C1 - Piscine PHP

C-COD-150

Day 02

Stay On Command !

1.1

{EPITECH.}

Day 02

repository name: pool_php_d02
repository rights: ramassage-tek

- Your repository must contain the totality of your source files, but no useless files (binary, temp files, obj files,...).
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

+ EXERCISE 01 (1PT)

Turn in: pool_php_d02/ex_01/ex_01_1.php
pool_php_d02/ex_01/ex_01_2.php
Prototype: my_included_putstr(mixed $str);
Restrictions: None.

In ex_01_2.php, create a function, my_included_putstr which will be an echo of the string given as parameter.
In ex_01_1.php, include the ex_01_2.php and call the function my_included_putstr giving it the first argument from command line.

+ EXERCISE 02 (2PT)

Turn in: pool_php_d02/ex_02/ex_02.php
Prototype: my_create_map(mixed ...$array);
Restrictions: None.

Create a function which takes a variable number of arrays as arguments and returns them in the form of a map.
The arrays must be valid. They must contain at least two values, you must use the first one as a key for the map and the second one as the associated value (ignore all other values).
If the parameters are not valid, you must print the following message: "The given arguments aren't valid." followed by a line feed. And return NULL.
Example:

Terminal

```
<?php
require("ex_02.php");

$array1 = ["pi", 3.14];
$array2 = ["answer", 42];

var_dump(my_create_map($array1, $array2));
```

will produce the following output:

Terminal

```
array(2) {
     ["pi"] =>
     float(3.14)
     ["answer"] =>
     int(42)
}
```

+ Exercise 03 (2pt)

Turn in: pool_php_d02/ex_03/ex_03.php

After a hacker attack, the Facebook database was compromised. We have determined that the hashed password of the user named "admin" is: "21232f297a57a5a743894a0e4a801fc3".
Create a file named "ex_03.php" and write inside it the decrypted password followed by a newline.

The algorithm used is MD5

+ Exercise 04 (2pt)

Turn in: pool_php_d02/ex_04/ex_04.php
Prototype: array my_password_hash(string $password);
Prototype: bool my_password_verify(string $password, string $salt, string $hash);

Write a function my_password_hash that takes as parameter a password in plain-text and returns an array containing the hashed password in MD5, as well as the "salt" used for hashing. The salt must be different on each call.
Then write a function my_password_verify that takes as parameter a password in plain-text, a salt and a hashed password. This function must return true or false, depending on whether the hashed password corresponds to the plain-text password with the associated salt or not.
The array returned by the function my_password_hash has to look like this:

Terminal

```
$array = [
    "hash" => "XXXXX",
    "salt" => "12345",
];
```

+ Exercise 05 (2pts)

Turn in: pool_php_d02/ex_05/ex_05.php
Prototype: my_add_elem_map(mixed $key, mixed $value, mixed &$map);
Prototype: my_modify_elem_map(mixed $key, mixed $value, mixed &$map);
Prototype: my_delete_elem_map(mixed $key, mixed &$map);
Prototype: my_is_elem_valid(mixed $key, mixed $value, mixed &$map);

Create three functions which can add, modify and delete a value in a map with its key. The functions will return the map after changes.
Create another function to check if the value of a certain key is the same than the value given in parameters, and if the key exists.
If the value and the key exist, return true, otherwise, return false.
Here is the error message (even for my_is_elem_valid): "You have to give good parameters." followed by a new line.

+ Exercise 06 (2pt)

Turn in: pool_php_d02/ex_06/ex_06.php
Prototype: my_create_continent(mixed $continentNameToAdd, mixed &$worldMap);
Prototype: my_create_country(mixed $countryNameToAdd, mixed $continentName, mixed &$worldMap);
Prototype: my_create_city(mixed $cityNameToAdd, mixed $postalCodeOfCityToAdd, mixed $countryName, mixed $continentName, mixed &$worldMap);
Prototype: get_country_of_continent(mixed $continentName, mixed $worldMap);
Prototype: get_cities_of_country(mixed $countryName, mixed $continentName, mixed $worldMap);
Prototype: get_city_postal_code(mixed $cityName, mixed $countryName, mixed $continentName, mixed $worldMap);

In this exercise, we will create an awesome world map. We will create a map, in a map, in a map. Hold on, it's not so complicated:
- We will have cities (name and postal code) in countries.
- We will have countries (name and cities) in continents.
- We will have continents (name and countries) in world.
So, we'll create some functions to create and get these maps.
If a continent, a country or a city is not found, you have to write the error message: "Failure to get continent.", "Failure to get country.", "Failure to get city" followed by a new line and return NULL.

Example:

Terminal

```
<?php
$map = array();
my_create_continent("Europe", $map);
my_create_country ("France", "Europe", $map);
my_create_city("Marseille", "13000", "France", "Europe", $map);
var_dump($map);
```

will produce the following output:

Terminal

```
array(1) {
     ["Europe"] =>
     array(1) {
          ["France"] =>
          array(1) {
               ["Marseille"] =>
               string(5) "13000"
          }
     }
}
```

+ EXERCISE 07 (2PT)

Turn in: pool_php_d02/ex_07/ex_07.php
Prototype: int my_division_modulo(mixed $fistNumber, mixed $operChar, mixed $secondNumber);

Write a function which will execute division and modulo operations, but when the given parameters are not good, the function will throw an exception with the message "The given arguments aren't good." with a new line.

Terminal

```
<?php
require("ex_07.php");

echo my_division_modulo(3, '/', 4) . "\n";
echo my_division_modulo(2, '/', 1) . "\n";
try
{
echo my_division_modulo(3, '+', 1) . "\n";
}
catch (Exception $err)
{
echo $err->getMessage();
}
try
{
echo my_division_modulo(2, '/', 0) . "\n";
}
catch (Exception $err)
{
echo $err->getMessage();
}
```

will produce the following output:

Terminal

```
0.75
2
The given arguments aren't good.
The given arguments aren't good.
```

+ EXERCISE 08 (2PTS)

Turn in: pool_php_d02/ex_08/ex_08.php
Prototype: array my_order_class_name(midex …$obj);

Write a function my_order_name that takes as parameter a variable number of arguments. It must return a two dimension array containing the objects passed as parameter, sorted by the length of the type name, then by alphabetical order of type name. If an argument is an object, the complete qualified class name will be used for sorting. Each type of class name must be unique.

Terminal

```
<?php
require("ex_08.php");

classMyClass
{

};

$args = [
     true,
     76,
     false,
     12.5,
     "Coucou !",
     [1, 2, 3],
     new MyClass(),
     NULL
];

print_r(my_order_class_name(...$args));
```

will produce the following output:

Terminal

```
Array
(
     [0] => Array
          (
               [0] => NULL
          )

     [1] => Array
          (
               [0] => array
          )

     [2] => Array
          (
               [0] => double
               [1] => string
          )

     [3] => Array
          (
               [0] => boolean
               [1] => integer
               [2] => MyClass
          )

)
```

+ EXERCISE 09 (2PTS)

Turn in: pool_php_d02/ex_09/ex_09.php
Prototype: bool check_array_sum(Array $numbers);

Create the function check_array_sum that takes as parameter an array of numbers and returns true if the sum of certain elements of the array can be equal to the highest value of the array. If there are no combination that can add up to the highest element of the array, the function returns **false**.
Example:

Terminal

```
<?php
require("ex_09.php");

$arr = [4, 6, 23, 10, 1, 3];

if (check_array_sum($arr) == true)
     echo "4 + 6 + 10 + 3 == 23";
```

will produce the following output:

Terminal

```
4 + 6 + 10 + 3 == 23
```

The array will not be empty and may contain negative values.

+ EXERCISE 10 (3PTS)

Turn in: pool_php_d02/ex_10/ex_10.php
Prototype: string simplify_polynomial_expression(string $expression);

Implement the function simplify_polynomial_expression that takes as parameter a string representing a polynomial expression and simplifies it. The string $expression will contain a single letter "x", "y", "z", etc … There will be no spaces in the output or input. The output will be sorted from the greatest to the lowest degree.
Example:

Terminal

```
<?php
require ("ex_10.php");

echo simplify_polynomial_expression("(2x^2+4) (6x^3+3)");
```

will produce the following output:

Terminal

```
12x^5+24x^3+6x^2+12
```