college CDI

Server-side technologies (TCS)

Class 4

Retrieving contents from external files

file_get_contents()

This is most probably the most effective way to import content to a PHP document as a string. Using this function allows you to bring the content of an external file into a web document.

For instance, you could decide to have an entire web page's content written into text files. Using file_get_contents(), you import the content of those text files in specific locations. Doing this makes web document less busy and confusing and, if modifications to texts need to be made, only a text file then has to be modified.

For example, imagine a long welcome word would have been saved into an external text file names welcome_word.txt to be imported into a PHP document :

```
<html>
<head>
<title>Importing contents</title>
</head>
<body>
<h1>Welcome</h1>
<p>
<?php echo file_get_contents("welcome_word.txt"); ?>
</p>
</body>
</html>
```

One very common use of this function is to code navigation elements into an external text file and to simply import it into web documents. This way, if the navigation needs to be modified, it only needs to be modified into one file for the changes to occur throughout the web site.

```
<html>
<head>
<title>Importing contents</title>
</head>
<body>
<header>
<img src="my-logo.svg" />
<?php echo file_get_contents("main-navigation.txt"); ?>
</header>
</body>
</html>
```

include()
require()

Those to functions can be used just like file_get_contents(). Although, file_get_contents() only allows you to import static data such as text, HTML tags and CSS. This mean it can't be used to import code that needs to be executed. Importing any code needing to be executed such as variables or plugin file will become possible with include() or require().

Difference between include() and require

The two functions are the same except when it comes to failure.

require() : produces a fatal error and stops the script.
include() : produces a warning and the script continues.

Example :

```
<?php include "myFile.php"; ?>
```

OR

```
<?php require "myFile.php"; ?>
```

MyFile.php could include the following code and much more :

```
<?php
echo "<p>This is an example</p>";
?>
```

Using variables list

Using include() and require() allows to basically import programing and programing elements such as in the following example.

```
<h1>I want a new car!</h1>
<?php include "myVariables.php";
echo "I want to buy a $color $model.";
?>
```

MyVariables.php could include the following code :

```
<?php
$color="black";
$model="Mercedes";
?>
```

File handling and manipulating

Handling files is very common in different tasks when it comes to web applications. You will often have to open and process various files using different PHP functions allowing you to create, read, upload and edit files.

Be careful!

Be extremely cautious when handling and manipulating files. You may delete or modify the wrong files by accident which can cause important damages.

readfile()

The readfile() function reads a file and writes it to the output buffer. Using *echo*, it will output the content followed by the number of bytes read. Note that line breaks are not taken into accounts.

Let say we have an external file (mydata.txt) containing a simple short sentence.

```
<?php
echo readfile("mydata.txt");
?>
```

Result :

This is the content of mydata.txt. The file's content is short. It is displayed on three lines.94

fopen(), fread() and fclose()

```php
<?php
$myFile = fopen("mydata.txt", "r") or die("Unable to open file!");
echo fread($myFile,filesize("mydata.txt"));
fclose($myFile);
?>
```

Explanations :
The content of the external file is stored in a variable ($myFile). ***fopen()*** opens the file to be read. ***or die()*** outputs an error message if the file is not found. ***fread()*** reads the content of the file and ***fclose()*** closes the file which isn't useful anymore.

fopen()

fopen() function is also used to read external files, but it offers different opening modes :

r	Opens a file in read only mode. File pointer is positioned at the beginning of the file.
w	Opens a file in write only mode. Deletes the contents of the file or creates a new file if it doesn't exist. File pointer positioned at the beginning of the file.
a	Opens a file in write only mode. Preserves existing data of the file or creates a new file if the file doesn't exist. File pointer is positioned at the end of the file.
x	Creates a new file in write only mode. Returns FALSE and an error if the file already exists.
r+	Opens a file in read and write mode. File pointer is positioned at the beginning of the file.
w+	Opens a file in read and write mode. Deletes the contents of the file or creates a new file if it doesn't exist. File pointer positioned at the beginning of the file.
a+	Open a file in read and write mode. Preserves existing data of the file or creates a new file if the file doesn't exist. File pointer positioned at the end of the file.
x+	Creates a new file in read and write mode. Returns FALSE and an error if the file already exists.

fread()

fread() function reads from an open file.

```
fread($myFile,filesize("mydata.txt"));
```

Explanations :
The first parameter of fread() is the file to be read; in our example, a variable ($myFile). The second parameter specifies the maximum number of bytes to read; in our example, *filesize* signifies the entire content of the file must be read.

fclose()

fclose() function closes file. Having several unused files opened would be taking resources from the server.

```
fclose($myFile);
```

fgets() and fgetc()

fgets() function reads a single line from a file and *fgetc()* reads a single character.

```
<?php
$myfile = fopen("mydata.txt", "r") or die("Unable to open file!");
echo fgets($myfile);          // could be fgetc
fclose($myfile);
?>
```

Result :

The first line (or character) of the file is read and the pointer is positioned to the beginning of the following line.

feof()

feof() function checks if the "end-of-file" has been reached. It can be used in a loop in order to read all lines of a file :

```
<?php
$myFile = fopen("mydata.txt", "r") or die("Unable to open file!");
while(!feof($myFile)) {
        echo fgets($myFile) . "<br>";
        }
fclose($myFile);
?>
```

Creating and modifying external files

Creating a file with fopen()

As we already told, *fopen()* may also be used to create a file. If you use this function to open a file that doesn't exist, the file is created (as long as write or append modes are chosen).

```
<?php
$myFile = fopen("mycontent.txt", "w")
?>
```

Writing into a file using fwrite()

The first parameter of fwrite() consist in the name of the file to write in, and the second parameter is the string to be written.

```
<?php
$myFile = fopen("mydata.txt", "w") or die("Unable to open file!");
$txt = "Let's invite John\n";
fwrite($myFile, $txt);
$txt = "But let's not forget to invite Mary\n";
fwrite($myFile, $txt);
fclose($myFile);
?>
```

Result :

If you look into your file or if you output it, you will see the two strings have been written into the file.

Overwriting a file using *fwrite()* :

Simply go through the same process of using *fwrite()*, but with different strings, you will then notice the existing content will have been overwritten.

fseek()

fseek() moves the file pointer from its current position to a new one, forward or backward, specified by the number of bytes.

```
fseek($myFile, 0);            // brings the pointer to the beginning
```

file_put_contents()

This function writes to an external file doing exactly the same thing as using fopen(), fwrite() and fclose(). Although, since it is about twice slower than the usual structure, its use should be limited to situation when only one or a few writes are necessary.

Syntax

file_put_contents(file,data,mode,context)

file	(Required) Name and extension of the file to write in. If the file doesn't exist, it is created.
data	(Required) Data to be written in the file (string, array or data stream).
mode	(Optional) How to open and write to the file (default is overwrite). Three are available, but for now FILE_APPEND can be used to position the pointer at the end of existing data
context	(Optional) Set of options used to modify the behaviour of a stream.

```
<?php
$myData = "This is a test! ";
file_put_contents("mycontent.php",$myData,FILE_APPEND);
?>
```

Explanation :
In the preceding example, the file *mycontent.php* will be opened, the data contained in the variable *$myData* will be written into it, and *FILE_APPEND* will having it done so the new data follows the existing one.

Assignment 4 : Create a visits counter

In a HTML document, use PHP to show the number of time the page has been visited like in the following example :

« This page has received 6 visits up to now. »

You will need to create a text file which will be used as a counter and you will need to read and write in this text file each time the page is visited.