# Homework: PHP Working with Forms and Sessions

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.

### WARNING!

In all tasks use **point** as a decimal separator and do not use comma or any other styling as thousands separator.

Quality OOP styled code is required. Proper data validation is also required and use PHP sessions for any persistent data storage where necessary. You can use cookies as well in combination with PHP sessions, but not in isolation.

## Print Tags

Write a PHP script **PrintTags.php** which generates an HTML **input text field** and a **submit button**. In the text field the user must enter different tags, separated by a comma and a space (", "). Output should be in ordered list by the order of appearance in the input. Semantic HTML is required. Styling is not required.

### Example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho, homework, homework, exam, course, PHP |  |

## Most Frequent Tag

Write a PHP script **MostFrequentTag.php** which generates an HTML **input text field**, **submit button** and **clear button**. In the text field the user must enter **different tags**, separated by a comma and a space **(", ")**. When the information is submitted, the script should generate a **list of the tags**, sorted **by** **frequency**. Then you must print: **"Most Frequent Tag is: [most frequent tag]"**. You should be able add **N** number of rows through multiple form submission, every time the input should be added up to the previous one and the list should be updated. When you press the **Clear** button the list should be cleared and the next input you add will generate a new list. Use pure PHP Solution without JavaScript. Semantic HTML is required. Styling is not required. Example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho, homework, homework, exam, course, PHP |  |

## Calculate Interest

Write a PHP script **CalculateInterest.php** which generates an HTML page that contains:

* An input text field to hold the **amount of money**
* Radio buttons to choose the **currency**
* An input text field to enter the **compound annual interest** **amount**
* A dropdown menu to choose the **period of time**
* A **submit button**

When the **information is submitted**, the script should print out the **amount of money** you will have after the selected period, rounded to 2 decimal places. Semantic HTML is required. Styling is not required.

### Example:

|  |  |  |
| --- | --- | --- |
| **Preview** | **Input** | **Output** |
|  | 1000  USD  12  6 months | $ 1061.52 |

***Hint:*** How to calculate **compound interest**? You may read the [Wikipedia](http://en.wikipedia.org/wiki/Compound_interest) article or see the **example below**.

***Explanation of the example:***

We have Compound Annual Interest of 12%. This makes 1% interest per month. This means that each month for a 6 months’ period of time we accrue interest of 1% on the current amount of money. Given the input from the example, the result would be:

|  |  |  |
| --- | --- | --- |
| **Month** | **Calculations** | **Amount of Money** |
| 1st | 1000 \* 101% | 1010 |
| 2nd | 1010 \* 101% | 1020.1 |
| 3rd | 1020.1 \* 101% | 1030.301 |
| 4th | 1030.301 \* 101% | 1040.60401 |
| 5th | 1040.60401 \* 101% | 1051.0100501 |
| 6th | 1051.0100501 \* 101% | 1061.520150601 |

After the **6th month** we have **1061.520150601 USD** in our account. We **round the result** and add the symbol **"$"** for USD. The output is **$ 1061.52**.

## HTML Tags Counter

Write a PHP script **HTMLTagsCounter.php** which generates an HTML form like in the example below. It should contain a **label**, an **input text field** and a **submit button**. The user **enters HTML tag** in the input field. If the tag is valid, the script should print “**Valid HTML tag**!”, and if it is invalid – “**Invalid HTML Tag!**”. On the second line, there should be a **score counter**. For every valid tag entered, the score should increase by 1. Use pure PHP Solution without JavaScript.

|  |  |
| --- | --- |
| **Correct Tag** | **Incorrect Tag** |
|  |  |

## CV Generator

Write a PHP script **CVGenerator.php** which generates an HTML form like in the example below. When the **information is submitted** (via Generate CV), the script should print out a **simple table-like CV**. Semantic HTML is required. Styling is not required. Example:

|  |  |
| --- | --- |
| **Input** | **Output** |
|  |  |

### Constraints

You must **validate the data**, entered by the user, **in the PHP script** as follows:

|  |  |
| --- | --- |
| **Input Fields** | **Server Validation in the PHP Script Criteria** |
| First Name, Last Name, Language | Only letters  Between 2 and 20 symbols |
| Company Name | Letters and Numbers  Between 2 and 20 symbols |
| Phone Number | Numbers and “+”, “-”, “ ” |
| Email | Letters, Numbers  Only one “@”, only one “.”  Example: **example@example.com** |

If you are familiar with JavaScript the buttons **"Remove Language"** and **"Add Language"** must **dynamically** **add /remove** aninput field for language and a selection for level of knowledge. The same goes for **Programming Languages** and **Languages from Other Skills**. Otherwise create static **N** number of fields for them.

### Example:

|  |  |
| --- | --- |
| **Action** | **Result** |
| Default |  |
| When we click "Add Language" 2 times |  |
| When we click "Remove Language" 1 time |  |

## Student Sorting

Remember that task from the first lesson. Let's update it a bit. Write a PHP program **StudentSorting.php** that receives data about several students from an **input form** (**first name**, **last name**, **email** and **grade**) and prints it as an HTML table. If you a familiar with JavaScript the user should be able to dynamically **add**/**remove** entries via the **+**/**-** buttons. Otherwise create **three** static rows that should be submitted on every input. The data can be sorted by 4 criteria: **First name**, **Last name**, **Email** and **Exam score**. The sorting can be done in **ascending**/**descending** order. The result should be printed as a table. The average exam score should be printed on the last row. (See the example below.). Styling the page is *optional*. Semantic HTML is required. Use pure PHP Solution without JavaScript for the generated result table.

|  |  |
| --- | --- |
| **Form** | |
|  | |
| **Result** | |
|  | |

## Paginate Students

Update the last task. Render max 5 students per page and two hyperlinks under the table – “Previous” and “Next”. The “Previous” hyperlink should not exist when there are no previous pages (e.g. we are on the first page) and the “Next” hyperlink should not exist when there are no next pages (e.g. when we are on the last page).

After clicking on “Next” hyperlink, the next chunk of 5 students is rendered. After clicking on “Previous” the previous chunk of 5 students is rendered. Average exam score should be on every page as a last row.

## \*Paginate Students (2)

Decorate more the last task. Between the hyperlinks “Previous” and “Next” render hyperlinks with pages. Display exactly 3 hyperlinks with pages. But be careful! In some ways there might not be 3 pages until the end of the collection, so you need to render the previous pages. The only exception of the rule is when there aren’t 3 pages at all.

For instance, if we have 16 students, we will need 4 pages. This means on the first request we will render:

[1] [2] [3] [Next]

The red color means the page is the current page and cannot be clicked

The green color means the page is available for clicking

When clicking on “Next” or “2” the pager will be like:

[Previous] [2] [3] [4] [Next]

But when clicking on “3” there is no fifth page, so in order the slider to show exactly 3 pages, we need to make it remain the same way, just make “2” clickable in favor of “3”:

[Previous] [2] [3] [4] [Next]

The next step is to go to the last page. Clicking on “4” will make the slider look like:

[Previous] [2] [3] [4]

The exception of the rule is when we have for example 9 students. The only thing we can do here is to render 2 pages (which is normal, no results for 3 pages ☺):

[1] [2] [Next]

[Previous] [1] [2]