PHP Arrays

An array can store one or more values in a single variable name.

When working with PHP, sooner or later, you might want to create many similar variables.

Instead of having many similar variables, you can store the data as elements in an array.

Each element in the array has its own ID so that it can be easily accessed.

There are three different kind of arrays:

- Numeric array - An array with a numeric ID key
- Associative array - An array where each ID key is associated with a value
- Multidimensional array - An array containing one or more arrays

Numeric Arrays

A numeric array stores each element with a numeric ID key.

There are different ways to create a numeric array.

In this example the ID key is automatically assigned:

```
$names = array("Peter","Quagmire","Joe");
```

In this example we assign the ID key manually:

```
$names[0] = "Peter";
$names[1] = "Quagmire";
$names[2] = "Joe";
```

Numeric Arrays

The ID keys can be used in a script:

```
<?php

$names[0] = "Peter";
$names[1] = "Quagmire";
$names[2] = "Joe";

echo $names[1] . " and " . $names[2] .
" are ". $names[0] . "'s neighbours";

?>
```

The code above will output:

```
Quagmire and Joe are Peter's neighbours
```

Associative Arrays

An associative array, each ID key is associated with a value.

When storing data about specific named values, a numerical array is not always the best way to do it.

With associative arrays we can use the values as keys and assign values to them.

In this example we use an array to assign ages to the different persons:

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

Associative Arrays

This example is the same as example 1, but shows a different way of creating the array:

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

Associative Arrays

The ID keys can be used in a script:

```
<?php

$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";

echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

The code above will output:

```
Peter is 32 years old.
```

Multidimensional Arrays

In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

In the next example we create a multidimensional array, with automatically assigned ID keys:

Multidimensional Arrays

```
$families = array
(
  "Griffin"=>array
  (
  "Peter",
  "Lois",
  "Megan"
  ),
  "Quagmire"=>array
  (
  "Glenn"
  ),
  "Brown"=>array
  (
  "Cleveland",
  "Loretta",
  "Junior"
  )
);
```

Multidimensional Arrays

The array above would look like this if written to the output:

```
Array
(
[Griffin] => Array
  (
  [0] => Peter
  [1] => Lois
  [2] => Megan
  )
[Quagmire] => Array
  (
  [0] => Glenn
  )
[Brown] => Array
  (
  [0] => Cleveland
  [1] => Loretta
  [2] => Junior
  )
)
```

Multidimensional Arrays

Lets try displaying a single value from the array above:

```
echo "Is " . $families['Griffin'][2] .
" a part of the Griffin family?";
```

The code above will output:

```
Is Megan a part of the Griffin family?
```

PHP Looping

Looping statements in PHP are used to execute the same block of code a specified number of times.

Very often when you write code, you want the same block of code to run a number of times. You can use looping statements in your code to perform this.

In PHP we have the following looping statements:

- **while** - loops through a block of code if and as long as a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a special 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 **while** Statement - Syntax

The while statement will execute a block of code **if and as long as** a condition is true.

```
while (condition)
code to be executed;
```

The **while** Statement - Example

The following example demonstrates a loop that will continue to run as long as the variable i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<html>
<body>

<?php
$i=1;
while($i<=5)
  {
  echo "The number is " . $i . "<br />";
  $i++;
  }
?>

</body>
</html>
```

The **do...while** Statement - Syntax

The do...while statement will execute a block of code **at least once** - it then will repeat the loop **as long as** a condition is true.

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

The do...while Statement - Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 5:

```
<html>
<body>

<?php
$i=0;
do
  {
  $i++;
  echo "The number is " . $i . "<br />";
  }
while ($i<5);
?>

</body>
</html>
```

The **for** Statement - Syntax

The for statement is used when you know how many times you want to execute a statement or a list of statements.

```
for (initialization; condition; increment)
{
  code to be executed;
}
```

The **for** Statement - Example

The following example prints the text "Hello World!" five times:

```
<html>
<body>

<?php
for ($i=1; $i<=5; $i++)
{
  echo "Hello World!<br />";
}
?>

</body>
</html>
```

The **foreach** Statement - Syntax

The foreach statement is used to loop through arrays.

For every loop, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop, you'll be looking at the next element.

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

The **foreach** Statement - Example

The following example demonstrates a loop that will print the values of the given array:

```
<html>
<body>

<?php
$arr=array("one", "two", "three");

foreach ($arr as $value)
{
  echo "Value: " . $value . "<br />";
}
?>

</body>
</html>
```