Review Day 2

ARRAYS

An ARRAY is a set of elements or data treated as one item. In PHP, an array is created during the declaration of the variable, as follows:

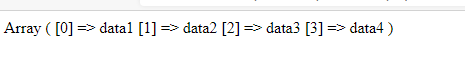
$arrayname = array("data1","data2","data3","data4");

In this array, which is a simple, one-level array, the values in the array may be called by identifying its index, which is its place within the array. Without identifying the index, PHP counts the position as the index, beginning with 0. So if we display the data in this array through the print\_r command as

$arrayname = array("data1","data2","data3","data4");

print\_r($arrayname);

This will display:



To display, for example, the data2, we echo the array variable with its index, as follows:

echo $arrayname[1];

Notice that the index used is 1, which is the second index.

If the array is associative, as this:

$arrayname = array("data1" => "value1",

   "data2" => "value2",

   "data3" => "value3",

   "data4" => "value4");

We get the value by referencing the associative index, which in this case is the data:

echo $arrayname["data1"];

What do you think would the screen display?

CONDITIONAL STATEMENTS

Three options were discussed: IF-THEN, IF-THEN-ELSE, and IF-THEN-ELSEIF

IF-THEN

Syntax:

$variable = "anyvalue";

if(Condition\_that\_should\_be\_tested\_if\_true){

    Instruction\_for\_PHP\_if\_condition\_is\_met;

}

For example, we want PHP to display “Your score passed” if the score is 76. The code would be:

$yourscore = 76;

if($yourscore >= 76){

    echo "Your score passed.";

}

In this case, if the variable $yourscore is 74, it does not do anything. If you want to give PHP an alternate instruction, use IF-THEN-ELSE

IF-THEN-ELSE

Syntax:

$variable = "anyvalue";

if(Condition\_that\_should\_be\_tested\_if\_true){

    Instruction\_for\_PHP\_if\_condition\_is\_met;

} else {

    Instruction\_for\_PHP\_if\_condition\_is\_not\_met;

}

Example, continuing with the previous case, if we want PHP to display “Your score failed” if the score was lower than 76, we write:

$yourscore = 76;

if($yourscore >= 76){

    echo "Your score passed.";

} else {

    echo "Your score failed";

}

If we have multiple options, we use the third function.

IF-THEN-ELSEIF

$variable = "anyvalue";

if(Condition\_that\_should\_be\_tested){

    Instruction\_for\_PHP\_if\_this\_condition\_is\_met;

} elseif(Alternate\_condition\_that\_would\_be\_tested) {

    Instruction\_for\_PHP\_if\_this\_condition\_is\_met;

}

For example, if we have a passing grade of 76, at which the grade is considered “Good”, but we have another range from 86 or higher to be Phenomenal, the code may be something like this:

$yourscore = 55;

if($yourscore < 76){

    echo "Your score Failed.";

} elseif((76 <= $yourscore) && ($yourscore <= 85)) {

    echo "Your score is Good";

} elseif($yourscore >=86){

    echo "Your score is Phenomenal";

}

SWITCH

A switch is like a structured IF-THEN-ELSEIF function, in that if your situation has fairly defined conditions, you can identify a set of instructions for each case. This is the format:

$variable = "value";

switch ($variable) {

    case "case1":

        instruction;

    break;

    case "case2":

        instruction;

    break;

    default:

        instruction;

}

To read this, the $variable is set at value “value”. It will then be subjected to SWITCH, which has a set of options depending on the value of the $variable, identified by the “case1”, “case2”. If the value of the $variable falls neither of the two identified cases, the default instruction will be done.

LOOPS

Loops are similar in conditional statements in that there is a set of instructions depending on a condition. However, a loop is primarily created so that a set of instructions is to be repeated until a certain condition changes or has been detected. The three functions discussed were WHILE, DO-WHILE, and FOREACH

WHILE

WHILE loop is used to repeat something while a certain condition is true. The syntax is as follows:

$i = initialvalue;

while(Condition\_that\_shall\_be\_repeatedly\_tested){

    Set\_of\_instruction\_that\_will\_be\_done\_while\_condition\_is\_true;

}

Consider the following example:

$i = 13;

while($i <15){

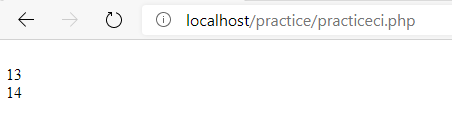
    echo $i."<br />";

    $i++;

}

This means that the initial set value of $i is 13. The loop will test if the $i is less than 15. So long as this true, PHP will echo the value, concatenated with the <br/> tag, then increase by one ($i++).

This will result into the following:



The set value is 13, which is then subjected to the loop. Instruction is to echo that value and enter a break line, then increase it by one. The next value is 14, so the loop is repeated. However, when the value reaches to 15, the value is no longer true to the condition, so the loop stops and the instruction moves to the next line of instruction.

If we want PHP to do the loop at least once, even if the condition does not meet the condition, we use DO-WHILE:

Syntax:

$i = initialvalue;

do {Set\_of\_instruction\_that\_will\_be\_done\_while\_condition\_is\_true}

while(Condition\_that\_shall\_be\_repeatedly\_tested);

Notice that the format is essentially the inverse of the WHILE function, ensuring that the instruction is done once. For example:

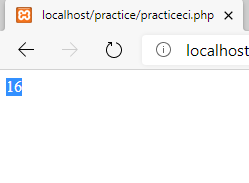
$i = 16;

do {echo $i."<br />";

    $i++;

} while($i <15);

This will display:



Notice that the value is higher than the set condition, but the instruction was done once.

FOREACH

Most of the time, in database-driven applications, a set of instruction will be repeated based on a set of data from a database query. For this purpose, we use FOREACH.

There are two ways of using it, depending on the result.

If the source of the list is a simple, one-level array, like this:

$arrayvariable = array("data1","data2","data3","data4");

foreach($arrayvariable as $instance\_in\_array){

    Instruction;

}

This is useful like this:

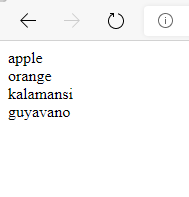
$fruitlist = array("apple", "orange", "kalamansi", "guyavano");

foreach($fruitlist as $fruit){

    echo $fruit . "<br />";

}

This will result in:



If your array is associative, you can use this to display identifier as well as the value:

$arrayvariable = array("data1" => "value1",

    "data2" => "value2",

    "data3" => "value3",

    "data4" => "value4");

foreach($arrayvariable as $instance\_in\_array => $identifier){

    Instruction;

}

For example, we improved our previous $fruitlist array to include the color of the fruit. We can display the name of the fruit with its corresponding color as follows:

$fruitlist = array("apple" => "red",

    "orange" => "orange",

    "kalamansi" => "green",

    "guyavano" => "lightgreen");

foreach($fruitlist as $fruit => $color){

    echo $fruit . " :: " . $color . "<br />";

}

This will display as below:

