บทที่ 5
ฟังก์ชัน

ในบทนี้จะกล่าวถึง PHP ฟังก์ชัน รวมถึงวิธีการสร้าง ข้อกำหนด คุณสมบัติต่างๆ และการเรียกใช้งานฟังก์ชัน การส่งค่าตัวแปรเพื่อประมวลผลในฟังก์ชัน การส่งค่ากลับเมื่อฟังก์ชันทำงานเสร็จ เป็นต้น หน้าที่หลักๆ ของฟังก์ชันนั้น จะช่วยลดขั้นตอนการเขียนโปรแกรมที่ต้องทำซ้ำๆ หรือใช้งานบ่อยครั้ง ทำให้การเขียนโปรแกรมทำได้ง่ายและรวดเร็ว สามารถสร้างเป็นไลบรารีฟังก์ชันสำหรับการใช้งานในครั้งต่อๆ ไป

ฟังก์ชัน (Functions) คือ กลุ่มหรือชุดของคำสั่งที่สร้างขึ้นเพื่อทำหน้าที่ใดหน้าที่หนึ่ง เมื่อต้องการใช้งานก็เพียงเรียกชื่อฟังก์ชันนั้นก็สามารถใช้งานได้ทันที ฟังก์ชันสามารถแบ่งได้เป็น 2 กลุ่ม ประกอบด้วย 1) ฟังก์ชันมาตรฐาน (Built-in Functions) และ 2) ฟังก์ชันที่ผู้ใช้งานเป็นผู้สร้างเอง (User Defined Functions)

ฟังก์ชันทั้ง 2 กลุ่ม สามารถจำแนกย่อยเป็น 2 ประเภทย่อยเหมือนกันทั้ง 2 กลุ่ม คือ 1) ฟังก์ชันไม่มีการส่งผ่านค่าพารามิเตอร์ และ 2) ฟังก์ชันที่มีการส่งผ่านค่าพารามิเตอร์ (พารามิเตอร์ หรือเรียกว่า อากิวเมนต์ (argument) หมายถึง ตัวแปรหรือข้อมูลที่ส่งจากภายนอกฟังก์ชันเข้ามาประมวลผลภายในฟังก์ชัน) ในหนังสือเล่มนี้จะใช้คำว่า พารามิเตอร์ เพียงอย่างเดียวเมื่อกล่าวถึงตัวแปรหรือข้อมูลที่ส่งจากภายนอกฟังก์ชันเข้ามาประมวลผลภายในฟังก์ชัน

5.1 ฟังก์ชันมาตรฐาน

ฟังก์ชันมาตรฐาน คือ ฟังก์ชันที่มาพร้อมกับภาษา PHP สามารถเรียกใช้งานได้ทันที ฟังก์ชันมาตรฐานมีหลายกลุ่มการทำงาน (ในส่วนของฟังก์ชันมาตรฐานจะกล่าวถึงรูปแบบ และตัวอย่างการใช้งานในบทถัดไป) สามารถจำแนกตามหน้าที่ ดังนี้ 1) ฟังก์ชันที่เกี่ยวกับวันที่และเวลา 2) ฟังก์ชันที่เกี่ยวกับการคำนวณทางคณิตศาสตร์ 3) ฟังก์ชันที่เกี่ยวกับการติดต่อกับฐานข้อมูล และ 4) ฟังก์ชันที่เกี่ยวกับการจัดการข้อความ

การเรียกใช้งานฟังก์ชันมาตรฐานภาษา PHP จะต้องตรวจสอบก่อนว่าฟังก์ชันนั้นๆ เป็นฟังก์ชันเพื่อทำหน้าที่อะไร มีการส่งผ่านค่าพารามิเตอร์หรือไม่ ถ้าไม่มีการส่งผ่านค่าพารามิเตอร์ ก็สามารถเรียกใช้งานได้เลย แต่ถ้ามีการส่งผ่านค่าพารามิเตอร์ ก็จำเป็นต้องระบุค่าพารามิเตอร์ ให้ถูกต้องตามรูปแบบที่ฟังก์ชันกำหนด

ตัวอย่างที่ 5.1 รูปแบบฟังก์ชันที่ไม่มีการส่งผ่านค่าพารามิเตอร์

```
ชื่อฟังก์ชัน ( );
```

ตัวอย่างที่ 5.2 รูปแบบฟังก์ชันที่มีการส่งผ่านค่าพารามิเตอร์

```
ชื่อฟังก์ชัน (พารามิเตอร์ 1, พารามิเตอร์ 2, ...);
```

ตัวอย่างที่ 5.3 การเรียกใช้งานฟังก์ชันมาตรฐาน ฟังก์ชัน date ( )

```
1    <?php
2        $today = date("d/m/Y");
3        echo $today;
4    ?>
```

ผลลัพธ์

```
6/7/2012
```

จากตัวอย่างที่ 5.3 การเรียกใช้งานฟังก์ชันมาตรฐาน ฟังก์ชัน date ( ) อธิบายดังนี้

บรรทัดที่ 1 เริ่มต้นสคริปต์ PHP

บรรทัดที่ 2 กำหนดให้ตัวแปร $today มีค่าเท่ากับ ผลของการเรียกใช้ฟังก์ชัน date ( ) โดยกำหนดพารามิเตอร์ ประกอบด้วย d/m/Y คือ วัน เดือน และปี ค.ศ.

บรรทัดที่ 3 แสดงค่าที่เก็บในตัวแปร $today

บรรทัดที่ 4 สิ้นสุดสคริปต์ PHP

5.2 ฟังก์ชันที่ผู้ใช้งานเป็นผู้สร้างเอง

ฟังก์ชันที่ผู้ใช้งานเป็นผู้สร้างเอง คือ กลุ่มของคำสั่งที่ผู้ใช้งานเป็นผู้เขียนหรือพัฒนาขึ้นมาเองเพื่อทำงานหรือทำหน้าที่อย่างใดอย่างหนึ่งตามต้องการ

5.2.1 การสร้างฟังก์ชันที่ผู้ใช้งานเป็นผู้สร้างเอง

ลักษณะของงานที่จะนำมาสร้างเป็นฟังก์ชันนั้น ควรเป็นงานหรือการกระทำอย่างใดอย่างหนึ่ง ที่จำเป็นต้องทำซ้ำๆ และบ่อยครั้ง เพื่อลดเวลาในการเขียนคำสั่งหรือชุดคำสั่งเมื่อต้องการทำงานแบบเดิม โดยสามารถแยกคำสั่งบางส่วนออกมาสร้างเป็นฟังก์ชันไว้เฉพาะ และเรียกใช้ตามลักษณะงานที่ต้องการ จะช่วยให้จำนวนบรรทัดคำสั่งน้อยลง (ได้ผลลัพธ์เหมือนเดิม) ช่วยลดการใช้ทรัพยากร และง่ายในการปรับปรุงแก้ไข เพราะสามารถแก้ไขเพียงครั้งเดียวก็จะมีผลทุกจุดภายในโปรแกรมที่เรียกใช้ฟังก์ชัน

รูปแบบ

```
<?php
        function ชื่อฟังก์ชัน (พารามิเตอร์) {
```

```
                คำสั่ง 1;
                คำสั่ง 2;
                ...
            }
    ?>
```

5.2.2 ข้อกำหนดหลักเกณฑ์การตั้งชื่อฟังก์ชัน มีหลักเกณฑ์คล้ายกับการตั้งชื่อตัวแปร ดังนี้

1) ต้องขึ้นต้นชื่อด้วย a-z หรือ _ เท่านั้น
2) ต้องประกอบด้วย a-z, 0-9 หรือ _ เท่านั้น
3) ต้องไม่ซ้ำกับชื่อฟังก์ชันที่มีอยู่แล้วหรือฟังก์ชันมาตรฐานของภาษา PHP

5.2.3 การเรียกใช้ฟังก์ชันที่ผู้ใช้งานเป็นผู้สร้างเอง สามารถทำได้เช่นเดียวกับการใช้ฟังก์ชันมาตรฐานของภาษา PHP คือ ต้องระบุชื่อฟังก์ชันที่ต้องการใช้งาน ระบุค่าพารามิเตอร์ (ถ้ามี) ตัวอย่างดังนี้

ตัวอย่างที่ 5.4 การสร้างฟังก์ชัน และเรียกใช้งานฟังก์ชันที่ผู้ใช้งานเป็นผู้สร้างเอง

```
1   <?php
2       function multiplication ($number, $start, $end) {
3           line_ (40, "*");
4           echo "Multiplication of $number : Starting at $start until $end <br>";
5           line_ (40, "*");
6           for ($start=1; $start <= $end; $start++){
7               printf ("%d x %d = %d <br>", $number, $start, $number*$start);
8           }
9           line_ (40, "*");
10      }
11      function line_ ($how_much, $symbol) {
12          for ($begin=1; $begin <= $how_much; $begin++) {
13              echo $symbol;
14          }
15          echo "<br>";
16      }
17      multiplication (12, 1, 5);
18  ?>
```

ผลลัพธ์

```
****************************************
Multiplication of 12 : Starting at 1 until 5
****************************************
12 x 1 = 12
12 x 2 = 24
12 x 3 = 36
12 x 4 = 48
12 x 5 = 60
****************************************
```

จากตัวอย่างที่ 5.4 การสร้างฟังก์ชัน และเรียกใช้งานฟังก์ชันที่ผู้ใช้งานเป็นผู้สร้างเอง ในสคริปต์ประกอบด้วย 2 ฟังก์ชันที่ผู้ใช้งานเป็นผู้สร้างเอง อธิบายดังนี้

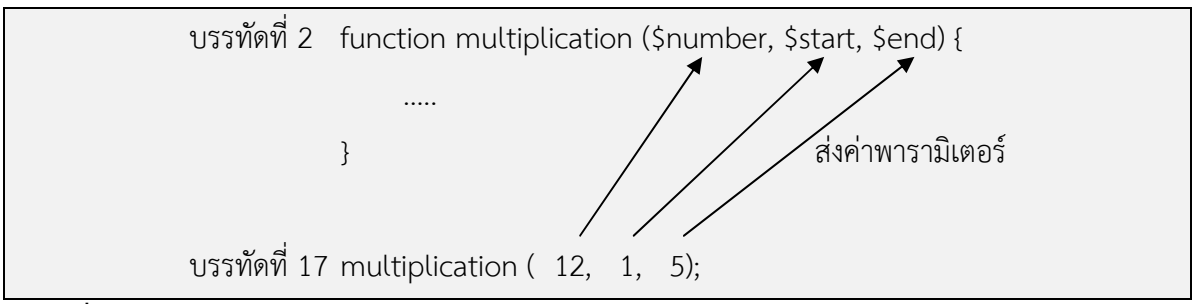
ฟังก์ชัน multiplication ( ) ใช้สำหรับหาค่าผลคูณของเลขตามที่ระบุ (อยู่ในช่วงบรรทัดที่ 2 ถึงบรรทัดที่ 10) พารามิเตอร์ ประกอบด้วย $number หมายถึง แม่เลขที่ต้องการแสดง $start หมายถึง ลำดับเลขเริ่มต้นของการคูณ และ $end หมายถึง ลำดับเลขสุดท้ายของการคูณ

ฟังก์ชัน line_ ( ) ใช้สำหรับแสดงสัญลักษณ์และจำนวนสัญลักษณ์ตามที่ระบุ (อยู่ในช่วงบรรทัดที่ 11 ถึงบรรทัดที่ 16) พารามิเตอร์ ประกอบด้วย $how_much หมายถึง จำนวนของ สัญลักษณ์ที่ต้องการแสดงต่อแถว และ $symbol หมายถึง สัญลักษณ์ที่ต้องการแสดง

จุดเริ่มต้นของคำสั่ง

บรรทัดที่ 17 ในช่วงเริ่มต้นนั้น ฟังก์ชันจะยังไม่ทำงานหากไม่มีการเรียกใช้งานฟังก์ชัน ดังนั้น เมื่อระบุ multiplication (12, 1, 5) จึงหมายความว่า แม่เลขที่ต้องการแสดงผล คือ 12 ลำดับเลขเริ่มต้นของการคูณ คือ เลข 1 และลำดับเลขสุดท้ายของการคูณ คือ เลข 5 ลำดับถัดไปที่จะเริ่มทำงาน คือ บรรทัดที่ 2

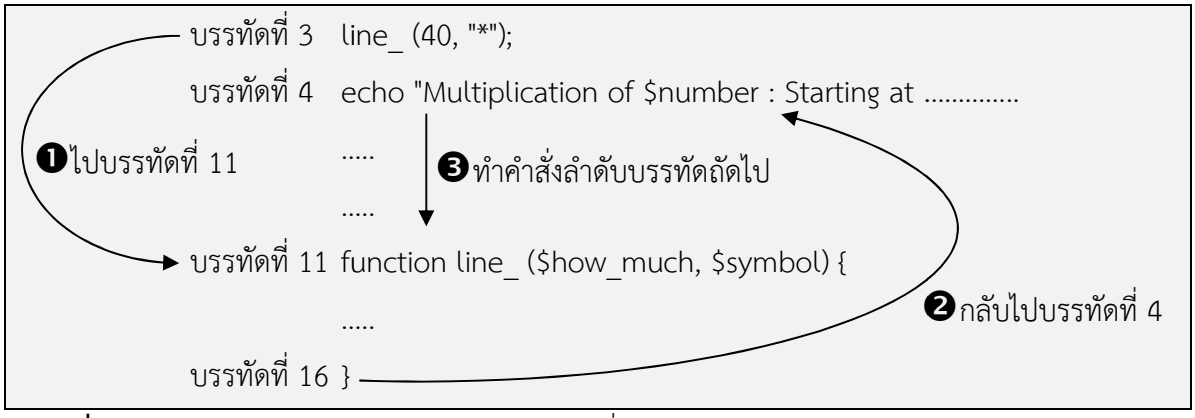
บรรทัดที่ 2 อธิบายเพิ่มเติมรูปแบบการส่งผ่านค่าพารามิเตอร์ไปยังฟังก์ชัน ดังภาพที่ 5.1



ภาพที่ 5.1 แสดงการส่งผ่านค่าพารามิเตอร์เข้าไปประมวลผลภายในฟังก์ชัน

จากภาพที่ 5.1 การส่งผ่านค่าพารามิเตอร์ ไปยังฟังก์ชันนั้น จะเรียงตามลำดับตัวแปรพารามิเตอร์ หมายความว่า ค่าเลข 12 จะถูกส่งไปยังตัวแปร $number ค่าเลข 1 ถูกส่งไปยังตัวแปร $start และ ค่าเลข 5 ถูกส่งไปยังตัวแปร $end หลังจากนั้นฟังก์ชันจะนำตัวแปรทั้ง 3 เข้าไปประมวลผลภายในฟังก์ชันตามลำดับคำสั่งร่วมกับโครงสร้างเงื่อนไขทำซ้ำ for ภายในฟังก์ชัน (โครงสร้างเงื่อนไขทำซ้ำได้กล่าวและอธิบายแล้วในบทที่ 4)

ภายในฟังก์ชัน multiplication ( ) มีการเรียกฟังก์ชัน line_ ( ) บรรทัดที่ 3 และบรรทัดอื่นๆ อธิบายลำดับการทำงาน ด้วยภาพที่ 5.2 ดังนี้



ภาพที่ 5.2 แสดงการเรียกใช้ฟังก์ชันภายนอกฟังก์ชันที่กำลังทำงานในปัจจุบัน

จากภาพที่ 5.2 บรรทัดที่ 3 line_ ( ) (เรียกใช้ฟังก์ชัน line ( ) ภายในฟังก์ชัน multiplication ( )) ดังนั้นลำดับบรรทัดถัดไปที่จะทำงาน คือ บรรทัดที่ 11 ถึงบรรทัดที่ 16 โดยค่าเลข 40 เป็นพารามิเตอร์ส่งไปให้ตัวแปร $how_much และสัญลักษณ์ "*" ถูกส่งไปให้กับตัวแปร $symbol ตามลำดับ แล้วประมวลผลตามลำดับคำสั่งร่วมกับโครงสร้างเงื่อนไขทำซ้ำ for และเมื่อทำงานตามคำสั่งถึงบรรทัดที่ 16 จะกลับไปทำคำสั่งในบรรทัดที่ 4

5.3 ฟังก์ชันพารามิเตอร์

ฟังก์ชันพารามิเตอร์ คือ ค่าที่จะนำไปใช้หรือประมวลผลภายในฟังก์ชัน ซึ่งจะทำให้ฟังก์ชันมีความยืดหยุ่นต่อการใช้งาน ดังตัวอย่างที่ 5.4 เพราะผลลัพธ์จะแปรเปลี่ยนไปตามค่าพารามิเตอร์ที่ได้รับ รูปแบบการกำหนดและส่งผ่านค่าพารามิเตอร์ มีรายละเอียดดังนี้

5.3.1 พารามิเตอร์แบบกำหนดค่าเริ่มต้น (Default Parameter)

พารามิเตอร์แบบกำหนดค่าเริ่มต้น ในบางฟังก์ชันอาจใช้ค่าพารามิเตอร์ค่าใดค่าหนึ่งเป็นค่าเริ่มต้น อาจจะมีการเปลี่ยนไปใช้ค่าอื่นบ้างในบางครั้ง ดังนั้นเพื่อความสะดวกจึงมีการกำหนดค่าพารามิเตอร์แบบกำหนดค่าเริ่มต้นขึ้น โดยจะกำหนดค่าพารามิเตอร์ที่ต้องใช้บ่อยๆ ไว้ล่วงหน้า หรือป้องกันปัญหาในกรณีที่ไม่ได้กำหนดค่าพารามิเตอร์ให้กับฟังก์ชัน เมื่อมีการเรียกใช้ฟังก์ชัน ซึ่งในกรณีที่ไม่ส่งค่าพารามิเตอร์มาให้ฟังก์ชัน ฟังก์ชันจะเรียกใช้ค่าเริ่มต้นที่กำหนดไว้ให้แทน รูปแบบการกำหนดพารามิเตอร์แบบกำหนดค่าเริ่มต้น มีรายละเอียด ดังนี้

รูปแบบ

```
<?php
        function ชื่อฟังก์ชัน (ชื่อพารามิเตอร์ = ค่าเริ่มต้น) {
                คำสั่ง;
        }
?>
```

ตัวอย่างที่ 5.5 ฟังก์ชันปิรามิดตัวเลข

```
<?php
        function pyramidNumber ($number=7) {
                for ($loop1=$number; $loop1>=1; $loop1--) {
                        for ($loop2=1; $loop2<=$loop1; $loop2++) {
                                printf (" %d ",$loop2);
                        }
                        echo "<br>";
                }
        }
?>
```

ตัวอย่างที่ 5.6 การเรียกใช้ฟังก์ชันปิรามิดตัวเลข

แบบไม่ส่งผ่านค่าพารามิเตอร์	**แบบส่งผ่านค่าพารามิเตอร์**
<?php pyramidNumber ( ); ?>	<?php pyramidNumber (4); ?>

ผลลัพธ์

1 2 3 4 5 6 7 1 2 3 4 5 6 1 2 3 4 5 1 2 3 4 1 2 3 1 2 1	1 2 3 4 1 2 3 1 2 1

จากตัวอย่างที่ 5.6 การเรียกใช้ฟังก์ชันปิรามิดตัวเลข คือ ฟังก์ชัน pyramidNumber ( ) จากตัวอย่างที่ 5.5 จะเห็นได้ว่าหากไม่ส่งค่าพารามิเตอร์ไปยังฟังก์ชัน pyramidNumber ( ) ฟังก์ชันจะ

กำหนดให้ตัวแปร $number มีค่าเท่ากับ 7 แล้วเริ่มทำงานภายในฟังก์ชัน ในกรณีเรียกใช้ฟังก์ชันโดยกำหนดค่าพารามิเตอร์เท่ากับ 4 ฟังก์ชัน pyramidNumber ( ) ก็จะกำหนดให้ตัวแปร $number มีค่าเท่ากับ 4 ตามค่าพารามิเตอร์ส่งค่าให้ฟังก์ชัน เป็นต้น

5.3.2 พารามิเตอร์แบบส่งผ่านค่าโดยการอ้างอิง (Passing Parameter by Value and Reference)

โดยปกติแล้วค่าที่ถูกส่งไปยังฟังก์ชันจะเป็นแบบส่งค่า (by Value) คือ เมื่อมีการเปลี่ยนแปลงค่าของตัวแปรภายในฟังก์ชันจะไม่ส่งต่อค่าตัวแปรตัวเดียวกันที่อยู่นอกฟังก์ชัน ทุกตัวอย่างก่อนหน้าในเรื่องฟังก์ชันนี้ใช้วิธีการผ่านค่าพารามิเตอร์แบบส่งค่าทั้งหมด

การส่งผ่านค่าแบบอ้างอิง (by Reference) นั้น หากในฟังก์ชันมีการเปลี่ยนแปลงค่าของตัวแปรภายในฟังก์ชันจะส่งผลให้ค่าของตัวแปรที่อ้างอิงกันนอกฟังก์ชันมีค่าเปลี่ยนตามไปด้วย การส่งผ่านค่าแบบอ้างอิงสามารถทำได้โดยการใส่เครื่องหมาย & ไว้หน้าพารามิเตอร์ตัวที่ต้องการอ้างอิง

ตัวอย่างที่ 5.7 การเปรียบเทียบการส่งผ่านค่าพารามิเตอร์แบบปกติและแบบอ้างอิง

ก. การส่งผ่านค่าพารามิเตอร์แบบปกติ

```
1   <?php
2       function foo ($var) {
3           $var++;
4       }
5       $a=5;
6       foo ($a);
7       echo $a;
8   ?>
```

ผลลัพธ์ตัวแปร $a มีค่าเท่ากับ 5

ข. การส่งผ่านค่าพารามิเตอร์แบบอ้างอิง

```
1   <?php
2       function foo (&$var) {
3           $var++;
4       }
5       $a=5;
6       foo ($a);
7       echo $a;
8   ?>
```

ผลลัพธ์ตัวแปร $a มีค่าเท่ากับ 6

จากตัวอย่างที่ 5.7 การเปรียบเทียบการส่งผ่านค่าพารามิเตอร์แบบปกติและแบบอ้างอิงนั้น จริงๆ แล้วก็คือ ขอบเขตของตัวแปรนั่นเอง (ศึกษาเพิ่มเติมในบทที่ 3) อธิบายดังนี้

ตัวอย่าง 5.7 ก. การส่งผ่านค่าพารามิเตอร์แบบส่งผ่านค่าแบบปกติ ค่าพารามิเตอร์ที่ส่งไปยังฟังก์ชัน foo ( ) ในบรรทัดที่ 6 มีค่าเท่ากับ 5 (ตัวแปร $a มีค่าเท่ากับ 5) ภายในฟังก์ชัน foo ( ) มีการนำค่าพารามิเตอร์ที่รับเข้ามาผ่านตัวแปร $var บรรทัดที่ 3 มีการเพิ่มค่าให้กับตัวแปร $var แล้วกลับมาบรรทัดที่ 7 เพื่อแสดงผลค่าตัวแปร $a ค่าของตัวแปร $a ที่แสดง คือ 5 เป็นผลมาจากบรรทัดที่ 5 ซึ่งถือว่าบรรทัดที่ 3 ไม่สามารถเปลี่ยนแปลงค่าตัวแปร $a ในบรรทัดที่ 5 ได้

ตัวอย่าง 5.7 ข. การส่งผ่านค่าพารามิเตอร์แบบอ้างอิง ค่าพารามิเตอร์ที่ส่งไปยังฟังก์ชัน foo ( ) ในบรรทัดที่ 6 มีค่าเท่ากับ 5 (ตัวแปร $a มีค่าเท่ากับ 5) และฟังก์ชันมีการกำหนดแบบอ้างอิง คือ &$var หมายความว่า กำหนดให้ตัวแปร $var อ้างอิงไปยัง $a (กำหนดให้เป็นตัวแปรอ้างอิง ดังนั้นหากตัวแปรตัวหนึ่งตัวใดมีการเปลี่ยนแปลงค่า จะยังผลให้ตัวแปรที่อ้างอิงกันมีค่าเปลี่ยนแปลงตามไปด้วย)

บรรทัดที่ 3 มีการเพิ่มค่าให้กับตัวแปร $var (เพิ่มค่าให้กับตัวแปร $a ด้วย) แล้วกลับมาบรรทัดที่ 7 เพื่อแสดงผลค่าตัวแปร $a ดังนั้นค่าของตัวแปร $a ที่แสดง จึงมีค่าเท่ากับ 6

5.4 การส่งค่ากลับจากฟังก์ชันด้วยคำสั่ง return

เนื่องจากฟังก์ชันจะใช้ในการประมวลผลอย่างใดอย่างหนึ่ง โดยฟังก์ชันมักจะถูกเรียกใช้โดยส่วนต่างๆ ของโปรแกรม เพื่อประมวลผลตามหน้าที่ต่างๆ ของฟังก์ชัน ในบางครั้งฟังก์ชันอาจจำเป็นต้องส่งค่าผลลัพธ์ของการทำงานกลับไปยังส่วนที่เรียกใช้ฟังก์ชันนั้นๆ หรือสามารถประยุกต์ใช้สำหรับการตรวจสอบการทำงานของฟังก์ชัน เช่น ทำงานปกติอาจส่งค่ากลับเป็นเลข 1 ทำงานไม่ถูกต้องส่งค่ากลับเป็นเลข 2 หรืออื่นๆ ตามต้องการ เป็นต้น สำหรับวิธีการส่งค่ากลับออกไปจะใช้คำสั่ง return แล้วตามด้วยค่าที่ต้องการส่งออกไป มีรูปแบบ ดังนี้

```
return ค่าที่จะส่งกลับ;
```

5.4.1 การส่งค่ากลับหนึ่งค่า (Returning Single Value)

กำหนดให้เมื่อฟังก์ชันทำงานเสร็จมีการส่งผลลัพธ์ที่ได้จากการประมวลผล มากำหนดค่าให้กับตัวแปร หรือแสดงผล หรืออื่นๆ ค่าที่ส่งกลับจะมีเพียง 1 ค่า ตัวอย่างดังนี้

ตัวอย่างที่ 5.8 ฟังก์ชัน calcSalesTax ( ) มีการส่งค่ากลับเป็นค่าของตัวแปร $total

```
1   <?php
2       function calcSalesTax ($price, $tax=.0675) {
3           $total = $price + ($price * $tax);
4           return $total;
5       }
6   ?>
```

จากตัวอย่างที่ 5.8 แสดงตัวอย่างการส่งค่ากลับจากฟังก์ชัน calcSalesTax ( ) ผ่านตัวแปร $total โดยค่าของตัวแปร $total มาจากนิพจน์ในบรรทัดที่ 3 แล้วฟังก์ชันส่งค่ากลับเป็นค่าของตัวแปร $total ไปยังตำแหน่งที่เรียกใช้ฟังก์ชัน

ตัวอย่างที่ 5.9 ฟังก์ชัน calcSalesTax ( ) มีการส่งค่ากลับเป็นผลลัพธ์ของการคำนวณ

```
1   <?php
2       function calcSalesTax ($price, $tax=.0675) {
3           return $price + ($price * $tax);
4       }
5   ?>
```

จากตัวอย่างที่ 5.9 แสดงตัวอย่างการส่งค่ากลับจากฟังก์ชัน calcSalesTax ( ) โดยใช้ผลลัพธ์จากนิพจน์ส่งค่ากลับไปยังตำแหน่งที่เรียกใช้ฟังก์ชัน

ตัวอย่างที่ 5.10 ตัวอย่างการเรียกใช้ฟังก์ชัน calcSalesTax ( )

```
1   <?php
2       echo calcSalesTax (500, $tax=.0675);
        // หรือใช้ตัวแปรรับผลการ return ของฟังก์ชัน ดังนี้
        // $value = calcSalesTax (500, $tax=.0675); แล้วนำตัวแปร $value ไปแสดงผล
3   ?>
```

ผลลัพธ์

```
533.75
```

จากตัวอย่างที่ 5.10 แสดงตัวอย่างการเรียกใช้ฟังก์ชัน calcSalesTax ( ) จากตำแหน่งใดๆ ภายในเว็บเพจเดียวกัน ในบรรทัดที่ 2 เป็นการแสดงผลลัพธ์โดยตรงจากค่าที่ส่งกลับจากฟังก์ชัน หรือสามารถใช้ตัวแปรรับผลค่าจากฟังก์ชันก่อน แล้วจึงนำตัวแปรที่รับผลค่าไปแสดงผล ดังนี้

ตัวอย่างที่ 5.11 ตัวอย่างการเรียกใช้ตัวแปรรับผลค่าจากฟังก์ชัน calcSalesTax ( )

```
1   <?php
2       $value = calcSalesTax (500, $tax=.0675);
3       echo $value;
4   ?>
```

5.4.2 การส่งค่ากลับแบบหลายค่า (Returning Multiple Values)

ในบางครั้งของการเรียกใช้ฟังก์ชัน อาจจำเป็นต้องต้องส่งค่ากลับมากกว่า 1 ค่า เช่น เขียนฟังก์ชันเพื่อติดต่อกับฐานข้อมูล แล้วต้องการส่งค่ากลับประกอบด้วย ชื่อ-สกุล อีเมล์ ที่อยู่ เบอร์โทรศัพท์ เป็นต้น ซึ่งในตัวอย่างก่อนหน้านี้จะกล่าวถึงเฉพาะส่งค่ากลับเพียง 1 ค่าเท่านั้น ในกรณีนี้มีฟังก์ชันมาตรฐานของภาษา PHP ที่สามารถช่วยได้ คือ ฟังก์ชัน list ( ) ใช้สำหรับส่งค่ากลับมากกว่า 1 ค่าในรูปแบบของอาร์เรย์ ตัวอย่างดังนี้

ตัวอย่างที่ 5.12 การส่งค่ากลับแบบหลายค่า

```
1   <?php
2       function retrieveUserProfile ( ) {
3           $user [ ] = "Parinya";
4           $user [ ] = "parinya@example.com";
5           $user [ ] = "Thai";
6           return $user;
```

```
7               }
8               list ($name, $email, $language) = retrieveUserProfile ( );
9               echo "Name: $name, email: $email, language: $language";
10      ?>
```

ผลลัพธ์

```
Name: Parinya, email: parinya@example.com, language: Thai
```

จากตัวอย่างที่ 5.11 การส่งค่ากลับแบบหลายค่า เป็นการประยุกต์ใช้ข้อมูลอาร์เรย์สำหรับส่งค่ากลับจากฟังก์ชัน (โดยปกติฟังก์ชันจะส่งค่ากลับได้เพียง 1 ค่าเท่านั้น) ในบรรทัดที่ 8 ผลของฟังก์ชัน retrieveUserProfile ( ) จะส่งค่ากลับเป็นข้อมูลชนิดอาร์เรย์ ให้กับฟังก์ชัน list ( ) เพื่อแยกข้อมูลอาร์เรย์ออกเป็นตัวแปรเดี่ยวแล้วกำหนดค่าให้กับตัวแปรตามลำดับ ประกอบด้วยตัวแปร $name, $email และ $language แล้วนำตัวแปรทั้งหมดไปแสดงผลในบรรทัดที่ 9

5.5 ฟังก์ชันแบบเรียกตัวเอง

ฟังก์ชันแบบเรียกตัวเอง (Recursive Functions) คือ ฟังก์ชันที่มีการเรียกใช้ฟังก์ชันตัวเอง ใช้สำหรับทำงานซ้ำๆ กันโดยมีจุดสิ้นสุดการทำงานอยู่ในฟังก์ชัน เพื่อให้ฟังก์ชันหยุดการทำงาน การเขียนฟังก์ชันแบบเรียกตัวเอง จะช่วยลดระยะเวลาในการเขียนสคริปต์ องค์ประกอบที่สำคัญของฟังก์ชันแบบเรียกตัวเอง มีรายละเอียดดังนี้

5.5.1 มีการเรียกใช้ฟังก์ชันซ้อนฟังก์ชันตัวเองภายในฟังก์ชัน อาจจะอยู่ในส่วนคำสั่งหรือเป็นองค์ประกอบในนิพจน์

5.5.2 มีการรับค่าผ่านพารามิเตอร์ของฟังก์ชัน โดยค่าดังกล่าวจะถูกนำไปใช้ภายในฟังก์ชันเรียกตัวเอง ผลของการคำนวณบางส่วนจะถูกนำกลับไปใช้เมื่อมีการเรียกใช้ฟังก์ชันแบบเรียกตัวเอง

5.5.3 มีการส่งผลการคำนวณกลับ โดยผลดังกล่าวจะนำไปใช้ในการคำนวณในฟังก์ชันแบบเรียกตัวเองหรืออาจจะส่งผลกลับไปให้กับส่วนที่เรียกใช้ฟังก์ชันในครั้งแรก

5.5.4 มีจุดสิ้นสุดของการเรียกใช้ฟังก์ชันตัวเอง ฟังก์ชันแบบเรียกตัวเองจะทำงานไม่ได้หากไม่มีช่องทางให้สคริปต์หยุดการเรียกตัวเองในขั้นใดขั้นหนึ่ง เพราะจะเกิดการเรียกฟังก์ชันแบบเรียกตัวเองไม่มีสิ้นสุด

การเขียนฟังก์ชันแบบเรียกตัวเองช่วยแก้ปัญหาที่ซับซ้อนได้โดยการเขียนฟังก์ชันในรูปแบบที่ง่ายและสั้น ตัวอย่างดังนี้

ตัวอย่างที่ 5.13 การสร้างและการเรียกใช้ฟังก์ชันแบบเรียกตัวเอง สำหรับหาค่าแฟกทอเรียล n!

```
1       <?php
2               function factorial ($n) {
```

```
3                       if ($n < 1)
4                               return 1;
5                       else
6                               return $n*factorial($n-1);
7               }
8               echo factorial (4);
9       ?>
```

ผลลัพธ์

```
24
```

จากตัวอย่างที่ 5.13 การสร้างและการเรียกใช้ฟังก์ชันแบบเรียกตัวเอง สำหรับหาค่าแฟกทอเรียล n! โดยหลักแนวคิด และการทำงานของฟังก์ชัน มีดังนี้

factorial ($n) = $n*($n-1)*($n-2)*($n-3)

ดังนั้น 4! = 4*3*2*1

= 24

จะเห็นได้ว่าตัวแปร $n จะต้องเป็นตัวแปรที่เก็บค่าที่เป็นเลขจำนวนเต็ม และไม่เป็นลบ ถ้าต้องการจะเขียนฟังก์ชันให้มีความปลอดภัยในการใช้งาน ก็อาจจะเพิ่มเงื่อนไข เพื่อตรวจสอบก่อนว่า ผู้ใช้ส่งค่าของตัวแปรตรงตามต้องการเงื่อนไขหรือไม่ เช่น ไม่ส่งค่าที่เป็นข้อความ หรือเป็นเลขทศนิยม หรือค่าที่เป็นลบ เป็นต้น จากตัวอย่างจะเป็นการทำงานซ้ำโดยใช้รูปแบบฟังก์ชันเรียกใช้ตัวเอง จะหยุดก็ต่อเมื่อตัวแปร $n มีค่าน้อยกว่าหรือเท่ากับ 1 เมื่อเป็นจริงตามเงื่อนไขฟังก์ชันจะหยุดทำงานโดยใช้คำสั่ง return 1 หรือ return true

ตัวอย่างที่ 5.14 การสร้างและการเรียกใช้ฟังก์ชันแบบเรียกตัวเอง สำหรับหาค่าผลรวม n จำนวน

```
1       <?php
2               function totalnum ($n) {
3                       if ($n <=1 ) {
4                               return 1;
5                       }
6                       $total=totalnum($n-1)+$n;
7                       return $total;
8               }
9               echo totalnum (20);
10      ?>
```

ผลลัพธ์

```
210
```

จากตัวอย่างที่ 5.14 ฟังก์ชัน totalnum ( ) ใช้สำหรับหาค่าผลรวมของ n จำนวน โดยหลักแนวคิด และการทำงานของฟังก์ชัน มีดังนี้

totalnum ($n) = ($n-19)+($n-18)+($n-17)+($n-16)+($n-15)+($n-14)+($n-13)+
($n-12)+($n-11)+($n-10)+($n-9)+($n-8)+($n-7)+($n-6)+($n-5)+
($n-4)+($n-3)+($n-2)+($n-1)+20

ดังนั้น = 1+2+3+4+5+6+7+8+9+10+11+12+13+14+15+16+17+18+19+20

= 210

จากตัวอย่างจะเป็นการทำงานซ้ำโดยใช้รูปแบบฟังก์ชันเรียกใช้ตัวเอง จะหยุดก็ต่อเมื่อตัวแปร $n มีค่าน้อยกว่าหรือเท่ากับ 1 เมื่อเป็นจริงตามเงื่อนไขฟังก์ชันจะหยุดทำงานโดยใช้คำสั่ง return 1 หรือ return true

5.6 ฟังก์ชันไลบรารี

ฟังก์ชันไลบรารี (Function Libraries) เป็นวิธีที่จะช่วยลดการเขียนคำสั่งหรือฟังก์ชันซ้ำซ้อน คือ การเขียนคำสั่งการทำงานแยกเก็บไว้อีกไฟล์หนึ่ง ซึ่งอาจเป็นชุดคำสั่ง ฟังก์ชัน คลาส หรืออื่นๆ ที่เกี่ยวข้อง ฟังก์ชันไลบรารีหรืออาจจะเรียกว่าคลังเก็บฟังก์ชันก็ว่าได้ และเมื่อต้องการนำไฟล์ดังกล่าวมาใช้งานที่เว็บเพจใด ก็ให้นำเข้าไฟล์ที่เก็บฟังก์ชันนั้นๆ หลังจากนั้นจะสามารถใช้งานชุดคำสั่ง ฟังก์ชัน คลาส หรืออื่นๆ ได้ เสมือนว่าฟังก์ชันที่จะเรียกใช้นั้นอยู่ในเว็บเพจเดียวกัน ฟังก์ชันสำหรับการเรียกใช้หรือนำเข้าไฟล์จากภายนอก ประกอบด้วย 1) include ( ) 2) include_once ( ) 3) require ( ) และ 4) require_once ( ) มีรายละเอียดการใช้งานในแต่ละฟังก์ชัน ดังนี้

5.6.1 ฟังก์ชัน include ( )

เป็นฟังก์ชันที่ใช้สำหรับนำเข้าไฟล์จากภายนอก เพื่อนำเข้ามาใช้งานร่วมกับคำสั่งอื่นๆ ภายในหน้าเว็บเพจ ในกรณีฟังก์ชัน include ( ) เรียกหาไฟล์ที่ระบุไม่พบ ระบบจะแจ้งเตือน (Warning) ข้อผิดพลาดแล้วข้ามไปทำส่วนอื่นๆ ของคำสั่งในลำดับถัดไป

5.6.2 ฟังก์ชัน include_once ( )

เป็นฟังก์ชันที่ใช้สำหรับนำเข้าไฟล์จากภายนอก เหมือนกับฟังก์ชัน include ( ) แต่ต่างกันที่ฟังก์ชัน include_once ( ) จะนำเข้าไฟล์จากภายนอกนั้นได้เพียงครั้งเดียวเท่านั้น

ตัวอย่างที่ 5.15 ตัวอย่างไฟล์ที่เก็บชุดคำสั่งกำหนดชื่อเป็น lib.php

```
<?php
        echo "parinya<br>";
?>
```

ตัวอย่างที่ 5.16 การเปรียบเทียบความแตกต่างระหว่างฟังก์ชัน include ( ) และ include_once ( )

```
1   <?php
2       include ("lib.php");
3       include ("lib.php");
4       include ("lib.php");
5   ?>
```

```
1   <?php
2       include_once ("lib.php");
3       include_once ("lib.php");
4       include_once ("lib.php");
5   ?>
```

ผลลัพธ์

ผลลัพธ์ของการใช้ฟังก์ชัน include ( );

parinya

parinya

parinya

ผลลัพธ์ของการใช้ฟังก์ชัน include_once ( );

parinya

ในกรณีที่เรียกใช้ฟังก์ชัน include ( ) หรือ include_once ( ) แล้วไม่พบไฟล์ที่จะนำเข้า จะปรากฏข้อความตามตัวอย่าง แต่ยังสามารถทำงานได้ในลำดับบรรทัดถัดไป ตัวอย่างดังต่อไปนี้

ตัวอย่างที่ 5.17 แสดงตัวอย่างข้อผิดพลาดเมื่อไม่พบไฟล์ที่จะนำเข้า

```
<?php
        include ("file.php");
        echo "ยังสามารถแสดงผลบรรทัดนี้ได้<br>";
?>
```

ผลลัพธ์

```
Warning: include(file.php) [function.include]: failed to open stream: No such file or directory in C:\AppServ\www\ test.php on line 2
Warning: include() [function.include]: Failed opening 'file.php' for inclusion (include_path='.;C:\php5\pear') in C:\AppServ\www\ test.php on line 2
ยังสามารถแสดงผลบรรทัดนี้ได้
```

5.6.3 ฟังก์ชัน require ( )

เป็นฟังก์ชันที่ใช้สำหรับนำเข้าไฟล์จากภายนอกเหมือนกับฟังก์ชัน include ( ) แต่ต่างกันที่ฟังก์ชัน require ( ) เมื่อไม่พบไฟล์ที่ที่จะนำเข้าจะแจ้งเตือน และแสดงข้อผิดพลาด (Fatal Error) จากนั้นจะหยุดการทำงานไม่ทำคำสั่งในลำดับบรรทัดถัดไป นิยมใช้นำเข้าไฟล์จากภายนอกที่จัดเก็บฟังก์ชันหลักหรือคลาสที่มีความจำเป็นต่อการทำงานของโปรแกรม

5.6.4 ฟังก์ชัน require_once ( )

เป็นฟังก์ชันที่มีการทำงานเหมือนกับฟังก์ชัน require ( ) แต่ต่างกันที่ฟังก์ชัน require_once ( ) จะนำเข้าไฟล์จากภายนอกนั้นได้เพียงครั้งเดียว

ในกรณีที่เรียกใช้ฟังก์ชัน require ( ) หรือ require_once ( ) แล้วไม่พบไฟล์ที่จะนำเข้า จะปรากฏข้อความตามตัวอย่าง และจะไม่ทำงานในลำดับบรรทัดต่อไป ตัวอย่างดังนี้

ตัวอย่างที่ 5.18 การใช้ฟังก์ชัน require_once ( )

```
1    <?php
2        require ("file.php");
3        echo "ยังสามารถแสดงผลบรรทัดนี้ได้<br>";
4    ?>
```

ผลลัพธ์

```
Warning: require(file.php) [function.require]: failed to open stream: No such file or directory in C:\AppServ\www\ test.php on line 2
Fatal error: require() [function.require]: Failed opening required 'file.php' (include_path='.;C:\php5\pear') in C:\AppServ\www\test.php on line 2
```

จากตัวอย่างดังกล่าว จะเห็นได้ว่าไม่ปรากฏคำว่า "ยังสามารถแสดงผลบรรทัดนี้ได้" เรียกลักษณะดังกล่าวว่า Fatal Error นิยมใช้นำเข้าไฟล์จากภายนอกที่จัดเก็บฟังก์ชันหลัก หรือคลาสที่มีความจำเป็นต่อการทำงานของโปรแกรม

สรุป

การเขียนสคริปต์ PHP นอกจากจะมีฟังก์ชันมาตรฐานที่มากับโปรแกรมภาษา PHP เพื่ออำนวยความสะดวกในการเขียนสคริปต์แล้ว ผู้ใช้ยังสามารถสร้างฟังก์ชันขึ้นมาใช้งานได้ด้วยตนเอง ลักษณะของงานที่จะนำมาสร้างเป็นฟังก์ชันนั้น ควรเป็นงานหรือการกระทำอย่างใดอย่างหนึ่ง ที่มีการทำงานแบบซ้ำๆ และบ่อยครั้ง สามารถนำมาสร้างเป็นฟังก์ชันเพื่อลดเวลาในการเขียนสคริปต์ ฟังก์ชันที่ผู้ใช้สร้างอาจจะมีการส่งผ่านค่าตัวแปรเข้าไปประมวลผลภายในฟังก์ชัน แล้วส่งค่ากลับเพื่อแสดงผลหรืออื่นๆ บางฟังก์ชันอาจเป็นเพียงรูปแบบการแสดงผลง่ายๆ แต่ทำงานแบบซ้ำๆ อาจจะไม่จำเป็นต้องส่งผ่านค่าตัวแปรก็ได้ อีกทั้งยังสามารถสร้างแยกออกจากเว็บเพจ เป็นไฟล์ที่เก็บฟังก์ชันโดยเฉพาะ หรือเรียกว่าฟังก์ชันไลบรารี และเมื่อต้องการเรียกใช้งานฟังก์ชันใดๆ ก็เพียงนำเข้าไฟล์ที่เก็บฟังก์ชันนั้น ก็จะสามารถเรียกใช้งานได้ทันที ทำให้การเขียนสคริปต์ทำได้ง่ายและรวดเร็ว

คำถามท้ายบท

1. จงอธิบายข้อกำหนด และหลักเกณฑ์การตั้งชื่อฟังก์ชัน พร้อมยกตัวอย่างรูปแบบการสร้างฟังก์ชันที่ผู้ใช้งานเป็นผู้สร้างเอง
2. จงอธิบายฟังก์ชันพารามิเตอร์แบบส่งผ่านค่าโดยการอ้างอิง พร้อมยกตัวอย่างประกอบ

3. จงอธิบายถึงหลักการทำงานของฟังก์ชันแบบเรียกตัวเอง และอธิบายถึงคำสั่งที่จะให้ฟังก์ชันแบบเรียกตัวเองสิ้นสุดการทำงาน

4. จงอธิบายหลักการทำงานของสคริปต์ด้านล่างตั้งแต่บรรทัดที่ 1 ถึงบรรทัดสุดท้าย และบอกผลลัพธ์ที่ได้จากสคริปต์

```
1   <?php
2       function pyramid ($number=9) {
3           for ($loop1 = $number; $loop1 >= 1; $loop1--) {
4               for ($loop2=1; $loop2<=$loop1; $loop2++) {
5                   printf (" %d ",$loop1);
6               }
7               echo "<br>";
8           }
9       }
10      pyramid (5);
11  ?>
```

5. จงอธิบายหลักการทำงานของสคริปต์ด้านล่างตั้งแต่บรรทัดที่ 1 ถึงบรรทัดสุดท้าย และบอกผลลัพธ์ที่ได้จากสคริปต์

```
1   <?php
2       function pyramid ($number=9) {
3           for ($loop = $number; $loop >=1; $loop--) {
4               printf (" %d ",$loop);
5           }
6       }
7       for($row=5; $row>=1; $row--) {
8           pyramid ($row);
9           echo "<br>";
10      }
11  ?>
```