Include

What is 'include' for?

Including other files in your PHP code is often very useful.
The include () function does this.

Here is the syntax:

```
<? Php
   include 'file_name';
?>
```

- 'include' is the name of the function to include a file
- 'file_name' is the name of the file to be included. It can be in parenthesis or without (with a space) but the quotation marks are obligatory.

Include a text file

Imagine that we want to include a text file in a web page.
We could use the function readfile () or fgets () as seen in the last class. Or we can use the function include ()

Try this piece of code:

```
<? Php
   echo 'Using include <br>';

   include 'movies.txt';
?>
```

We see that our text file is included in our PHP script!

Include PHP code

We can also include another PHP file in our script.

Try this piece of code:

```
<? Php
   include 'hello_world.php';
   echo "Using include";
?>
```

Here is the content of the hello_world.php file:

```
<? Php
   echo "Hello, World!";
?>
```

We will see, in the rest of the course, a better utility to include a PHP file.

Include HTML

We can also include HTML!
Imagine that we develop a website with several pages (home, contact, services ...). We need a menu, common to every page.
With a traditional HTML site, you'd have to rewrite the same code for each new page.
It is very heavy, very redundant and so there is a lot of lost time!

Try this piece of code:

```
<! DOCTYPE html>
<html>
<head>
   <title> Include an HTML file </ title>
</ head>

<body>
   <header>
      <nav>
         <a href = "/ index.php "> Home </a>
         <a href="/about.php"> About us </a>
         <a href="/services.php"> Services </a>
         <a href =" / contact.php " > Contact </a>
      </ nav>
   </ header>
   <! - CONTENT OF THE PAGE ->
</ body>
</ html>
```

With PHP, we can use include to code our menu once!

To do this, create a 'header.php' file that contains our menu:

```
    <header>
         <nav>
                  <a href="/index.php"> Home </a>
      <a href = "/ about.php" > About us </a>
      <a href="/services.php"> Services </a>
      <a href="/contact.php" Contact </a>
   </ nav>
</ header>
```

Next, just modify the code from our previous example:

```
<! DOCTYPE html>
<html>
<head>
        <title> Include an HTML file </ title>
</ head>

<body>
        <? php include "header.php"; ?>
        <! - CONTENT OF THE PAGE ->
</ body>
</ html>
```

This saves us a lot of time.
Imagine that our site changes and that we have to modify our menu. Thanks to the use of Include, we just have to modify only one file (header.php) !! And not every page.

The functions

What is a function?

A function is a block / piece of code, reusable, that performs a specific action.

Functions can either return values when called, or simply perform an operation without returning a value.

PHP has a lot of built-in functions that perform different tasks.
For example, in the previous class, we could use functions to manipulate strings.

Why use the functions?

- Better Code Organization - Functions allow us to group blocks of code that perform a specific task.

- Reusability - once defined, a function can be called as many times as we want in our PHP files. This saves us time when we want to do some common tasks.

- Easy maintenance - code updates should only be done in one place.

Creating your own function

The built-in PHP functions are useful but it is often necessary to write your own functions.
We will see how to write our own functions!

First of all here is the syntax of a function:

```
<? Php
   function my_function () {

   }
?>
```

- We start by typing the word 'function' followed by a space.
- Then we have to give a name to our function. Here 'my_function'.
  It's like a variable, you can call it whatever you want.
- After the name of the function, you must add the parentheses ()
- Then finish with the brackets {}
- What your function should do is between the brackets.

Attention, however, there are some rules to respect when you create your functions:

- The name of the function must begin with an underscore or a letter (no particular number or character)
- The name must be unique
- The name must not contain of space
- The name must be explicit / describe what the function does (it is not obligatory but a good practice)

To start, we will create a function that only displays "Hello, World".

Here is the piece of code:

```
<? Php
   function hello_world () {
      echo "Hello, World!";
   }
?>
```

Try this piece of code and launch it. What is going on ?
You should realize that nothing is happening!

The reason nothing has happened is because a function is a piece of code separate from everything else. It does not run until you tell it to do it.
It's a bit like the functions we've already seen for strings, like strpos (). You can only use strpos () if you type the name and what you want PHP to find.

The same applies to your own functions - you must "tell" PHP that you want to use a function that you have written. You do this by simply typing the name of your function.

We say that we "call" a function.

Try this new version of the script:

```
<? Php
   function hello_world () {
      echo "Hello, World!";
   }

            // I call my function
   hello_world ();
?>
```

The notion of scope

In programming, there is an important notion to understand: the scope ('scope' in English).
This refers to where in our script a variable can be seen. If a variable can be seen from anywhere, it is said to have a global scope. In PHP, variables inside functions can not be seen from outside. And functions can not see variables if they are not part of the function itself.

For example, let's take this piece of code:

```
<? Php
   $ message = "Hello, World";

   Bonjour Monde();

   function hello_world () {
      echo $ message;
   }
?>
```

It's almost the same thing as the script before.
We declared a $ message variable that contains the message to display. This variable is declared outside the function.
If you try to run the script, an error will be displayed.

Also try this piece of code:

```
<? Php
   hello_world ();

   echo $ message;

   function hello_world () {
      $ message = "Hello, World";
   }
?>
```

This time, it's the opposite. We declare a variable in the function and we try to display it outside. The error will always appear.

If we need to access what is inside a variable, we need a way to get the variable into the function.
This is where the arguments come in.

Arguments

Variables can be passed to functions, so that we can do something with what they contain.
To pass a variable to a function, you must type them inside the parentheses of the function.

We repeat the previous example:

```
<? Php
   $ message = "Hello, World!";
   hello_world ($ message);

   function hello_world ($ message) {
      echo $ message;
   }
?>
```

our code:

- *Let's break downfunction hello_world ($ message)*

The name of our function is the same but we have placed a variable between parentheses. This is the variable with which we want to do something.
By typing a variable inside the square brackets, you configure what is called an argument. An argument is a variable or value that you want your function to process.

- *$ message = "Hello, World!";*
We declare a variable with the message to display.

- *hello_world ($ message);*
We call our function and we pass in argument our variable.

We could have used our function this way too:

```
<? Php

    hello_world ("Hello, World!");

    function hello_world ($ message) {
        echo $ message;
    }

?>
```

The result is the same.

A function can have several arguments, you just have to separate them with commas ','

For example:

```
<? Php

    $ message1 = "Hello";
    $ message2 = "World";

    hello_world ($ message1, $ message2);

    function hello_world ($ message1, $ message2) {
        echo $ message1. ",". $ message2;
    }

?>
```

Warning, when a function expects an argument, if you call it without arguments, you will cause an error.

Function with return value

We will now see how to create a function that returns a value.
To understand, let's take the example of a function we have already seen:

```
<? Php
   $ mail = 'simon@gmail.com'
   $ position = strpos ($ mail, '@');
?>
```

This function sends us a value: the position of our character.

Let's take the function of exercise 2. We want to calculate the addition of two numbers and know the result so that we can display it later or do something else.

We take the end of code:

```
<? Php
   add_two_numbers (1, 4);

   function add_two_numbers ($ nb1, $ nb2) {
      $ result = $ nb1 + $ nb2;
      echo $ result;
   }
?>
```

And we change it a bit:

```
<? Php
   $ addition = add_two_numbers (1, 4);
   echo $ addition;

   function add_two_numbers ($ nb1, $ nb2) {
      $ result = $ nb1 + $ nb2;
      return $ result;
   }
?>
```

The word **'return'** tells PHP to return a value. The return value is the one that you stored in the variable that follows the word return.

Here, we tell PHP to return what is stored in the variable we called $ result.

Access to the argument by reference or value

The last part to understand is how values can change, or not change, depending on the scope. Scope, if you remember, refers to where a variable can be seen in your code.

Let's take a look at this code:

```
   <? Php
      $ number = 10;

echo "Before calling my function =".
     $ number. "<br>";

      example ($ number);

      echo "After calling my function =".
          $ number;

      function function ($ number) {

         $ number = $ number + 10;
         echo "Inside my function =".
              $ number. "<br>";

      }
   ?>
```

Here we have three echo statements: one before the function is called, one inside the function and one after the function is called. Each time, we call the value of $ number.
Inside the function, we add ten to the value of the variable. When you run the code, it will print this:

```
Before calling my function = 10
Inside my function = 20
After calling my function = 10
```

Even though we have changed the value of $ number in the function, the function continues to print 10 after the call to the function! This is because the function received a copy, NOT the original.
When you give a function a copy of a variable, it's called passing a variable by value (just a copy).

The alternative is NOT to transmit a copy, but to return the original.
Make one small change to your script:

```
function example (& $ number)
```

The only addition is a character & before the variable in parentheses. This tells PHP that you want to make changes to the original and that you do not just want a copy.

When you run the script, it now displays the following:

```
Before calling my function = 10
Inside my function = 20
After calling my function = 20
```

When you make changes to the original, this is called passing the variable by reference.