# More Exercises:

# Functions, Objects and Classes

This document defines a set of tasks to be done as a part of the "[PHP Web Development Basics - 2018](https://softuni.bg/trainings/2163/php-web-development-basics-september2018#lesson-9663).

# Part I: Objects and Classes

## Define a class Car

Define a class **Car** with fields for **brand**, **model** and **year**. Make a three instances of the class.

## Add a Constructor

1. Add constructor to the **Car** class from the last task
2. It should accept two arguments as properties - brand and model.

## Add methods

* + - 1. Add a set method for setting a property **year**
      2. Validate input in а new method
      3. Add get method for all properties

## Make an instances

Using the Car class, write a program that reads from the console **4** lines of car information. Every line contains brand, model and year, separated by space. Make a list of objects of class Car. On finish - prints all cars, **sorted in alphabetical order by brand, model and year**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Nissan X-trail 2007  Renault Scenic 2001  Audi A6 2001  Renault Clio 1995 | Audi,A6  Nissan,X-trail,2007  Renault,Clio,1995  Renault,Scenic,2001 |

## Add sub class

Using the previous problem, add a second class that define car model extra details (engine, number of seats, horsepower, etc.). Add this information as property of main class and populate it via method. Make one instance, populate all properties and dump the object.

## Rewrite a code

Rewrite procedural/functional code to OOP style

|  |
| --- |
| <?php  function math\_sum($a,$b){  return $a+$b;  }  function math\_div($a,$b){  math\_check\_if\_zero($a);  math\_check\_if\_zero($b);  return $a / $b;  }  function math\_check\_if\_zero($x){  if($x == 0){  echo "division by zero is not possible";  exit;  }  }  echo math\_sum(2,3);  echo math\_div(1,2); |

## Create Anonymous Object

1. Create Anonymous Object with 10 properties by your choice and populate with values
2. Print all properties with foreach like - {name}->{value)

# Part II: First-class Functions – Optional

# // Search information about built-in 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.

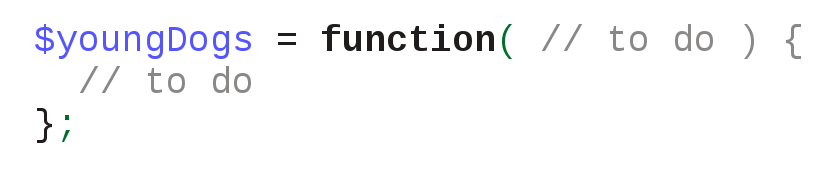
See the example below:

### 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?**

## 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.

## 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.

## Create Anonymous Function

## 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 III: Practice with PHP built-in functions

## 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

## 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

## 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.

## 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.

## 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.

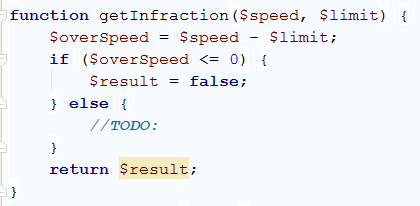
# Hint

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



We calculate the difference between the current speed and the limit – if it’s negative or zero, this means the driver is within the rules and we return false, and in any other case, return the infraction as a string and store the result of the operation in a variable.

