# PHP Functional Programming - Lab

This document defines a set of tasks to be done as a part of the PHP Functional Programming lecture’s exercises.

# Part I: Higher Order Functions

## Filter The Old Dogs

Write a simple program that receives as input a two dimensional array of data about animals: dogs and cats. See the example below. Use a closure and the built in function **array\_filter()** to filter all dogs which are at age larger than 10 years.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Two dimensional array of dogs | Two dimensional array of old dogs |
| $animals = [  [ 'name' => 'Waffles', 'type' => 'dog', 'age' => 12],  [ 'name' => 'Fluffy', 'type' => 'cat', 'age' => 14],  [ 'name' => 'Spelunky', 'type' => 'dog', 'age' => 4],  [ 'name' => 'Hank', 'type' => 'dog', 'age' => 11],  ]; | Array (  [0] => Array ( [name] => Waffles, [type] => dog, [age] => 12 ),  [3] => Array( [name] => Hank, [type] => dog, [age] => 11 )  ) |

### Hint

**array\_filter()** has two input parameters. The first is an **input array** which we well filter. The second is a **function that does a comparison operation**. It this function returns true then the element of the array stays else it is filtered out.

## Filter the young dogs

Write a new filtering function and store it in the variable **$youngDogs**. Filter all dogs younger than 11 years.

See the example below:

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Two dimensional array of dogs | Two dimensional array of young dogs |
| $animals = [  [ 'name' => 'Waffles', 'type' => 'dog', 'age' => 12],  [ 'name' => 'Fluffy', 'type' => 'cat', 'age' => 14],  [ 'name' => 'Spelunky', 'type' => 'dog', 'age' => 4],  [ 'name' => 'Hank' , 'type' => 'dog', 'age' => 11],  ]; | Array(  [2] => Array( [name] => Spelunky, [type] => dog, [age] => 4  ) |

### Hint

### Your function should look something like this:

### How would the arguments of array\_filter() change?

## 1.2. Change the year in young dogs filter

Add an argument to **$youngDogs** which should specify which dog should be considered as young. Then change your array\_filter code and filter dogs less than 9.

## 1.3. Write your own version of array\_filter()\*

Write your own version of array\_filter and store it in the variable **$filter**. Wrap all in a new function stored in $dogsOutput, invoke it and filter all young dogs less than 8 years using the new version of $youngDogs.

## 1.4. Write a wrapper of $filter called $filterDogs\*\*

Write a wrapper which will use **$filter** which will have one more argument: a text string 'young' or 'old' which should invoke either **$youngDogs** or **$oldDogs**.

# Part II: Practice with PHP built-in functions

# 2. Body Mass Index Calculator

Calculate the Body Mass Index (BMI) which is **weight, kg / (height, m \* height, m)**. The input array consists of an subarray of name, weight and a height. The output should be single dimension array. See the example below:

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Two dimensional array of people | One dimensional array of BMI's |
| $people = [  [ 'name' => 'John' , 'weight' => 69, 'height' => 1.69 ],  [ 'name' => 'Peter' , 'weight' => 85, 'height' => 1.68 ],  [ 'name' => 'Ivan' , 'weight' => 75, 'height' => 1.72 ],  [ 'name' => 'Mitko', 'weight' => 95, 'height' => 1.70 ]  ]; | Array (  [0] => 24.158817968559  [1] => 30.116213151927  [2] => 25.351541373716  [3] => 32.871972318339  ) |

Do not use a loop construct (foreach, for or while) and maintain a functional programing style of coding. Use **array\_map()** to do the calculation.

# Hint

Use a **syntax like array\_filter**. The callback function will return the BMI. Pass **$people** to the anonymous callback function

# 2.1.Find the average BMI

Find the average BMI as a single number. Use array\_reduce for this. Return the value in $bmiAvg. Wrap the new code in a function and put it into a variable **$bmiCalcAvg**. Echo or print\_r the result ($bmiAvg) at the end.

# Hint

Go online to <http://php.net/manual/en/function.array-reduce.php> to see the exact parameters put into the callback of array\_reduce. There are two of them:

* $carry, which holds the iterative value
* $item, which holds the current item

# 2.2.Find the average BMI above some value\*

Extend the code of the function **$bmiCalcAvg** to include only BMI's above a particular value. To import the variable in the scope use the **'use'** keyword inside the callback.

# 2.3.Extend $bmiCalc to hold people's names\*

Extend the code of the function in **$bmiCalc** to return an array that holds the names of the people and be two dimensional. See the example below:

|  |  |
| --- | --- |
| **Input** | **Output** |
| Two dimensional array of people's names, weight, height | Two dimensional array of people's names and BMI's |
| $people = [  [ 'name' => 'John' , 'weight' => 69, 'height' => 1.69 ],  [ 'name' => 'Peter' , 'weight' => 85, 'height' => 1.68 ]  ]; | Array (  [0] => array('name' => 'John', 'bmi'=>24.158817968559),  [1] => array('name => 'Peter', 'bmi'=>30.116213151927)  ) |

Use a simple **foreach / for** loop for this. This would be a mixed style of programming which we can afterward do in a functional style.

# 2.4.Extend $bmiCalc: functional \*\*

Rewrite your code from 2.3. and use a functional programming style now instead a loop construct. The output is the same as in 2.3.

# Hint

Use the result from **array\_map()** and combine it with the data from **$people**. If you have written the code with a for or foreach now you have to replace this with a recursive function to do the loop. The function needs some arguments:

$i – to hold the value which we iterate

$n – to hold the number of array items

$out – to hold the output array