# Homework: PHP Cli

This document defines the homework assignments from the ["PHP Fundamentals" Course @ Software University](https://softuni.bg/courses/php-basics/). Please submit as homework a single zip / rar / 7z archive holding the solutions (source code) of all below described problems.

# Part I: Play with PHP

This first part will walk you through the PHP’s basic syntax by making you solve some logical basic tasks. This is very important before going to build web applications. Please, be patient.

## Configure the command line

You will first start using PHP in CLI mode (console application). This means you will receive user input through the standard input and will return result into the standard output.

PHP, like most of the modern languages, supports argument passing on application start as well as waiting for user input.

For instance, consider the following C# Console Application (it could be Java, Python or whatever language, but for the sake of the exercise a C# analogue was chosen):



The entry point (Main method) accepts an array of strings called “args”. These are the application start arguments. If you have used any console tool (e.g. git) you have already passed arguments to a CLI program. For example: “git commit” invokes the “git” executable with one argument “commit” (on the first position of the array)

Once compiled our program it will produce “cli-test.exe” file. We can execute our program using the standard shell by calling its name and passing an argument either “sum” or “subtract”.

Afterwards the program will block the execution until the user enters two lines, preferably with integers, in order to be either summed or subtracted. Then it will print to the standard output the result of the operation e.g. “ == 55” or “ == Wrong operation supplied” in case of argument different than sum or subtract



This is also possible in PHP. As it does not produce an executable that the operating system can handle, we need to invoke the interpreter in order to parse and execute the file. This means we need to invoke the “php.exe”, but chances are it is in different folder than our “cli-test.php”, which makes it very verbose.

In order to invoke the php executable from wherever without specifying the full path, we need to add its root to the environment variables called “PATH” (that’s how you can use “git” or “dir” from everywhere).

The PHP executable often resides in the WAMP/XAMPP folder/bin/php



Copy the path e.g. “C:\xampp\bin\php\” and paste it in the end of the PATH environment variable’s content.











After clicking all the OK buttons, close all the CMD windows open.

Let’s go back to our mission. To define the same C# program but in PHP.

Create a new PHP file called e.g. “cli-test.php”, by your favorite editor (PHPStorm is preferred).

When PHP is run in CLI mode, it creates a variable in the global scope called **$**argv. The variable is similar to the “args” variable that C#/Java’s Main() method accepts. It is an array of strings and it is automatically filled by the environment **with the script name as first position**. All other positions in the array are left for user input.

This means $argv[0] **is always the script name** (e.g. cli-test.php) and $argv[1]…$argv[n] **are the things that user has sent to the script as space delimited arguments**

Let’s dump the contents of the $argv array



And open the project folder in the command prompt. Then try to **run the script with the php executable** by typing

 php cli-test.php



Now let’s pass some arguments. For example: “my first cli program” by typing “php cli-test.php my first cli program”



As you can see, in position [0] is the script name and on positions [1] to [4] are the space-separated arguments.

So we can extract the operation the same way as in the above C# script:



Now we need to read the standard input. PHP relies heavily on [streams](http://php.net/manual/en/features.commandline.io-streams.php) (maybe as heavy as Java does). We can read a line from a resource handle by using the function [fgets()](http://php.net/manual/en/function.fgets.php). It might be usual for files, but other streams are also acceptable, thus we can pass the STDIN to it.



There are **two possible problems** here. If we need to read a text, **we will receive the line terminating symbols** as part of the variable contents. We can **trim** them safely. As we read numbers, **we** **can just cast or convert to number**, which will remove the redundant extra symbols.



To read the second number, we need to read the second line as well



Keep in mind that **no matter if you output to a web page or to the CLI, you can use “**echo**” construction**.

Good. Enough with the $argv ☺