Exercise 3 – PHP Handling Input

Weighting: 2%

In this exercise we are going to write and modify existing simple PHP applications.

Task 1 – Github lecture examples

Clone the lecture examples from our Github repository: *https://github.com/Griffith-ICT/WebDev-Examples*

Hint: See the instructions presented in the lecture video.

Task 2 – Personal Details Application (peer review exercise)

Examine Personal Details Application (in WebDev-Examples/week3) before your Week 4 workshop. During week 4 workshop, explain to your reviewers how the forms are processed by PHP, i.e. from form generation to submission, processing, and displaying results. Also show your reviewers that you can call show_details.php by entering parameter into the URL instead of using the form. Ask your reviewers to provide you with feedback on your explanation.

Hint: you can refer to lecture video if you need help.

Task 3 – Improved Factorise Example

Modify the original Factorise example by implementing the following:

- After performing a factorisation, a form for a new factorisation is displayed at the bottom of the same page.
- After performing a factorisation, the form shows the value last entered in the input field, so that it can be easily modified.
- If an empty, invalid (e.g. 'abc'), or out of range (e.g. 1) number is entered an appropriate error message is displayed *on the same page* instead of the factorisation.

A working solution of what you need to implement can be found here:
http://www.ict.griffith.edu.au/teaching/WP/Examples/improved-factorise/

Hints:

1. If the user input is invalid, instead of printing an error message immediately, assign the error message to a PHP variable, **$error**. Later, if the variable **$error** is not empty, compute the number's factors and print the factorisation as before; otherwise print the value of **$error**.
2. To display the value of variable **$number** in the form's input field, use the value attribute as follows:

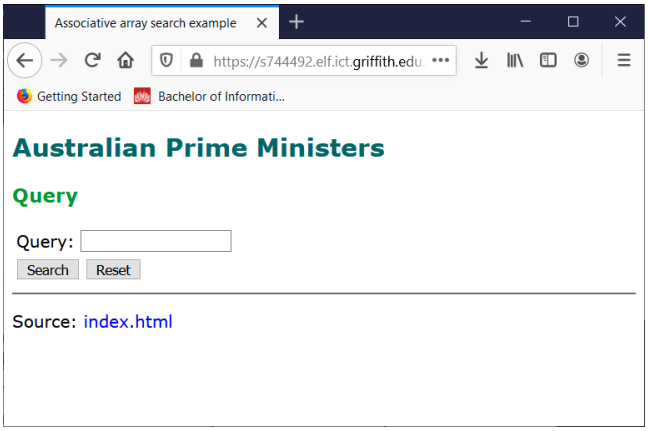
```
<input type="text" name="number" value="<?= $number ?>" >
```

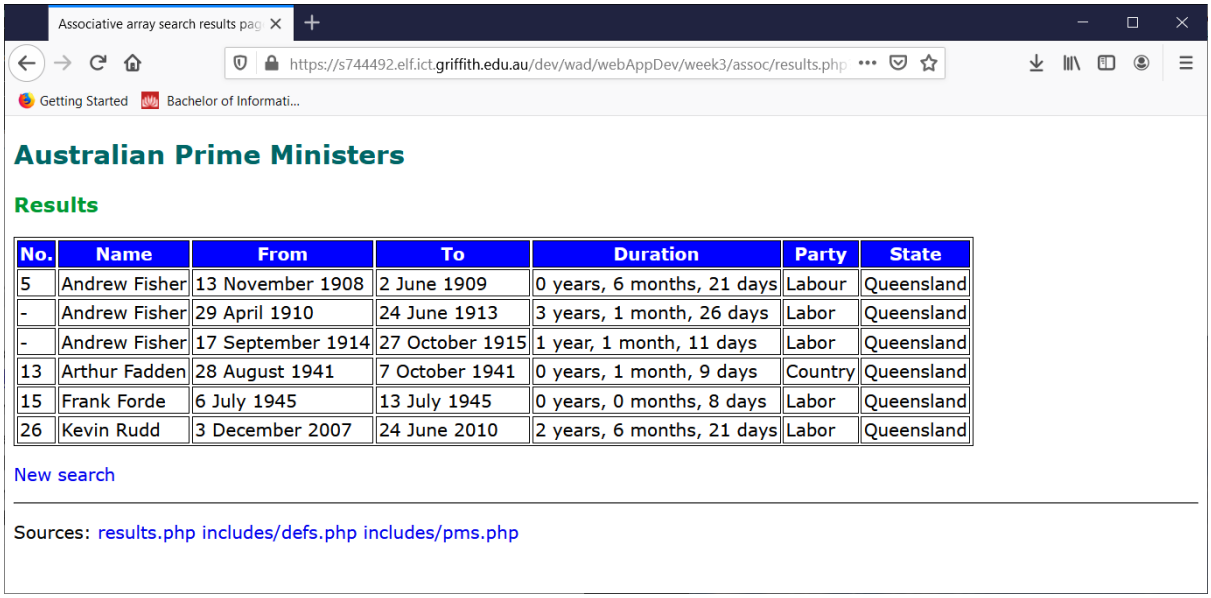
3. Refer to lecture video if you need help.

Task 4 - Prime Ministers Example

Test out the Prime Ministers Example in the "**assoc**" to ensure that it works.

The Prime Ministers Example has three search form fields. Modify the form so that it only has a single text field which supports searching for Name, Year, and State. Once the form is submitted it will display all Prime Ministers whose name **or** year **or** state matches that input string.





Hint:

- To do the search, loop through the list of Prime Ministers once only. Test whether each Prime Minister has a name or year or state that matches the given query, and if so add the Prime Minister to an initially empty result list. Finally, return the result list.
- Refer to the demonstration video if you need help.

Task 5 – Input validation for the original PM example

Implement input validation for the original PM example (i.e. the one with 3 inputs). If the input is invalid, then redisplay the error message and the search form, otherwise display the query **and** the search result.

The input is invalid if:

1. All input fields are empty, then the error message should be: “At least one field must contain value.”
2. If the input contains only year, and year is not an integer, then the error message should be: “Year must be a number.”

Task 6 – Different types of validation

Explain to your peers the difference between client-side and server-side validation? Are the validations we did in the above tasks client or server side validation? Which type of validation is more important and why?

Task 7 – Save to file

Further modify your solution to Task 3 by appending each factorisation (12 = 2 . 2 . 3) to a text file so you can display the list of *all* previous factorisations below the current one.