTER>";
?>
```

สำหรับรายอื่นเพิ่มเติมเกี่ยวกับฟังก์ชัน date() สามารถดูได้จาก PHP manual

บทที่ 34 ตัวอย่างฟังก์ชันที่เกี่ยวข้องกับการทำงานของสตริงค์

สองฟังก์ชันแรกที่เราจะทำความรู้จักคือ ฟังก์ชัน strtolower() และ strtoupper() ซึ่งมีหน้าที่คือ เอาไว้แปลงตัวอักขระภาษาอังกฤษให้เป็นตัวพิมพ์เล็ก หรือตัวพิมพ์ใหญ่ทั้งหมด ตามตัวอย่างต่อไปนี้

```
<?
  $answer = "Yes";
  if ($answer == "yes")
    echo "yes...\n";
  else
    echo "error!\n";

  $answer = strtolower("Yes");
  if ($answer == "yes")
    echo "yes...\n";
  else
    echo "error!\n";

  $answer = strtoupper("Yes");
  if ($answer == "YES")
    echo "YES...\n";
  else
    echo "error!\n";
?>
```

ประโยชน์ของฟังก์ชันทั้งสองที่เห็นได้ชัด คือ เอาไว้ใช้แปลงข้อความให้เป็นตัวพิมพ์ใหญ่หรือเล็กทั้งหมดก่อนที่เราจะใช้ในการเปรียบเทียบข้อความ เช่น ผู้ใช้อาจจะใส่ข้อความไว้ใน $answer ว่า "Yes" "YeS" "yES" หรือ "YES" เป็นต้น แต่เราอยากรู้ว่า ผู้ใช้ใส่คำว่า yes หรือไม่ โดยไม่สนใจว่าจะเป็น ตัวพิมพ์ใหญ่หรือเล็ก ในกรณีนี้ เราก็แปลงให้เป็นตัวพิมพ์เล็กก่อน แล้วก็นำมาเปรียบเทียบ

สมมุติว่า มีสตริงค์หรือข้อความอยู่แล้วต้องการจะแยกออกเป็นส่วนย่อยๆ โดยใช้ตัวอักขระ หรือสตริงค์ที่มีอยู่ข้างในเป็นตัวแยก เราจะใช้ฟังก์ชัน explode() ตามตัวอย่างต่อไปนี้

```
<?
  $str = "ohh:users:bash";
  list($user,$group,$shell) = explode(":",$str);
  echo "$user $group $shell";
?>
```

จากตัวอย่างข้างบนเราใช้ ":" เป็นตัวแยกส่วนของข้อความว่า "ohh:users:/bash" และค่าที่ได้จากฟังก์ชัน explode() จะเป็น array ดังนั้น เราก็สามารถใช้ฟังก์ชัน list() เก็บส่วนของข้อความที่ถูกแยกแล้วได้ ในกรณีนี้มีสามส่วนและถูกแยกเก็บไว้ในตัวแปร $user $group และ $shell ตามลำดับ

ฟังก์ชันที่ทำงานตรงกันข้ามกับฟังก์ชัน explode() คือฟังก์ชัน join ตัวอย่างการใช้งานมีดังนี้

```
<?
  unset($a);
  $a[]="aaa";
  $a[]="bbb";
  $a[]="ccc";
  echo join(":",$a)."<BR>\n";
?>
```

บทที่ 35 การแปลง \n ให้เป็น

ฟังก์ชัน nl2br จะทำหน้าที่แปลง \n ให้เป็น
 สำหรับขึ้นบรรทัดใหม่ในเอกสาร HTML โดยอัตโนมัติ เช่น สมมุติว่า เราเปิดไฟล์และอ่านข้อความจากไฟล์นั้น แล้วต้องการจะแทรกข้อความเหล่านั้น เป็นบรรทัดๆ ลงในเอกสาร HTML เนื่องจากว่าในข้อความที่เป็นสตริงค์และมี '\n' จบท้าย และเราต้องการจะแปลงให้เป็น
 เพื่อจัดหน้าเอกสารให้เหมาะสม เราก็ใช้ฟังก์ชันดังกล่าวช่วย

```
<?
  // convert \n to <br>
  $br=nl2br("\n\n");
  echo $br."hello".$br;
?>
```

สัปดาห์ที่ 5.5

บทที่ 36 การใช้คำสั่ง include และ require

คำสั่งทั้งสองเอาไว้แทรกเนื้อหาจากไฟล์อื่นที่ต้องการ ข้อแตกต่างระหว่าง include และ require อยู่ตรงที่ว่า ในกรณีของการแทรกไฟล์ใช้ชื่อต่างๆ กันมากกว่าหนึ่งครั้งโดยใช้ลูป คำสั่ง require จะอ่านเพียงแค่ครั้งเดียว คือ ไฟล์แรก และจะแทรกไฟล์นี้เท่านั้นไปตามจำนวนครั้งที่วนลูป ในขณะที่ include สามารถอ่านได้ไฟล์ต่างๆ กันตามจำนวนครั้งที่ต้องการ

```
<?
$filename[]="file1.inc";
$filename[]="file2.inc";
for ($i = 0; $i < 2; $i++) {
   include $filename[$i];
}
?>
```

ไฟล์ file1.inc

```
Hello world 1<BR>
```

ไฟล์ file2.inc

```
Hello world 2<BR>
```

ตัวอย่างการแทรกไฟล์ที่มีคำสั่งสคริปต์

```
<?
include ("script.inc");
?>
```

ไฟล์ script.inc:

```
<P><CENTER><BLINK><? echo "Hi, How are you!" ?></BLINK></CENTER>
```

การแทรกไฟล์ภายในโครงสร้างของ if-else หรือ for-loop เป็นต้น มีข้อควรระวังเวลาใช้ คือ จะต้องใส่ { } เอาไว้เพื่อให้อยู่ในบล็อกของโครงสร้าง

```
if ($version < 1.0) {
  include ($DOCUMENT_ROOT."/old.inc");
}
else {
  include ( $DOCUMENT_ROOT."/new.inc") ;
}
```

ดังนั้นควรจะระมัดระวัง การแทรกไฟล์โดยใช้ include หรือ require ในตำแหน่งๆต่าง โดยเฉพาะอย่างยิ่งในกรณีที่ แทรกไฟล์ที่มีคำสั่ง PHP อยู่ด้วย

บทที่ 37 Regular Expression

Regular Expression หรือเรียกย่อๆว่า Regex หมายถึง รูปแบบของลำดับ หรือกลุ่มของสัญลักษณ์ ที่ใช้แทนลำดับ หรือกลุ่มของอักขระตามที่ต้องการ

เราใช้สัญลักษณ์ [ ] (square brackets) เพื่อกำหนดขอบเขตของกลุ่มตัวอักขระหลายตัวที่ใช้เป็นตัวเลือก เช่น สมมุติว่า เราต้องการจะเขียนรูปแบบที่ใช้แทนตัวอักขระหนึ่งตัว อะไรก็ได้จาก {a,e,i,o,u} เราก็จะเขียนว่า [aeiou] โดยจะเรียงลำดับก่อนหลังอย่างไรก็ได้ เช่น [eioua] ให้ผลเหมือนกับ [aeoui] หรือ ถ้าเราต้องการเขียนรูปแบบเพื่อใช้แทนตัวขระหนึ่งตัวที่เป็นตัวเลขตัวใดตัวหนึ่งจาก 0 ถึง 9 เราก็เขียนว่า [0123456789] หรือจะเขียนแบบสั้นๆใหม่ได้เป็น [0-9] หรืออีกตัวอย่างหนึ่ง ถ้าเราต้องการจะเขียนนิพจน์แบบ regex ขึ้นมา เพื่อใช้แทนอักขระตัวใดตัวหนึ่งที่เป็นได้ทั้งตัวพิมพ์ใหญ่หรือเล็กในภาษาอังกฤษหรือตัวเลขระหว่าง 0 ถึง 9 เราก็เขียนว่า [A-Za-z0-9]

[aeiou]	ตัวอักขระตัวหนึ่งจาก {a,e,i,o,u} ตัวไหนก็ได้
[0-9]	ตัวอักขระตัวหนึ่งจาก {0,1,...,9} ตัวไหนก็ได้
[A-Za-z0-9]	ตัวอักขระตัวหนึ่งจาก {A,B,..,Z, a, b, ... , z, 0, 1, ... 9} ตัวไหนก็ได้

ถ้าเรามีข้อความแล้วเราต้องการจะค้นหาอักขระหรือลำดับของอักขระ (หรือ pattern) ในข้อความเรานั้น เราเรียกขั้นตอนในการค้นหาตามรูปแบบนี้ว่า pattern matching ในภาษา PHP จะมีฟังก์ชันที่เราใช้ในการค้นหาลำดับของตัวอักขระตามแบบที่ต้องการคือ ereg() และ eregi() และต่างกันตรงที่ว่า ฟังก์ชัน eregi() จะเปรียบเทียบโดยไม่คำนึงถึงเรื่องตัวพิมพ์เล็กหรือใหญ่

ตัวอย่างเช่น สมมุติว่า เรามีข้อความอยู่ในอาร์เรย์เป็นข้อความที่มีแค่ตัวอักขระตัวเดียว แล้วเราต้องการจะหาว่าตัวไหนบ้างที่เป็นตัวเลข 0 ถึง 9 บ้างและตัวไหนบ้างที่เป็นตัวพิมพ์ภาษาอังกฤษ a, b, หรือ c เราก็เขียนสคริปต์โดยใช้ฟังก์ชัน ereg() ได้ดังนี้

```
<?  $a=array("0","1","2","3","5","7","a","b","c");

 for ($i=0; $i < count($a); $i++) { // print only digit
   if ( ereg("[0-9]",$a[$i]) ) {
```

```
    print ("$a[$i] <BR>\n");
  }
 }
?>
<HR>
<?
 for ($i=0; $i < count($a); $i++) { // print only a, b or c
  if ( ereg("[a-c]",$a[$i]) ) {
    print ("$a[$i] <BR>\n");
  }
 }
?>
```

บทที่ 38 ก่อนจบ

จากเนื้อหาที่เขียนมาในเอกสารนี้ ได้อธิบายเฉพาะส่วนของหลักภาษา PHP ซึ่งเป็นพื้นฐาน ในการนำไปเขียนสคริปต์ และถ้าได้ทำความเข้าใจอย่างดีแล้ว ก็สามารถนำไปใช้งานได้ ในระดับหนึ่ง และถ้าเริ่มศึกษาการใช้งานคุณลักษณะต่างๆที่เราจะได้จาก PHP ซึ่งมีอยู่และแบ่งออกเป็นหมวดหมู่ เช่น ความสามารถในการสร้างกราฟิกเป็นไฟล์ GIF หรือสร้างเอกสารแบบ PS หรือ PDF การเชื่อมโยงเข้ากับ database หรือการเข้ารหัสข้อมูลในการรับส่งข้อมูล เป็นต้น ซึ่งยังไม่ได้กล่าวถึงในเอกสารนี้ ก็จะทำให้เราสามารถมองเห็นประโยชน์ของเครื่องมือนี้ได้อย่างชัดเจน สิ่งเหล่านี้ถือว่า เป็นสิ่งสำคัญ และเป็นประโยชน์ที่เราจะได้รับจากการใช้ PHP และเพื่อความเหมาะสม จึงขอแยกเขียนและอธิบายการใช้งานในโอกาสต่อไป...

แนะนำ Code PHP