Government Science College, Valod

Fundaments of PHP

Unit-3

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

- **if statement** - executes some code only if a specified condition is true
- **if...else statement** - executes some code if a condition is true and another code if the condition is false
- **if...elseif....else statement** - specifies a new condition to test, if the first condition is false
- **switch statement** - selects one of many blocks of code to be executed

PHP - The if Statement

The if statement is used to execute some code **only if a specified condition is true**.

Syntax

```
if (condition) {
    code to be executed if condition is true;
}
```

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

Example

```
<?php
$t = date("H");

if ($t < "20") {
    echo "Have a good day!";
}
?>
```

PHP - The if...else Statement

Use the if....else statement to execute some code **if a condition is true and another code if the condition is false**.

Syntax

```
if (condition) {
    code to be executed if condition is true;
} else {
    code to be executed if condition is false;
}
```

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

Example

```
<?php
$t = date("H");

if ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

PHP - The if...elseif....else Statement

Use the if....elseif...else statement to **specify a new condition to test, if the first condition is false.**

Syntax

```
if (condition) {
    code to be executed if condition is true;
} elseif (condition) {
    code to be executed if condition is true;
} else {
    code to be executed if condition is false;
}
```

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

Example

```
<?php
$t = date("H");

if ($t < "10") {
    echo "Have a good morning!";
} elseif ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

The PHP switch Statement

The switch statement is used to perform different actions based on different conditions.

Use the switch statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n) {
    case label1:
        code to be executed if n=label1;
        break;
    case label2:
        code to be executed if n=label2;
        break;
    case label3:
        code to be executed if n=label3;
        break;
    ...
    default:
        code to be executed if n is different from all labels;
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from

running into the next case automatically. The **default** statement is used if no match is found.

Example

```
<?php
$favcolor = "red";

switch ($favcolor) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is blue!";
        break;
    case "green":
        echo "Your favorite color is green!";
        break;
    default:
        echo "Your favorite color is neither red, blue, nor green!";
}
?>
```

PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- **while** - loops through a block of code as long as the specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

The PHP while Loop

The while loop executes a block of code as long as the specified condition is true.

Syntax

```
while (condition is true) {
    code to be executed;
}
```

The example below first sets a variable $x to 1 ($x = 1). Then, the while loop will continue to run as long as $x is less than, or equal to 5 ($x <= 5). $x will increase by 1 each time the loop runs ($x++):

Example

```
<?php
$x = 1;

while($x <= 5) {
    echo "The number is: $x <br>";
    $x++;
}
?>
```

The PHP do...while Loop

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

```
do {
    code to be executed;
} while (condition is true);
```

The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

Example

```
<?php
$x = 1;

do {
    echo "The number is: $x <br>";
    $x++;
} while ($x <= 5);
?>
```

Notice that in a do while loop the condition is tested AFTER executing the statements within the loop. This means that the do while loop would execute its statements at least once, even if the condition is false the first time.

The example below sets the $x variable to 6, then it runs the loop, **and then the condition is checked**:

Example

```
<?php
$x = 6;

do {
    echo "The number is: $x <br>";
    $x++;
} while ($x<=5);
?>
```

The PHP for Loop

PHP for loops execute a block of code a specified number of times.

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (init counter; test counter; increment counter) {
    code to be executed;
}
```

Parameters:

- *init counter*: Initialize the loop counter value
- *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment counter*: Increases the loop counter value

The example below displays the numbers from 0 to 10:

Example

```
<?php
for ($x = 0; $x <= 10; $x++) {
    echo "The number is: $x <br>";
}
?>
```

The PHP foreach Loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

```
foreach ($array as $value) {
    code to be executed;
}
```

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

The following example demonstrates a loop that will output the values of the given array ($colors):

Example

```
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
    echo "$value <br>";
}
?>
```

PHP User Defined Functions

A function is a block of statements that can be used repeatedly in a program.

A function will not execute immediately when a page loads.

A function will be executed by a call to the function.

Create a User Defined Function in PHP

A user defined function declaration starts with the word "function":

Syntax

```
function functionName() {
    code to be executed;
}
```

Note: A function name can start with a letter or underscore (not a number).

Tip: Give the function a name that reflects what the function does!

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code and the closing

curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

Example

```
<?php
function writeMsg() {
    echo "Hello world!";
}

writeMsg(); // call the function
?>
```

PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just seperate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example

```
<?php
function familyName($fname) {
    echo "$fname Refsnes.<br>";
}

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
?>
```

The following example has a function with two arguments ($fname and $year):

Example

```
<?php
function familyName($fname, $year) {
    echo "$fname Refsnes. Born in $year <br>";
}

familyName("Hege", "1975");
familyName("Stale", "1978");
```

```
familyName("Kai Jim", "1983");
?>
```

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example

```
<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

PHP Functions - Returning values

To let a function return a value, use the return statement:

Example

```
<?php
function sum($x, $y) {
    $z = $x + $y;
    return $z;
}
echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```

PHP Arrays

An array stores multiple values in one single variable:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
```

```
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

In PHP, the array() function is used to create an array:

```
array();
```

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

Get The Length of an Array - The count() Function

The count() function is used to return the length (the number of elements) of an array:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo count($cars);
?>
```

Loop Through an Indexed Array

To loop through and print all the values of an indexed array, you could use a for loop, like this:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);

for($x = 0; $x < $arrlength; $x++) {
    echo $cars[$x];
    echo "<br>";
}
?>
```

PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
```

Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
```

PHP Sorting Arrays

The elements in an array can be sorted in alphabetical or numerical order, descending or ascending.

PHP - Sort Functions For Arrays

In this chapter, we will go through the following PHP array sort functions:

- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order
- asort() - sort associative arrays in ascending order, according to the value
- ksort() - sort associative arrays in ascending order, according to the key
- arsort() - sort associative arrays in descending order, according to the value
- krsort() - sort associative arrays in descending order, according to the key

Sort Array in Ascending Order - sort()

The following example sorts the elements of the $cars array in ascending alphabetical order:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);
?>
```

The following example sorts the elements of the $numbers array in ascending numerical order:

Example

```
<?php
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);
?>
```

Sort Array in Descending Order - rsort()

The following example sorts the elements of the $cars array in descending alphabetical order:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
rsort($cars);
?>
```

The following example sorts the elements of the $numbers array in descending numerical order:

Example

```
<?php
$numbers = array(4, 6, 2, 22, 11);
rsort($numbers);
?>
```

Sort Array (Ascending Order), According to Value - asort()

The following example sorts an associative array in ascending order, according to the value:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
asort($age);
?>
```

Sort Array (Ascending Order), According to Key - ksort()

The following example sorts an associative array in ascending order, according to the key:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);
?>
```

Sort Array (Descending Order), According to Value - arsort()

The following example sorts an associative array in descending order, according to the value:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);
?>
```

Sort Array (Descending Order), According to Key - krsort()

The following example sorts an associative array in descending order, according to the key:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
krsort($age);
?>
```

References :

1. W3C School for PHP http://www.w3schools.com/php/