Continue to __PHP__

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
<?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

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;
}
```

```php
<?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!";
}
?>
```

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 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
<?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
<?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
<?php
$x = 6;

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

The for loop and the foreach loop will be explained in the next chapter.

The PHP for Loop

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

The PHP for each Loop

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

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

The real power of PHP comes from its functions; it has more than 1000 built-in functions.