

PHP Basics – Part 2: Conditions and Loops

Conditional Processing

Conditional Processing is used in almost every single PHP script that you'll create. By using conditional processing, you can check a certain condition, such as the contents of a variable, and then run one block of code or another, according to the condition that you set. An example of how this could be used is in a login system, so if a user logs in with the correct username and password you could run a block of code that redirects the user to another page, inside of the system, whereas if the wrong username or password is entered you can run a different block of code to display a message telling the user their details are incorrect.

PHP offers a number of different commands for conditional processing, which we're going to look at now starting with the PHP `if` command.

if

As the name suggests, the PHP `if` command is used to run a section of code if a certain condition is true. The command uses the following syntax:

```
if(condition){  
    // code to run if condition is true  
}
```

If the condition evaluates to true, then the code in between the braces (`{ }`) is run, otherwise it is simply ignored.

As a very simple example, open a new PHP page in Dreamweaver MX, switch to code view, and add the following code to the page body (between the HTML `<body></body>` tags).

```
<?php  
$name = "Gareth";  
if($name == "Gareth"){  
    echo "Hello Gareth";  
}  
?>
```

When the page is viewed in the browser, it will display "Hello Gareth". First, we have set the variable `$name` to contain "Gareth". We then use the `if` command to check whether `$name` contains "Gareth", and if it does then the code in between the braces is run, outputting "Hello Gareth" to the screen. Note that, as explained in [part 1 of this tutorial](#), when checking for equality we use two equals signs i.e. `==`. If we only used one equals sign we would reassign "Gareth" to `$name` and so the condition would always evaluate to true.

Change the name assigned to `$name` as shown in the code below, and view the page in your browser.

```
<?php  
$name = "Bruce";  
if($name == "Gareth"){  
    echo "Hello Gareth";  
}
```

```
}  
?>
```

This time, when the page is viewed you won't see any output, as the `if` condition is now false and so the code in the braces is ignored.

So far, we've seen that if the condition used with the `if` command evaluates to false, the code in the braces is simply ignored. However, we can extend the `if` command so that if the condition in the brackets evaluates to false then we run some alternative code.

else

The `else` command is used with the `if` command as shown below:

```
if(condition){  
    // code to run if condition is true  
} else {  
    // code to run if condition is false  
}
```

When the `if` command is used with `else`, then one block of code in the braces is always run, depending on whether the condition is true or false. As an example, try the following block of code below:

```
<?php  
$name = "Gareth";  
if($name == "Gareth"){  
    echo "Hello Gareth (Administrator)";  
} else {  
    echo "Hello " . $name;  
}  
?>
```

When this code is run, because `$name` is set to `Gareth`, it will output `"Hello Gareth (Administrator)"`. However, if you change `$name` to `"Bruce"` because the `if` condition is no longer true, it will output `"Hello Bruce"`.

Although this is extremely useful in scripts, it does limit you to only two possible outcomes i.e. when the condition is true or false. However, if you require more choices, you can extend the `else` command with a new command, `elseif`.

elseif

Using the `elseif` command allows you to test multiple conditions and run different blocks of code depending on the outcome.

The format for using `elseif` is shown below:

```
if(condition){  
    // code to run if condition is true  
} elseif(condition) {  
    // code to run if new condition is true
```

```
} else {  
    // code to run if all previous conditions are false  
}
```

With the `elseif` statement, the various conditions are tested in the order that they appear, and if one of the conditions evaluates to true the relevant block of code is run. If none of the conditions are true, then the final block of code is run. As an example, try the code below:

```
<?php  
$animal = "iguana";  
if($animal == "dog"){  
    echo "The dogs name is Fido";  
} elseif($animal == "iguana") {  
    echo "The iguanas name is Iggy";  
} else {  
    echo "We don`t know what the " . $animal . " is called!";  
}  
?>
```

Here, we have three different options, the first two are matched if the conditions are true and the last option is matched if the previous conditions fail.

When the code is run, if `$animal` contains "dog" then the first block will print "The dogs name is Fido". If `$animal` contains "iguana", then the code will print "The iguanas name is Iggy". However, if `$animal` contains an unknown animal, for example if `$animal` contains "cow", then because none of the conditions match the code will print "We don't know what the cow is called!".

You can use as many `elseif` statements as you need, to cover a range of different conditions. However, there is an alternative command that you can use which evaluates a single expression and can provide different output depending on the result of the expression.

Switch

Whereas the PHP `if` and `elseif` commands run code depending on the result of different expressions, as shown in the examples above, the `switch` command evaluates a single expression and runs different code depending on different values of the expression.

The format for the `switch` command is shown below:

```
switch(expression){  
    case result1:  
        // This code is run if expression equals result1  
        break;  
    case result2:  
        // This code is run if expression equals result2  
        break;  
    default:  
        // This code is run if the expression does not equal result1 or result2  
}
```

As an example, we'll rewrite the animal example above to use the `switch` statement. Change the code to that shown below:

```
<?php
$animal = "dog";
switch($animal){
    case "iguana":
        print "The iguana is called Iggy";
        break;
    case "dog":
        print "The dogs name is called Fido";
        break;
    case "cow":
        print "The cows name is Daisy";
        break;
    default:
        print "We don`t know what the " . $animal . " is called!";
}
?>
```

In the code above, the `case` statement is matched with the value in `$animal`, and the relevant block of code is run. If none of the case values are matched with the value in `$animal`, then the block of code under the `default` statement is run.

Note that at the end of each block of code for each case statement, there is a PHP `break` command, which stops the execution of any further code. This is essential, otherwise all further case statements underneath the one which matches the value in `$animal` are run, which will create unintended results.

Whilst all of these examples have been fairly simple, and show simple output, they can easily be used to output HTML which can be used to display different sections of a web page, depending on values the users have entered in a form for example, and we'll be looking at this further on this series of tutorials.

Loops

Loops are another essential part of PHP scripting, and as the name suggests, allow you to repeat a section of PHP code a certain number of times.

For Loops

The first loop command that we're going to look at is the PHP `for` loop. This has the format:

```
for(variable initialization; test expression; variable modification){
    // Code to run
}
```

The easiest way to understand this is with a simple example, as shown with the code below.

```
<?php
for($i=1;$i <= 10; $i++){
    echo $i . "<br>";
}
?>
```

First, we set the variable `$i` to 1 to start with. We then set the expression to test which in this case is `$i <= 10`, and the loop will run as long as this expression remains true. We then have the variable modification, which adds one to the value in `$i` each time round the loop. If we didn't do this then the loop would run forever.

Inside the loop, we output the value of `$i` each time, along with an HTML `
` tag so that each number appears on a new line.

When this code is run, it outputs the values from 1 to 10 down the page. First, `$i` starts containing the value 1 and this is tested against the expression. As `$i` is less than 10, the code in the loop is run, and then the value in `$i` is incremented by 1. This process is repeated until the value in `$i` is greater than 10, which causes the expression to return false.

Loops are extremely useful for creating HTML tables, as shown by the example below:

```
<table width="50" border="1" cellspacing="0" cellpadding="0">
<?php
for($i=1;$i <= 5; $i++){
    echo "<tr>";
    echo "<td align='center'>" . $i . "</td>";
    echo "</tr>";
}
?>
</table>
```

Here, we create an opening and closing HTML `<table></table>` tag, and then use a PHP `for` loop to output the rest of the HTML for the table rows and cells. When this is run it will output the following HTML

```
<table width="50" border="1" cellspacing="0" cellpadding="0">
<tr>
<td align='center'>1</td>
</tr>
<tr>
<td align='center'>2</td>
</tr>
<tr>
<td align='center'>3</td>
</tr>
<tr>
<td align='center'>4</td>
</tr>
<tr>
<td align='center'>5</td>
</tr>
</table>
```

Which outputs a properly formed HTML table with just a few lines of PHP code, this is extremely useful for displaying data from a database in a table, which we'll be looking at later on in this series. Although in the above example we have printed out the HTML using PHP, we can also code the example in another way as shown below.

```
<table width="50" border="1" cellspacing="0" cellpadding="0">
<?php for($i=1;$i <= 5; $i++){ ?>
<tr>
    <td align='center'><?php echo $i; ?></td>
```

```
</tr>
<?php } ?>
</table>
```

In the above example, rather than using PHP to output the HTML required for the table, we've wrapped the PHP code around the HTML, and just used the PHP to output the actual value of `$i`. Note that the HTML which is sandwiched between the PHP code is still repeated each time round the loop. Whichever of these two methods you use is up to you, although the first method where you use PHP to output the HTML is generally best used when you only have a small amount of HTML. Although it's irrelevant in terms of PHP, the second method has an advantage when you use Dreamweaver MX, as it can still display the HTML in design view, which it can't if you use PHP to output the HTML.

While

The next loop that we are going to look at is the `while` loop, which runs while a certain condition is true. The format for the `while` loop is shown below.

```
while(condition){
    // This code is run while the condition is true
}
```

Unlike the `for` loop, which runs a certain number of times, the `while` loop repeats continuously as long as the condition is true, and only stops when the condition becomes false. This means that it's important to have some code inside the loop which will affect the condition, otherwise the loop will run forever!

As a simple example, try the code shown below.

```
<?php
$count = 1;
while($count < 10){
    echo $count . "<br>";
    $count++;
}
?>
```

This will print out the numbers 1 to 9 to the browser. Each time round the loop we increment the value in `$count` by 1, so after 9 iterations the loop will stop, as `$count` becomes 10 and so the condition becomes false.

Do ... While

The `do ... while` loop is similar to the `while` loop above, but works in the opposite way in that the condition comes at the end of the loop rather than the start. This means that the code in the loop is always run once at least, as it comes before the test which could stop the loop.

The format for the `do ... while` loop is shown below.

```
do {
    // This code is run each time round the loop
} while (condition)
```

An example of the do ... while loop is shown below.

```
<?php
$count = 1;
do{
    echo $count . "<br>";
    $count++;
} while($count < 10);
?>
```

First, we set `$count` to 1. Each time round the loop we output the value held in `$count` and then increment it by 1, and as a result the numbers 1 – 9 are printed to the browser, as with the previous example.

Try modifying the code as shown below.

```
<?php
$count = 10;
do{
    echo $count . "<br>";
    $count++;
} while($count < 10);
?>
```

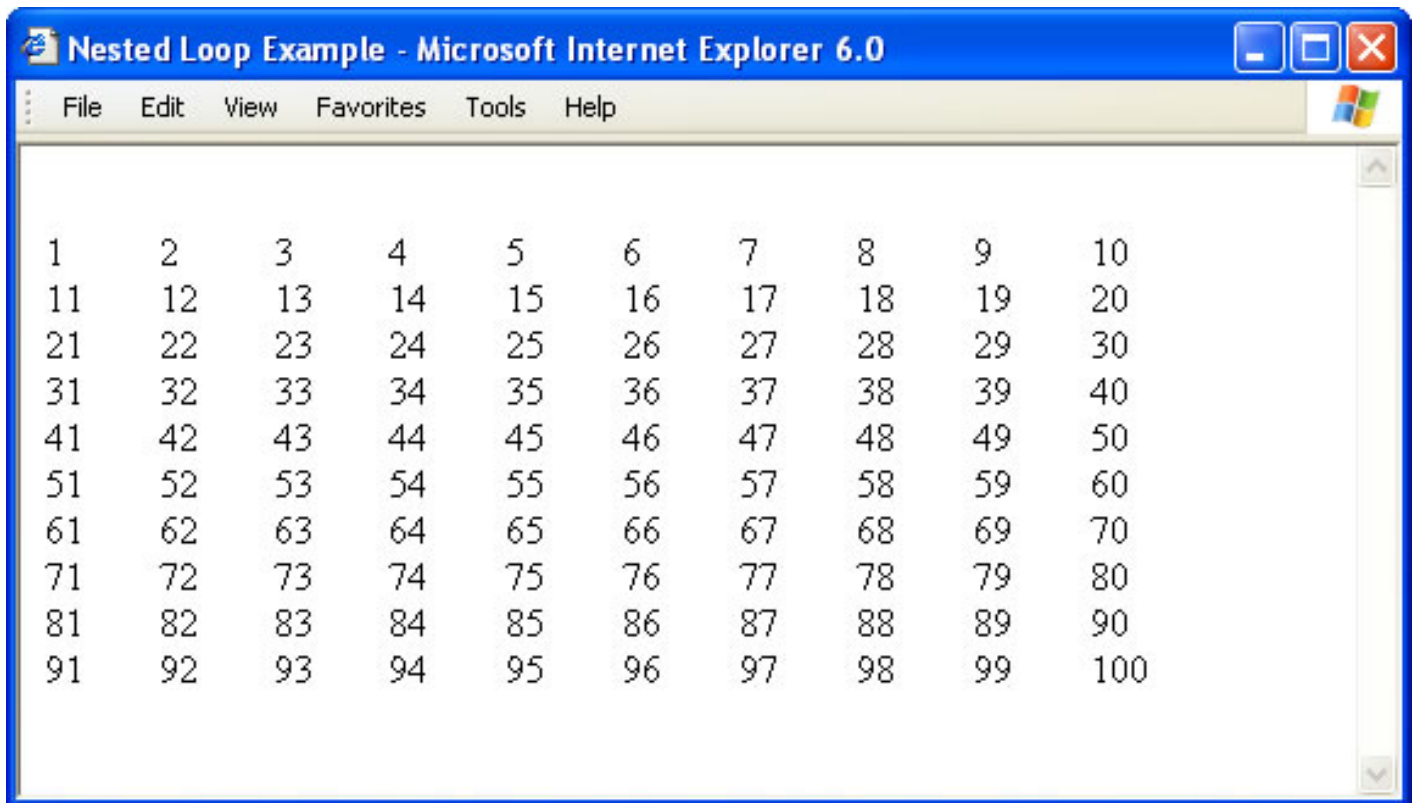
This time when the code is run, 10 is printed to the screen as the value in `$count` is output before the condition (`$count < 10`) is checked.

Nested Loops

Nested loops are loops within a loop. These are extremely useful when creating HTML tables, with one loop to create the table rows, and another nested loop to create the table columns. As an example, try the code shown below.

```
<?php
$count = 1;
// Loop to create table rows
for($i=1; $i < 11; $i++){
    echo "<tr>";
    // Loop to create table columns
    for($j=1; $j < 11; $j++){
        echo "<td>" . $count . "</td>";
        $count++;
    }
    echo "</tr>";
}
?>
```

When this code is run, it produces the output shown in the figure below.



The screenshot shows a web browser window titled "Nested Loop Example - Microsoft Internet Explorer 6.0". The browser's menu bar includes File, Edit, View, Favorites, Tools, and Help. The main content area displays a 10x10 grid of numbers from 1 to 100, arranged in 10 rows and 10 columns. The numbers are: Row 1: 1-10; Row 2: 11-20; Row 3: 21-30; Row 4: 31-40; Row 5: 41-50; Row 6: 51-60; Row 7: 61-70; Row 8: 71-80; Row 9: 81-90; Row 10: 91-100.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

First, we setup a variable `$count` which holds a running total and will contain a figure between 1 and 100. We then create a loop which uses the variable `$i` to run 10 times to create the table rows. We then use a nested loop with the variable `$j` to again run 10 times to create the table columns, incrementing the `$count` variable for each cell of the table.

This technique can be used to easily create tables with any number of rows and columns, and is useful for laying out pictures for example on a gallery site.

Summary

In the first part of this tutorial we looked at using conditional processing, so that code is run only if certain conditions are true. This allows you to change the output of your web page depending on the actions a user takes, or the data that they've entered.

In the second part of this tutorial we looked at the different types of loops that are available in PHP, and how they are used, for example with creating HTML tables.

In the next part of this series, we'll be looking at arrays, and how to work with them, to read data submitted from a form by a user for example.