college CDI

Server-side technologies (TCS)

Class 3

Arrays

An array is a variable containing several values indexed with numbers starting with zero. Very flexible, arrays can be used for different purposes and their values can even be other arrays to create a tree.

Creating an array

There are two basic ways an array can be created using the predefined function ***array()*** or directly (and/or automatically) assigning indexes to variables.

```
<?php
$myInstruments = array("Piano", "Drum", "Guitar");
?>
```

OR

```
$myInstruments = array();
$Instruments[0] = "Piano";
$Instruments[1] = "Drum";
$Instruments[2] = "Guitar";
```

OR

```
$myInstruments = array();
$Instruments[] = "Piano";
$Instruments[] = "Drum";
$Instruments[] = "Guitar";
```

Output to screen

In both cases, the index number is used to display one of the items of the array to screen:

```
<?php
echo $Instruments[0];
?>
```

Result:

```
Piano
```

OR

```
<?php
echo "I like hearing a " . $Instruments[1] . "and a " . $Instruments[2]
?>
```

Result:

```
I like hearing a Drum and a Guitar
```

Replacing index numbers with strings

```
<?php
$myInstruments = array();
$Instruments[Chopin_favorite] = "Piano";
$Instruments[Boom_Boom] = "Drum";
$Instruments[Hendrix_thing] = "Guitar";

echo $Instruments[Hendrix_thing];             // would show : Guitar
?>
```

Creating an array using keys

Keys are used to identify a value otherwise than using an automatic index number. The key is placed between quotes followeed by => and the value.

Syntax :

```
<?php
$myArray = array("a"=>"Piano", "name"=>"John", 29, "Karen");
?>
```

Outputs to screen :

```
echo $myArray["a"];              // would show : Piano
echo $myArray["name"];           // would show : John
echo $myArray[0];                // would show : 29
echo $myArray[1];                // would show : Karen
```

Note

Values not associated to keys are automatically assigned an index number in the usual order starting with zero.

Example :

```
<?php
$myArray = array("a"=>"Piano", 29, "name"=>"John", "Karen");
                               [0]                   [1]
?>
```

Conditions

Control operators

==	Equal (type and value)
!=	Different of
<	Lower than
>	Higher than
<=	Lower or equal to
>=	Higer or equal to
and OR **&&**	AND / AND (type and value)
or OR **ll**	OR / OR (type and value)

if, else and *elseif*

```
<?php
$myNumber = 11;
if ($myNumber >= 0 && $myNumber < 10) {
// if the value of the variable is between 0 and 9 :
        echo $myNumber . " is between 0 and 9";
}
elseif ($myNumber >= 10 && $myNumber < 20) {
// if not, if the value of the variable between 10 and 19 :
        echo $myNumber . " is between 10 and 19";
}
else {
// if none of the two preceding conditions apply :
        echo $myNumber . " is higher than 19";
}
?>
```

Explanation

At first $myNumber is tested (with the condition between parenthesis) in order to know if the value is equal or higher than zero and lower than 10. If so, the echo is output to screen. Since our value is 11, this won't happen.

At this point, there are two possibilities. We could use ***else*** to directly echo an appropriate message, or we can use a second condition to test if the first one fails (***elseif***). That's what we did in our example.

Here, $myNumber is tested in order to know if the value equals or is higher than 10 or lower than 20, which is the case in our example (the value is 11). the elseif echo is then output to screen : « 11 is between 10 and 19 ».

Finally, if none of the two conditions would have been verified, if the two tests failed, else would've had been executed out-putting to screen : « 11 is higher than 19 ».

Switch

Switch is the equivalent of a condition structure that would start with an ***if*** followed by several ***elseif***.

Here is an example :

```
<?php
$name = "Morpheus";
switch ($name) {
            case "John" :                        // if John
            echo "Your name is John.";           // output John
            break;
            case "Karen" :
            echo "Your name is Karen.";
            break;
            case "Morpheus" :
            echo "Your name is Morpheus.";
            break;
            default :                            // if none of the possibilities
            echo "I don't know who you are.";    // output default message
}
?>
```

Explanation

In the example above, the output would be *« Your name is Morpheus »* because *$name* contains « Morpheus ». If you change the value of the variable $name, for anything else than the possibilities offered (John, Karen or Morpheus), the output would be *« I don't know who you are. »*.

Here is the same example with an « if » structure :

```
<?php
$name = "Morpheus";
if ($name == "John") {                           // if John
            echo "Your name is John.";           // output John
}
elseif ($name == "Karen") {                      // not John but Karen
            echo "Your name is Karen.";          // output Karen
}
elseif ($name == "Morpheus") {                   // not Karen but Morpheus
            echo "Your name is Morpheus.";
}
else {                                           // otherwise
            echo "I don't know who you are.";    // output this message
}
?>
```

While

The while condition structure allows us to create a loop repeating itself as long as a condition is true.

Syntax :

```
<?php
$number = 5;
$i = 0;
while ($i < $number) {
        echo "Our number isn't " . $i. "<br />";
        $i = $i + 1;                         // OR $i++;
}
echo "Our number is ". $i;
?>
```

Result :

```
Our number isn't 0
Our number isn't 1
Our number isn't 2
Our number isn't 3
Our number isn't 4
Our number is 5
```

Explanation :

In the example above, the value of ***$number*** is defined as 5, and the iteration variable (***$i***) is set at zero. *While* creates a loop (it repeats itself) for as long as the condition is *true* (***as long as $i is lower than 5***). When the condition becomes *false* (***when $i is equal or higher than 5***), the loop is stopped and the program continues being executed (next commands).

For

The « for » condition structure allows us to create loops repeating itself as long as a condition is true, which is useful when it is needed to count or to limit number of tries, for instance. This is an important structure which can replace ***if*** in many cases.

Syntax :

```
for ($i=0; $i < $number; $i++) {
        Some commands...
}
```

Explanation :
The condition first declares the iteration variable (***$i***) value as zero. It then verifies if the iteration variable is lower than another variable (***$number***). As long as the condition is ***true***, the iteration variable will be increased by one and the commands will be executed. When the condition is finally ***false***, the loop will stop and the rest of the program will then be executed.

Here is an example :

```
<?php
$number = 5;
for ($i=1; $i < $number; $i++) {
        echo "The number is " . $number . "<br />";
}
echo "And it has been output " . $i . " times";
?>
```

Result :

```
The number is 5
The number is 5
The number is 5
The number is 5
The number is 5
And it has been output 5 times
```

Explanation :
In the example above, the value of ***$number*** is defined as *5* and **$i** as *0*. The condition states that as long as the condition is *true* (***as long as $i is lower than 10***), the loop's commands has to be executed, than the condition tested again. Finally, when the condition becomes *false* (when ***$i is equal or higher than 10***), the loop stops and the program continues being executed.

Assignment 3 : Guessing the secret number

In a HTML document, use PHP to obtain the following :

The first variable ($number) to be declared is the one representing the number chosen by the user.

The second variable declared ($secret) is the secret number. You need for the program to generate a random number between 0 and 10.

You need messages saying whether the secret number is higher, lower or equal to the chosen number.