Chapter 2 - Control Structures in PHP Scripts

ICT24A Web 2 Programming

Introduction

Any PHP script is built out of a series of statements. A statement can be an assignment, a function call, a loop, a conditional statement or even a statement that does nothing.

In this chapter you will learn about control structures which allow us to control the order in which script statements are executed. Like many other programming languages, PHP includes several different control structures that can be grouped into two different areas: conditional statements and loops. We will investigate various options for each type of structure.

Chapter 2 - Control Structures in PHP Scripts

ICT24A Web 2 Programming

Expressions and Conditions.

Before we proceed any further, consideration needs to be given to Expressions, Conditions, and Operators because they form an integral part of these type of PHP statements

Expressions

Expressions form part of many script statements. An expression is a part of the statement in a programming language that returns a value. It is made up of a combination of explicit values, variables, constants, operators and functions . The expression is evaluated according to what are called the "Rules of Precedence" of the programming language.

Some examples of expressions are:

5 + 2 -3
5 > 3
NOT "Smith"
First_Name = 'Peter'

The phrase **Last_Name = "Rennick"** is an expression, and in fact is an example of what is called a Boolean expression (one that will be either true or false). **Last_Name = "Rennick"** is either true or false.

Conditions

Expressions contain one or more conditions and conditions make selective statements possible. In their most common form they are composed of a variable, a constant, and a comparison operator. In the preceding example the expression **Last_Name = "Rennick"** is also a condition.

Conditional Statements

The 'if' Construct

The if construct is one of the most important features of many languages, PHP included. It allows for conditional execution of code statements. As described in the section about expressions; if the expression evaluates to **true**, PHP will execute the following statement, and if it evaluates to **False** it will ignore it.

```
<?php
// example2.1.php – The IF Construct

$intA = 5;
$intB = 3;
if ($intA > $intB)
echo "<p>$intA is greater than $intB</p>";
?>
```

Chapter 2 - Control Structures in PHP Scripts

ICT24A Web 2 Programming

Often you will need to execute a number of statements if a condition is met. If this is the case we need to group those statements together using opening and closing brace characters as shown below.

```
<?php
// Example2.2.php

$strColour = "green";
if ($strColour == "green") {
        echo "<p>The colour is green</p>";
        echo "<p>Green is a nice colour</p>";
        echo "<p>We have the colour of grass</p>";
}
?>
```

The else statement

Often you'd want to execute a statement if a certain condition is met, and a different statement if the condition is not met. This is the purpose of ***else*** *statement.*

Else extends an ***if*** statement to execute a statement in case the expression in the ***if*** statement evaluates to FALSE

```
<?php
// Example2.3.php

$intA =9;
$intB = 3;

if ($intA > $intB)
   echo "<p>$intA is greater than $intB</p>";
else
   echo "<p>$intA is less than or equal to $intB</p>";
?>
```

Note: As you can see from the example above there is no closing endif statement as is the case with many programming languages.

In the example below we can see the use of the open and closing braces in both the if and the else sections of the selection control structure.

```
<?php
// Example2.4.php

$strColour = "blue";

if ($strColour == "green") {
        echo "<p>The colour is green</p>";
        echo "<p>Green is a nice colour</p>";
        echo "<p>We have the colour of grass</p>";
}
else {
        echo "<p>We don't know what colour we have</p>";
        echo "<p>Other than it is not green</p>";
}
?>
```

See Over..

elseif

Like the ***else*** statement elseif extends an ***if*** statement to execute a different statement in case the original ***if*** expression evaluates to **FALSE**. However, unlike ***else***, it will execute that alternative expression only if the ***elseif*** conditional expression evaluates to **TRUE**

There may be several ***elseif***s within the same ***if*** statement. The first ***elseif*** expression (if any) that evaluates to **TRUE** would be executed. In PHP, you can also write 'else if' (in two words) and the behavior would be identical to the one of 'elseif' (in a single word).

```
<?php

// Example2.5.php

$intNumber1 = 100;
$intNumber2 = 80;

if ($intNumber1 > $intNumber2) {
        echo "<p>$intNumber1 is larger than $intNumber2</p>";
}

elseif ($intNumber1 == $intNumber2) {
        echo "<p>$intNumber1 is equal to $intNumber2</p>";
}

else {
        echo "<p>$intNumber1 is smaller than $intNumber2</p>";
}
?>
```

See Over..

switch

The ***switch*** statement is similar to a series of IF statements on the same expression. In many occasions, you may want to compare the same variable (or expression) with many different values, and execute a different piece of code depending on which value it equals to. This is precisely the purpose of the ***switch*** statement.

Switch works in a curious way however and it is easy to make a mistake using this statement. If you code the example below and run the program you will notice something add

```
<?php
// Example2.6.php

$strName = "Elizabeth";

switch ($strName) {
case "Simon":
        echo "<p>Hello Simon</p>";
case "Elizabeth":
        echo "<p>Hello Elizabeth</p>";
case "Hayley":
        echo "<p>Hello Hayley</p>";
case "Alan":
        echo "<p>Hello Alan</p>";
}
?>
```

The output is

Hello Elizabeth
Hello Hayley
Hello Alan

It is important to understand how the ***switch*** statement is executed in order to avoid mistakes. The ***switch*** statement executes line by line (actually, statement by statement). In the beginning, no code is executed. Only when a ***case*** statement is found with a value that matches the value of the ***switch*** expression does PHP begin to execute the statements.
PHP continues to execute the statements until the end of the ***switch*** block, or the first time it sees a ***break*** statement. If you don't write a ***break*** statement at the end of a case's statement list, PHP will go on executing the statements of the following case.

In the example above you can see the result is not what we want. We need to include some break statements to stop processing and jump out of the Switch statement.

The corrected code is shown below.

```
<?php

// Example2.7.php

$strName = "Elizabeth";

switch ($strName) {
case "Simon":
        echo "<p>Hello Simon</p>";
        break;
case "Elizabeth":
        echo "<p>Hello Elizabeth</p>";
        break;
case "Hayley":
        echo "<p>Hello Hayley</p>";
        break;
case "Alan":
        echo "<p>Hello Alan</p>";
}
?>
```

A special case is the ***default*** case. This case matches anything that wasn't matched by the other cases and therefore should be the last statement.

```
<?php

// Example2.8.php

$intTotal = 6;

switch ($intTotal) {
case 0:
case 1:
case 2:
case 3:
case 4:
        echo "<p>$intTotal is less than or equal to four!</p>";
        break;
case 5:
        echo "<p>$intTotal is equal to five!</p>";
        break;
default:
        echo "<p>$intTotal is greater than five!</p>";
}
?>
```

Chapter 2 - Control Structures in PHP Scripts

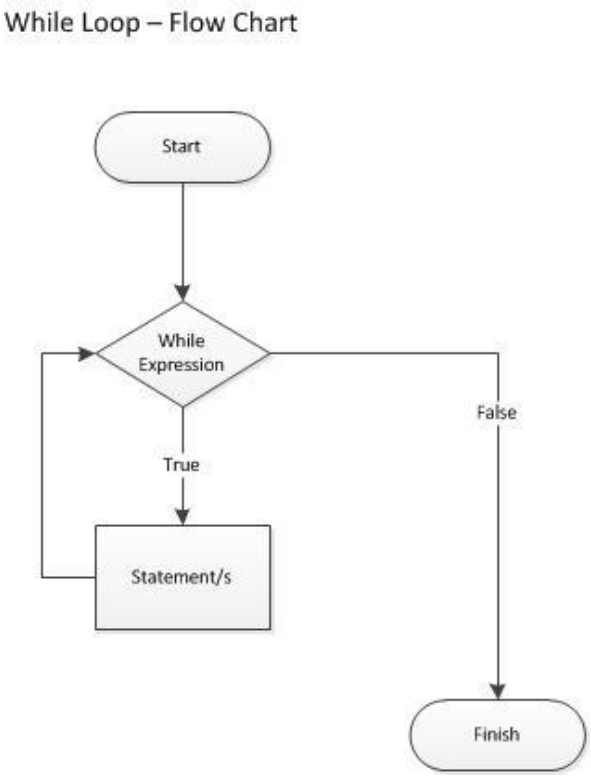
ICT24A Web 2 Programming

Loops

while

A while loop is a pre-test loop and if the expression that is being tested the evaluates to FALSE on the first iteration statements within the loop are not executed at all.

while loops are the simplest type of loop in PHP. The meaning of a ***while*** statement is simple. It tells PHP to execute the nested statement(s) repeatedly, as long as the ***while*** expression evaluates to **TRUE**. See the Flow chart below.



The value of the expression is checked each time at the beginning of the loop, so even if this value changes during the execution of the nested statement(s), execution will not stop until the end of the iteration. Each time PHP runs the statements in the loop is one iteration.

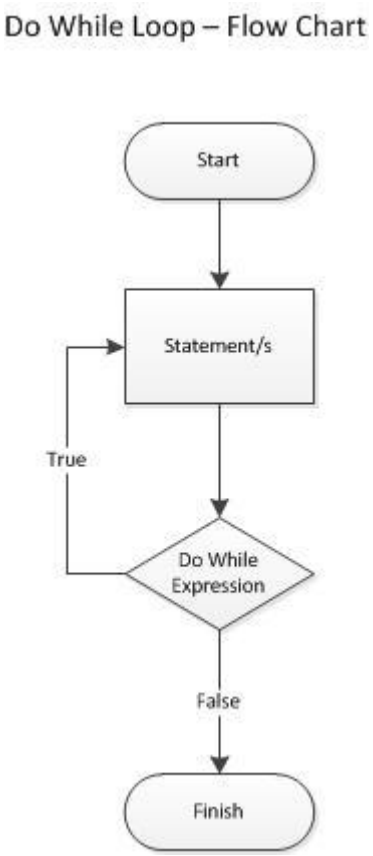
You can group multiple statements within the same ***while*** loop by surrounding a group of statements with curly braces.

```
<?php

// Example2.9.php

$intCount = 1;
while ($intCount <= 10) {
        echo "<p>Iteration $intCount</p>";
        $intCount++;
}
?>
```

While loops often use a variable which is incremented or decremented within the loop to control the number of loop iterations.

do-while

*A **do-while** loop is a post-test loop where the expression is being tested after the statements have been run. So the statements will run once even if the expression evaluates to FALSE. (as in Example 2.11)* See the Flow chart below.




```
<?php

// Example2.10.php

$intCount = 1;

do {
   echo "<p>Iteration $intCount</p>";
   $intCount++;
} while ($intCount <= 10);
?>
```

```
<?php

// Example2.11.php

$intCount = 1;

do {
   echo "<p>Iteration $intCount</p>";
   $intCount++;
} while ($intCount < 1);
?>
```

for

for loops are the most complex loops in PHP.

The first expression (*expr1*) is evaluated (executed) once unconditionally at the beginning of the loop.

In the beginning of each iteration, *expr2* is evaluated. If it evaluates to **TRUE**, the loop continues and the nested statement(s) are executed. If it evaluates to **FALSE**, the execution of the loop ends.

At the end of each iteration, *expr3* is evaluated (executed).

Each of the expressions can be empty or contain multiple expressions separated by commas.
In *expr2*, all expressions separated by a comma are evaluated but the result is taken from the last part. *expr2* being empty means the loop should be run indefinitely (PHP implicitly considers it as **TRUE**, like C). This may not be as useless as you might think, since often you'd want to end the loop using a conditional **break** statement instead of using the ***for*** truth expression.

```
<?php

// Example2.12.php

for($intCount = 1; $intCount <= 10; $intCount++)
   echo "<p>Iteration $intCount</p>";
?>
```

Chapter 2 - Control Structures in PHP Scripts

ICT24A Web 2 Programming

Nested Loops

The example below includes a range of control structures that we have previously examined together with HTML tags to create a table that looks like a chess board. It uses loops within loops or what are named 'Nested Loops'. When completing this script you will need to place the 2 image files in the same folder as your script.

```
<?php

// Example2.13.php

$booBlackWhite = 0;
echo "<table border='1'>";
for($intRows = 1; $intRows <= 8; $intRows++) {
   $intColumns = 1;
   echo "<tr>";
   while ($intColumns <= 8) {
      if ($booBlackWhite)
        echo "<td><img src='whiteSquare.gif' width='30' height='30' alt='blackSquare'  align='top'/></td>";
      else
        echo "<td><img src='blackSquare.gif' width='30' height='30' alt='whiteSquare' align='top'/></td>";
      $intColumns++;
      if ($booBlackWhite == 1)
         $booBlackWhite = 0;
      else
         $booBlackWhite = 1;
   }
   echo "</tr>";
   if ($booBlackWhite == 1)
      $booBlackWhite = 0;
   else
      $booBlackWhite = 1;
}
echo "</table>";
?>
```

Chapter 2 - Control Structures in PHP Scripts

ICT24A Web 2 Programming

The break statement

The script below is a program that would continue without end if we removed the 'break' statement as the while(1) condition is always true. Break terminates process within a loop and execution of the commands will move on to the next command after the loop (if one exists).

```
<?php

// Example2.14.php

$intCount = 1;

while (1) {
        echo "<p>Iteration $intCount</p>";
        $intCount++;
        if ($intCount > 6)
                break;
}
?>
```

The continue statement

The continue statement is used within the loop to skip the remainder of the statements within the current iteration and jump to the next iteration. If we look at the example below the continue statement is used to jump over the command will print a value to the screen if the number is an odd number.

```
<?php

// Example2.15.php

for($intCount=1; $intCount<10; $intCount++){
        if ($intCount % 2)
     continue;
  echo "<p>$intCount is even.</p>";
}
?>
```

Continue your work by completing Tutorial 3.