Loops

What is a loop?

A loop is something that goes around in circles.
Imagine your favorite film character or series that ends up in a time loop.
He saw the same day in a loop until he triggered a particular event that allows him to resume his life normally.

In programming, it's exactly the same thing. Except that a programming loop goes around in circles until you tell it to stop.
Of course it's up to you to decide where to start your loop and what to do after finishing a lap.

In PHP, and programming in general, you do not have to use loops. But, many times, they simplify life.
For example, we want to add the numbers from 1 to 4. You can do this way:

```
$ total = 1 + 2 + 3 + 4;
echo $ total;
```

It's quite simple and there is not a lot of code either.
But if you wanted to add a hundred numbers? Thousand?
It will become tedious and very long to write. In this case, a loop makes life a lot easier.

*/! \ You use a loop when you want to run the same code again and again. /! *

Several types of loops exist.

'For'

loop The syntax of a 'For' loop is as follows:

```
<? php
   for (start value; end value; increment) {
      // code to be executed
   }
?>
```

The first thing to do is to write the name of the loop to use, here 'for'. Then we open the parentheses and we enter the following three conditions:

- 'Start value'

The first condition is that where you tell PHP the initial value of your loop.
In other words: "start the loop from which number?"

- 'End value'

The second condition is that where you tell PHP when to finish your loop.

- 'Increment'

The third condition allows to update the initial value.

We return to our previous example with the addition of the numbers from 1 to 4.
This time, we want to add the numbers from 1 to 100:

```
<? php
   $ start = 1;
   $ total = 0;

   for ($ start; $ start <101; $ start = $ start + 1) {
      $ total = $ total + $ start;
   }

   echo "The sum of the numbers is:". total $;
?>
```

Let's take a closer look at this code:

1. We define a variable ($ start = 1) which is our initial value.
2. We define a variable ($ total = 0) that will contain the result of the addition. Initially it is 0.
3. We write our loop that will:
   - ($ start) Start at 1
   - ($ start <101) Stop when the value is greater than 100
   - ($ start = $ start + 1) At each end of the loop, we will increment the value by +1

4. For each loop, we add the total with the current number.
5. Once our loop is finished, we display the result.

To know that:
- You will often see $ i as initial value of a loop.
- You can set the initial value directly in the loop ($ i = 0)
- In PHP, the double plus symbol (++) can increment the value by 1. ($ start = $ start + 1) can be written as ($ start ++)

With what we just saw, we can rewrite our script this way:

```
<? php
   $ total = 0;

   for ($ i = 0; $ i <101; $ i ++) {
      $ total = $ total + $ i;
   }

   echo "The sum of the numbers is:". total $;
?>
```

'For ... Each'

loop The 'foreach' loop is used to browse the values of an array.

Here is the syntax:

```
<? Php
   foreach ($ my_array as $ my_array_elements) {
      // block of code to be executed
   }
?>
```

The first thing to do is to write the name of the loop to use, here 'foreach'. Then we open the parentheses and write:
- $ my_array is the array we want to loop
- $ my_array_elements is a temporary variable that contains the value of the current element of the array.

To illustrate, we repeat the exercise given in TP:

```
<? Php
   // I declare my movie board
   $ movies [0] = 'Pirates of the Caribbean';
   $ movies [1] = 'Star Wars';
   $ movies [2] = 'Fight Club';

   // loop on myboard
   foreach($ movies as $ mov) {
      echo "The movie is $ mov <br>";
       }
?>
```

We can also write 'foreach' in this way:

```
   foreach ($ full_name as $ first_name => $ surname)
       {
echo "My name is:". $ first_name. "and my nickname:". $ surname;
       }
```

This means: "Retain the Key and its Value" (associative array generally).

'While'

loop Instead of using a 'for' loop, you can use a 'while' loop.
The structure of a 'while' loop is simpler than a for loop because you evaluate only one condition.
The loop goes around in circles as long as the condition is true. When the condition is false, the program exits the 'while' loop.

Here is the syntax of a while loop:

```
<? Php
   while (condition) {
      // block of code to be executed;
   }
?>
```

Here is an example of a 'While' loop that will display numbers from 1 to 5

```
<? Php
   $ i = 0;

   while ($ i <5) {
      echo $ i + 1. "<br>";
      $ i ++;
   }
?>
```

At each loop, the condition is checked. If $ i is less than 5, the condition is true and we continue.
If $ i is greater than or equal to 5, the condition is false and therefore leaves the loop.

Loop 'Do ... While'

The two 'while' loops are almost identical.
The only difference is that the condition comes at the end.
So with a 'do ... while' loop, the code is executed at least once before checking the condition.

Here is the syntax:

```
<? Php
   do {
      // code to be executed;
   } while (condition)
?>
```

Here is an example that will only show '9'.
Even if the condition is false one enters once in the loop:

```
<? Php
   $ i = 9;

   do {
      echo $ i. "<br>";
   } while ($ i <9);

?>
```

The 'break' statement

There are times when you want to break out of a loop before everything is executed.
Or, you want to break out of the loop because of an error your user has made.

In this case, you can use the statement **break.**
It means nothing more than typing the word **break**.

Summary

- The 'For' loop is used to execute code a specific number of times.
- The 'For ... each' loop is used to loop on arrays
- The 'While' loop is used to execute code as long as a condition is true
- The 'Do ... While' loop is used to execute code at least once, then the rest depends on the condition