# Homework: PHP Functional Programming

This document defines the homework assignments from the ["PHP Fundamentals" Course @ Software University](https://softuni.bg/trainings/1746/php-web-developmentbasics-september-2017).

## Sum Two Numbers

Write a PHP script **SumTwo.php** by using an anonymous function which is held in the variable **$sumTwo**. Use the function inside echo. In your code sum 7.1234 and 7.4321 by calling the anonymous function (hard-code the input as an argument of echo) .

See the example below:

|  |  |
| --- | --- |
| **Input** | **Output** |
| 7.1234  7.4321 | The sum is 14.5555 |

## Filter Tall People

Write a PHP script **FilterPeople.php** that uses **array\_filter()** built-in function to filter all who are taller than 1.80 (height is in meters). The input should be hard-coded inside **$people** as a two dimensional array. The output should be printed by **print\_r**.

See the example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| Two dimensional array of people | Two dimensional array of tall people |
| $people = [  ['name'=> 'John' , 'height'=> 1.65],  ['name'=> 'Peter' , 'height'=> 1.85],  ['name'=> 'Silvia', 'height'=> 1.69],  ['name'=> 'Martin', 'height'=> 1.82]  ]; | Array (  [1] => Array ( [name] => Peter, [height] => 1.85),  [3] => Array( [name] => Martin, [height] => 1.82)  ) |

## Average Height

Extend the previous script **FilterPeople.php** with a new piece of code which calculates the average height of all people by using the built-in function **array\_reduce()**. The input is the same as in Problem 2. Return the output from array\_reduce() in the variable **$averageHeight** and echo it.

|  |  |
| --- | --- |
| **Input** | **Output** |
| Two dimensional array of people | Echo the average height |
| $people = [  ['name'=> 'John' , 'height'=> 1.65],  ['name'=> 'Peter' , 'height'=> 1.85],  ['name'=> 'Silvia', 'height'=> 1.69],  ['name'=> 'Martin', 'height'=> 1.82]  ]; | Average height is 1.7525 meters |

### Hints

Be careful what you pass to the callback function. The second operand of array\_reduce() is the **current item** but it is an **array** ( like ['name'=>'John','height'=> 1.65] ) and you should **take only the height**. To find the average height you must use the **number of people**. See what built-in function is used to do that.

## Volume of Rooms

Write a script **roomVolume.php**, which calculates the volume of the rooms in an apartment. Use array\_map() built-in function to do this. See the example for input and output:

|  |  |
| --- | --- |
| **Input** | **Output** |
| An associative array holding arrays | Associative array |
| $room = [  'kithen' => ['width'=> 3,'length'=>2, 'height' => 2.4],  'living\_room' => ['width'=> 6,'length'=>4, 'height' => 2.4],  'bedroom' => ['width'=> 5,'length'=>3, 'height' => 2.2],  ]; | Array (  [kithen] => 14.4  [living\_room] => 57.6  [bedroom] => 33  ) |

### Hints

It is convenient to pass to the callback function **$current\_room** which will be an assoicative array like this: ['width'=> 3,'length'=>2,'height' => 2.4]

Be careful how you will calculate the volume in the return statement of the callback function.

## Repeat a string

Write a simple anonymous function which is contained in **$repeatString** which should be **recursive** and repeat a string **$n** times. The input should be a hard-coded value. The output should be displayed by echo. See the example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| A string like “Hello, there!” | A string for $n = 3 |
| $input = 'Hello, there! ' | 'Hello, there! Hello, there! Hello, there!' |

### Hint

You should import the function variable **$repeatString** in the inner scope to be used again from within. You must have a condition to stop the recursion, a variable holding the number of the iteration, $n and the output.

## Concatenate long strings\*

Write a script which will contain a function (might be recursive) held in the variable **$concatLongStr** that concatenates strings that are longer than $minLen characters. See the example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| An array of strings | $above = 20, Single string |
| $input = ['Hello ', 'there.', 'This is just another long string', 'that would pass the check.', ' but',' this', ' will',' not', 'pass it.']; | 'This is just another long string, that would pass the check.' |

### Hint

This looks like the previous problem but **the input is an array**. You should iterate trough the array somehow (in your function **recursively** or using a **built-in** function) and return to the output only the elements that are longer than $above. See php.net to find how to take the length of a string.

## Problem 7. Sum or subtract

Write a functional script that **either sums or subtracts values in an array**. The output should be an **array which should have the same indexes as the input array** but with the result of the operation. The two operations should be done in **two anonymous functions** held in **$opSum** and **$opSubtract**. The work should be done by an anonymous function stored in $simpleCalc which will use either $opSum or $opSubtract.

### Input

Your input is a two dimensional array. The outer array uses numeric indexes starting from 0 which increase by 1. The inner array holds 3 values (string, number, number): sum / subtract, 12.4, 123.56

### Output

The output is a single dimension array with the same indexes as the $input which will have only the result of the operation.

### Constraints

* The **numbers** should be **between 0 and 100, integer or float, else an error should be returned.**
* The error is returned as a string in the output array.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Outer array  0 Inner Array -> sum 12, 13  1 Inner Array -> subtract, 3, 3  2 Inner Array -> sum, 1, 2 | [25, 0, 3] |
| Outer array  0 Inner Array -> sum 12, 156  1 Inner Array -> subtract, 5, 6  2 Inner Array -> subtract, 1, 2 | ['error', -1, -1] |

### Hint

You should call one or another function depending on the operation. You can do this by using **switch** and using a variable called **$operation** which would contain one of the two functions.

## Problem 8. Multiply or Divide\*

Extend the code of problem 6 with two functions held in $opMultiply and $opDivide. This should be applied the same way as $opSum and $opSubtract. $opDivide may return an error when the second number is 0 (zero).

### Constraints

* The **numbers** should be **between 0 and 100, integer or float, else an error should be returned.**
* If number is out of range the error is **out\_of\_range**
* If operation is division and second number is 0 then the error Is **division\_by\_zero**
* If a certain number is not a number than the error is **op1\_not\_numeric** or **op2\_not\_numeric**

### Examples

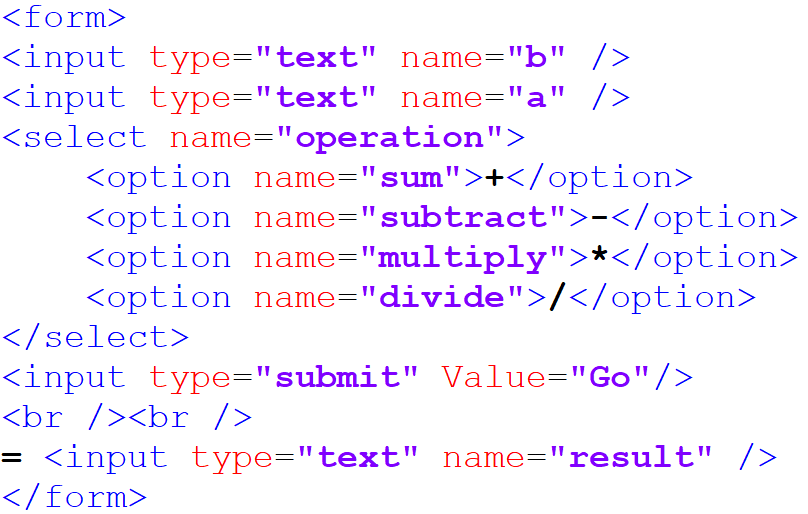
|  |  |
| --- | --- |
| **Input** | **Output** |
| Outer array  0 Inner array -> sum 12, 133  1 Inner array -> subtract, 3, 3  2 Inner array -> sum, 1, 2  3 Inner array -> divide, 100, 0  4 Inner array -> divide, 100, PP  5 Inner array -> sum, P123, 123 | ['out\_of\_range', 0, 3, 'division\_by\_zero', 'op2\_not\_numeric',  'op1\_not\_numeric'  ] |

## Problem 9. Calculator in Functional Code

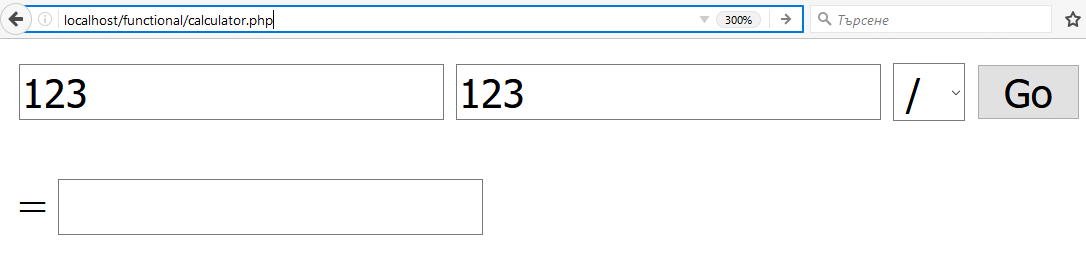
Write a script in PHP which consists of files **calculator.php (program logic)** and **calculator\_frontend.php** (HTML) which does all four operations: sum, subtract, multiply and divide. Use the anonymous functions you created in **problem 6** and **7.**

### Input

You can use following HTML in calculator\_frontend.php:



The form (localhost/functional/calculator.php) will look like:



**Include calculator\_frontend.php at the end of calculator.php !**

### Output

The output should be returned to the form in the result box. All possible errors should be displayed also in the result box. What other errors can there be.

### Constraints

* The **numbers** should be **between -1E-15 and 1E+15** (See: http://php.net/manual/en/language.types.float.php).
* Errors: **Out of range, Division By zero, Operand not numeric**

### Hints

Use the **same algorithm like in the previous problem**. Try to **maintain a functional style** of programming using anonymous functions. You can create an anonymous function held **in $opExecute which can return a function** - $opSum, $opSubtract and etc. depending on an input argument received by the form.

Also you can create a function to handle the errors and store it in **$opError**. This can be passed back as a variable to $opExecute if an error occurs.

To get the data from HTML you can use the **$\_GET** super global array (<form **method="get"**> in your HTML)

## Problem 10. Extended Calculator\*

Extend the calculator **to maintain its state** by putting the **result back as the first number** if there is no error.

Add an operation called **Average** which would **calculate the average of two numbers** by using a new anonymous function.

Add an **operation n! (n factorial)** on the first number which should be done recursively. Research the internet about n! and how you can code it in PHP. Simply put, this is the multiplication of all numbers up to N like this:

1\*2\*3\*4\* 5 \* 6 for n = 6

## Problem 11. Create a power matrix\*

Create a matrix out of a single dimensional array in which **every new row** consists of the **first row** on the power of the Number of row. Use a **single recursive function** to do this.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 2 4 | 1 2 2 4  1 4 4 16  1 8 8 64  1 16 16 256 |
| 1.3 2.4 | 1.3 2.4  1.69 5.76 |
| 1 2 3 | 1 2 3  1 4 9  1 8 27 |

### Input

An array of numbers starting with index = 0.

### Output

An array of arrays starting with index 0, and each subarray starting also with index = 0. Do not use associative arrays.

### Constraints

* Up to 5 **numbers in a row**
* All numbers should be positive
* All numbers should be integer or float < 10

## Problem 12. Create a Fibonacci matrix\*\*

Create a matrix out of a single dimensional array in which **every new row** consists of the **previous row** in which **every new number** is the **sum of the previous two numbers in the column**. The zero row is full with 1. Use a recursive anonymous function to do this.

Create a HTML form with three fields: text field – the numbers divided by a comma (,), a submit field called “Calculate” and a table in HTML to display the matrix. Divide the code in back-end and front-end separated files.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 5 3 | **1 1 1 1 +**   1. **2 5 3 =** 2. **3 6 4** 3. 5 11 7 |
| 1 2 | 1. 1   1 2 |
| 1 2 3 4 5 | 1 1 1 1 1  1 2 3 4 5  **2 3 4 5 6 +**  **3 5 7 9 11 =**  **5 8 11 14 17** |

### Input

An array of numbers starting with index = 0.

### Output

An array of arrays starting with index 0, and each subarray starting also with index = 0. Do not use associative arrays.