**[IS113] Extra Exercises - Weeks 4-5 - Form Processing**

**Objectives**

* To master the concepts of HTML form elements and form processing
* To be able to differentiate between POST and GET
* To practice on retrieving form input values via POST and GET

**Instructions**

* Questions with no asterisk mark are easy peasy.
* Questions marked with \* are slightly challenging.
* Questions marked with **\***\* are challenging.
* Questions marked with \*\*\* are very challenging.

**Download**

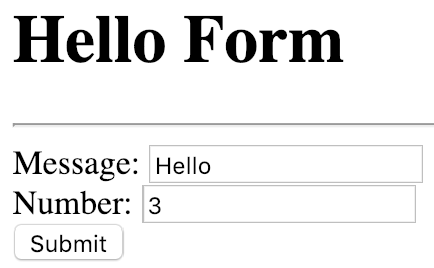
* **Resources**: Click [here](https://smu.sg/AY20-21-is113-extra-exercises-wk4-resources)
* **Solutions:** Click [here](https://smu.sg/AY20-21-is113-extra-exercises-wk4-solutions)

**NOTE:** If you spot any mistakes/errors in the questions, please contact your instructors by email and state the issues. We will try to address it as soon as possible.

**Question 1 - Hello**

Go to **hello** directory**.** Complete the following **Parts A** and **B** inside **sent.php** file.

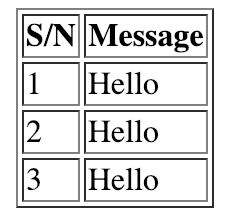
**hello.html** page has TWO form input fields (TEXT).



**Part A (\*)**

Complete **sent.php** file so that it takes the **Message** and prints it **Number** number of times.

Given the above input, **sent.php** page displays the following **HTML Table**.



**Part B (\*\*)**

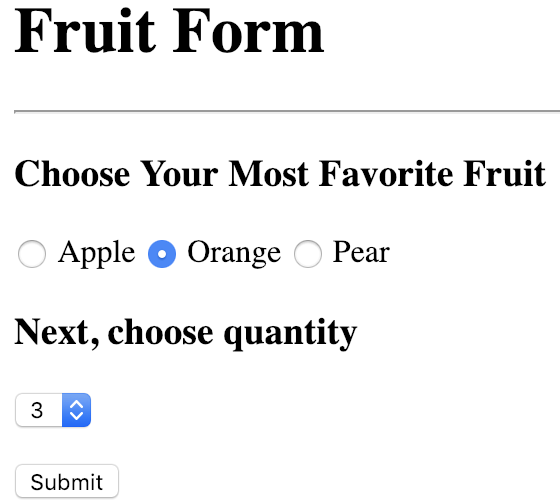
Modify **sent.php** file so that it performs **Form Input Validation**.

|  |  |
| --- | --- |
| **hello.html** | **sent.php** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Question 2 - Fruits**

Go to **fruits** directory**.** Complete the following **Parts A** and **B** inside **sent.php** file.

**fruit.html** page allows the user to select a **fruit** and **quantity**.



**Part A (\*)**

Complete **sent.php** file so that it takes the **fruit’s name** (e.g. “orange”) and displays the selected fruit’s image file **Quantity** number of times. Given the above input, **sent.php** page displays the following:



The image files are located in **/is113/extra4/fruits/images/**.

**Part B (\*\*)**

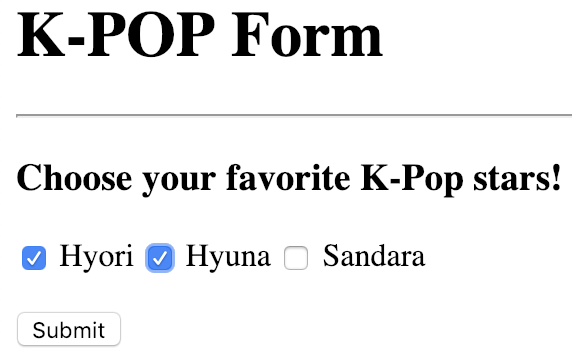
Modify **sent.php** file so that it performs **Form Input Validation**.

|  |  |
| --- | --- |
| **fruit.html** | **sent.php** |
|  |  |
|  |  |
|  |  |

**Question 3 - K-Pop**

Go to **kpop** directory**.** Complete the following **Parts A** and **B** inside **sent.php** file.

**kpop.html** page allows the user to select K-Pop celebrity names.



**Part A (\*)**

Complete **sent.php** file so that it takes the user input (celebrity names) and displays their images. Given the above input, **sent.php** page displays the following:



The image files are located in **/is113/extra4/kpop/images/**.

**Part B (\*\*)**

Modify **sent.php** file so that it performs **Form Input Validation**.

|  |  |
| --- | --- |
| **kpop.html** | **sent.php** |
|  |  |
|  |  |
|  |  |
|  |  |

**Question 4 - Login**

Go to **login** directory**.** Complete **Part A** below.

**login.html** page allows the user to key in **username**, **password** and retype the password (**confirm password**).



**Part A (\*\*)**

Page **sent.php** is **broken**. Fix the code so that it performs **Form Input Validation** as shown below:

|  |  |
| --- | --- |
| **login.html** | **sent.php** |
| Password: trump123  Confirm Password: trump123 |  |
| Password: trump123  Confirm Password: trump123 |  |
| Password: trump123  Confirm Password: abcdefgh |  |
| Password: <empty>  Confirm Password: <empty> |  |
| Password: trump123  Confirm Password: <empty> |  |
| Password: trump123  Confirm Password: trump123 |  |

**Question 5 - Asterisk**

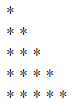
Go to **asterisk** directory**.** Complete the following **Parts A** and **B**.

Both pages **ex1.html** and **ex2.html** have ONE form input field (TEXT).



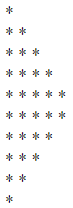
**Part A (\*\*)**

Complete **ex1.php** file so that it takes the **Num** (Integer value) and prints **Num** lines of one or more asterisks. Given the above input **5**, it prints ONE line of 1 asterisk, ONE line of 2 asterisks, ONE line of 3 asterisks, ONE line of 4 asterisks and ONE last line of **5** asterisks. Given the input of ZERO **Num**, **ex1.php** would not print anything.



**Part B (\*\*)**

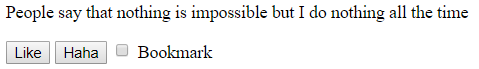
Complete **ex2.php** file so that it takes the **Num** (Integer value) and prints **Num** lines of one or more asterisks. Given the above input **5**, it is to print the following shape. Given the input of ZERO **Num**, **ex2.php** would not print anything.



**Question 6 - Like**

Go to **like** directory**.** Complete the following **Part A**.

Page **like.php** has TWO (2) submit buttons and ONE checkbox.



**Part A (\*\*)**

Complete **like.php** file so that the following are performed:

1. The **FORM** inside **like.php** submits back to this same file.
2. If the user clicks ‘Like’ button, **like.php** must display ‘**You like it!**’.
3. If the user clicks ‘Haha’ button, **like.php** must display ‘**You find it funny.**’
4. If the user checks ‘Bookmark’ checkbox, **like.php** must also display ‘ **Bookmarked!**’.

|  |  |
| --- | --- |
| **like.php** | **like.php**  **(after clicking a submit button)** |
| **Click on ‘Like’ submit button** |  |
| **Click on ‘Haha’ submit button** |  |
| **Click on ‘Like’ submit button** |  |
| **Click on ‘Haha’ submit button** |  |

**Question 7 - Calculator**

Go to **calculator** directory**.** Complete the following **Parts A**, **B**, and **C**.

**Part A (\*)**

**two.html** page has TWO form input fields (TEXT) and ONE (1) drop-down menu. Complete **two.php** file so that it takes TWO (2) numbers and ONE (1) operator (e.g. plus, minus, multiply, divide). The page then performs the calculation and displays the calculation result. You don’t have to perform form input validation at this time. You may assume that **two.php** is guaranteed to have all form input fields passed to it in correct format.

**Test Cases**

|  |  |
| --- | --- |
| **two.html** | **two.php** |
|  |  |
|  |  |
|  |  |

**Part B (\*\*)**

Modify **two.php** file so that it performs **Form Input Validation** as shown below.

|  |  |
| --- | --- |
| **two.html** | **two.php** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Part C (\*\*)**

**three.html** page has THREE form input fields (TEXT) and TWO (2) drop-down menus. Complete **three.php** file so that it takes all form input fields and their values. The page then performs the calculation and displays the calculation result. You don’t have to perform form input validation at this time. You may assume that **three.php** is guaranteed to have all form input fields passed to it in correct format.

**IMPORTANT**: Please make use of **calculate2()** function you completed in **Part A**.

**Test Cases**

|  |  |
| --- | --- |
| **three.html** | **three.php** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Question 8 - Temperature Conversion**

Go to **temp** directory**.** Complete the following **Part A**.

**Part A (\*\*)**

**temp.php** page allows the user to perform temperature conversion. The page has ONE form input field (TEXT), ONE (1) drop-down menu and a SUBMIT button (“equals”).

|  |
| --- |
| **temp.php** |
|  |

Complete **temp.php** page so that:

* The user can key in a numeric value in the form input field (temperature);
* The user can select the temperature scale (Fahrenheit or Celsius);
* The user can click SUBMIT, and the user is taken back to **temp.php** page. This time, **temp.php** page must perform:
  1. Temperature conversion
     + If the user’s input was “Fahrenheit”, **temp.php** must convert the temperature to Celsius.
     + If the user’s input was “Celsius”, **temp.php** must convert the temperature to Fahrenheit.
  2. Display the converted temperature value in HTML (next to “equals” SUBMIT button)
     + Converted temperature must be displayed to FIVE (5) decimal places
  3. Remember the user’s original input (temperature value **and** drop-down menu selection) and display them

**Test Cases**

|  |  |
| --- | --- |
| **temp.php (user input)** | **temp.php (after form submission)** |
|  |  |
|  |  |
| // empty input for temperature input field | // Display as above in case of empty input |

**Question 9 - Select**

Go to **select** directory**.** Complete the following **Part A**.

**Part A (\*)**

**select.php** page ONE (1) drop-down menu and a SUBMIT button (“equals”).

|  |
| --- |
| **select.php** |
|  |

The page has a pre-populated Associative Array **$schools**:

|  |
| --- |
| **select.php** |
| $schools = [  'LKCSB' => 'Business',  'SOE' => 'Economics',  'SIS' => 'Information systems',  'SOL' => 'Law',  'SOA' => 'Accountancy',  'SOSS' => 'Social Sciences' ]; |

Complete **select.php** page so that it displays a drop-down menu with the following options:

|  |  |
| --- | --- |
|  | // EXAMPLE  // <option value='LKCSB'>Business</option>  // <option value='SOE'>Economics</option>  // and so on...  // DO NOT HARDCODE  // Make use of the Associative Array **$schools** |

**Part B (\*\*)**

Modify **select.php** page. After the user selects a **school** (e.g. Law) and clicks on **SUBMIT** button, the page performs the following:

* Use the option value (e.g. SOL) as **key** and retrieve the corresponding **value** from Associative Array **$messages**
* Display this **value** (message) BELOW the drop-down menu and ABOVE the SUBMIT button.
* Remember the user’s original input (drop-down menu selection) and display it.
  + e.g. If the user selected “Law” and clicked SUBMIT, **select.php** page must pre-select Law option in the drop-down menu.
* See next page for **Test Cases**.

|  |  |
| --- | --- |
| <select name='school'>  <option value='LKCSB'>Business</option>  <option value='SIS'>Information Systems</option>  <option value='SOE' **selected**>Economics</option>  </select> | |

**Test Cases**

|  |  |
| --- | --- |
| **select.php (user input)** | **select.php (after form submission)** |
| First time visiting **select.php** | |
|  |  |
| Next, choose a different school in the drop-down menu | |
|  |  |
| Next, choose a different school in the drop-down menu | |
|  |  |

**Question 10 - Fortune Cookie**

Go to **fortune** directory**.** Complete the following **Part A**.

What makes cracking open a fortune cookie fun? You don’t know what **quote** you’ll get. In this exercise, you are to develop a webpage that will allow users to crack open a fortune cookie.

**Part A (\*\*)**

**index.php** page, when loaded in a web browser for the first time, looks like below.

|  |
| --- |
| **index.php** |
|  |

* The page displays a **closed** fortune cookie image.
* Below, the page displays a **SUBMIT** button (“Crack open your cookie!”).
* Below a Horizontal Line, there is a Hyperlink back to itself (**index.php**).

Complete **index.php** page so that it performs the following:

* When the page loads for the first time, display a **closed** fortune cookie image (as shown above).
* When the user clicks **SUBMIT** button, **index.php** must:
  + **Randomly select** a **quote** from Associative Array **$quotes** and display the **quote** in Size-1 Heading in HTML.
  + Below the **quote**, display an **open** fortune cookie image.
* At any point of time, when the user clicks on **“Home” hyperlink** at the bottom of the page, the page must treat it as if the page is loaded for the first time such that:
  + The page displays a **closed** fortune cookie image.
  + Below, the page displays a **SUBMIT** button (“Crack open your cookie!”).
  + Below a Horizontal Line, there is a Hyperlink back to itself (**index.php**).
* See next page for **Test Cases**

**Test Cases**

|  |  |
| --- | --- |
| **index.php (user input)** | **index.php (after form submission)** |
| First time visiting index.php | |
| // Click **SUBMIT** button | // The quote has been **randomly selected** |
| // Click **SUBMIT** button | // The quote has been **randomly selected** |
| // Click on **“Home” hyperlink** at the bottom |  |

**Question 11 - Validate & Forward**

Go to **validate** directory**.** Complete the following **Part A**.

**Part A (\*\*)**

**signup.php** page allows the user to specify two kinds of input.

|  |
| --- |
| **signup.php** |
|  |

The user **MUST** specify **BOTH** the 1) reason (for joining) and 2) gym type. Complete **signup.php** page so that it performs **form validation**.

**Test Cases**

|  |  |
| --- | --- |
| **signup.php (user input)** | **signup.php (after form submission)** |
| // NOTE that the **first time** this page loads, it must show the form as shown above.  // The error messages must NOT be shown. |  |
|  |  |
|  |  |
|  | Forwarded to **thankyou.php**  **(Hint: use *header("Location: thankyou.php"); More on week 12.*)** |

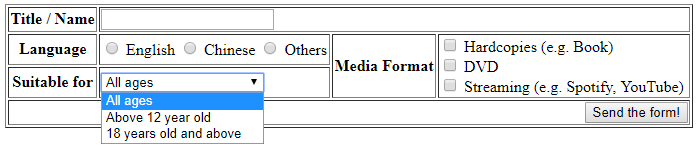
**Question 12 - Missing index.html (\*)**

*This question is to recap HTML and basic form processing concepts.*

Go to **missing\_index** directory.

The file index.html has a form that submits to receive.php.

However, the file index.html is lost. Fortunately, we know that index.html looks like this when it is loaded in a web browser.



The displayed choices for the respective fields are:

|  |  |  |
| --- | --- | --- |
| **Language**   * English * Chinese * Others | **Suitable for**   * All ages * Above 12 year old * 18 years old and above | **Media Format**   * Hardcopies (e.g. Book) * DVD * Streaming (e.g. Spotify, YouTube) |

And we also have receive.php (available in the resource folder ‘missing\_index’).

**To do:**

You are to create index.html

* Based on the above screenshot and
* Such that the given receive.php will process the submitted form values successfully.
  + You may assume receive.php is correct.
  + You must NOT edit receive.php.

**Question 13 - Text (\*)**

*This question is to recap HTML and basic form processing concepts.*

Go to **text** directory.

The file index.html has a form that submits to text.php.

|  |  |
| --- | --- |
| **User submits with the following values** | **text.php** |
| When index.html is loaded in a web browser for the first time.    User submits without entering/changing any value. | Displays error:  Message cannot be empty |
| * Message is ‘I love PHP!’ * Select both styles ‘Italic’ and ‘Bold’ * Select ‘Red’ for color and ‘Yellow’ for background | The message is italics, bold and red in color.  The background of the whole page is yellow. |
| * Message is ‘I love PHP!’ * Select ‘Blue’ for color | The message is blue in color.  The background of the whole page is white.. |

The file index.html is given in the resource folder 'text'. Do **NOT** edit this file!

**To do:**

Create text.php according to the above requirements. You may use either CSS or HTML attributes/tags for the styling.

If you intend to use CSS (out-of-scope of IS113-WAD1),

|  |
| --- |
| <html>  <body **style=’font-style: italic; font-weight: bold; color: red; background-color: yellow;’** > |
| ***The above CSS code will make***  ***The background of the web page yellow***  ***Text in italics, bold and red color*** |
| </body>  </html> |

For more information

* <https://www.w3schools.com/cssref/pr_font_font-style.asp>
* <https://www.w3schools.com/cssref/pr_font_weight.asp>
* <https://www.w3schools.com/Css/css_colors.asp>

**Question 14 - Guild**

*This question is to recap form re-population and array concepts.*

Go to **guild** directory.

You are to edit the given index.php according to the instructions below.

* Do NOT change lines 3 to 19 of index.php.
* The elements in the arrays $guilds and $heroes may change.
  + Name of guilds and its members may change.
  + Name of heros and values of their attributes may change. Each hero will always have attributes ‘Strength’ and ‘ Magic’; no additional attributes.
* You may assume there is at least one hero per guild.

**Part A (\*\*)**

Create a form has a list of checkboxes that allow user to select one or more guilds.

For each checkbox, display the guild’s name and the number of members in the guild. The guilds’ names and list of members is from the array $guilds.

* When user clicks on the displayed text, the checkbox is ticked.

E.g. guild ‘Black Star’ has 3 members, thus, the display text is ‘Black Star(3)’.

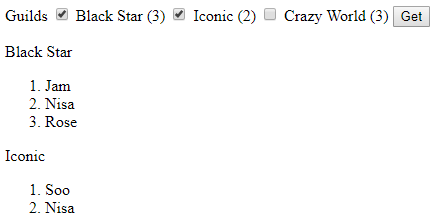
The form submits back to index.php via HTTP POST.



If user submits the form without selecting any guilds, display a message ‘No guild selected’ below the form.  

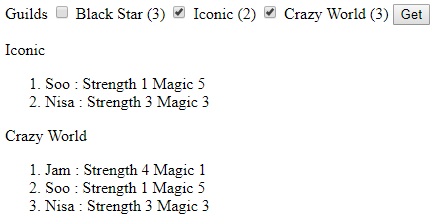

If user selects one or more guilds then submits the form,

* Re-populate the form with the user’s selection; i.e. the check boxes for the selected guilds remains ticked.
* Display the selected guilds’ names and its members’ names as an ordered list.



**Part B (\*)**

Upgrade to display each hero’s attributes which is obtainable from $heroes.

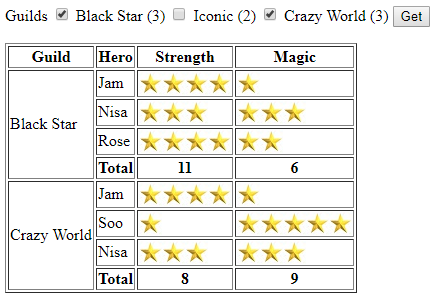


**Part C (\*\*)**

Upgrade to display the guilds’ details in a table.

For each guild, display its hero on each row.

* For each hero, display stars for strength and magic. E.g. Jam’s strength and magic are 4 and 1, thus, 4 stars and 1 star are displayed in the corresponding cells.
* Display the total value for all the guild’s heroes’ strength and magic.
* For guild ‘Black Star’, the total strength of all its heroes’ is 11 and total magic is 6.



**Question 15 - Guild-Alliance (\*\*\*)**

Go to **guild\_alliance** directory.

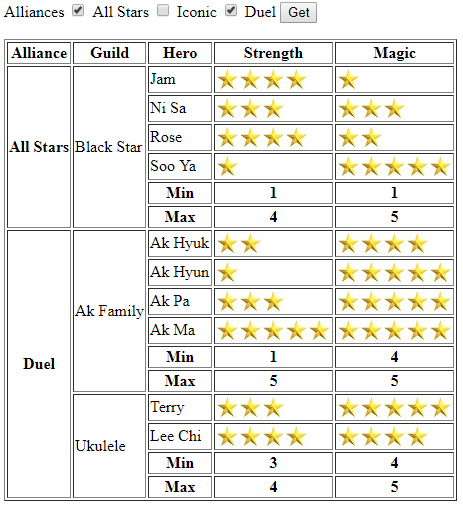
You are to edit the given index.php according to the instructions below.

* Do NOT change lines 3 to 42 of index.php.
* The elements in the arrays $alliances, $guilds and $heroes may change.
  + Name of alliances and its members may change.
  + Name of guilds and its members may change.
  + Name of heros and values of their attributes may change. Each hero will always have attributes ‘Strength’ and ‘ Magic’; no additional attributes.
* You may assume there is at least one guild per alliance and one hero per guild.

1. Create a form has a list of checkboxes that allow user to select one or more alliances.
   1. For each checkbox, display the alliance’s name
      1. The alliances’ names and list of member guilds is from the array $alliances.
      2. When user clicks on the displayed text, the checkbox is ticked.
   2. The form submits back to index.php via HTTP POST.



1. If user submits the form without selecting any alliance, display a message ‘No alliance selected’ below the form.  
     
   
2. If user selects one or more alliances then submits the form,
   1. Re-populate the form with the user’s selection; i.e. the check boxes for the selected alliances remains ticked.
   2. Display the selected alliances’ names and details of its member guilds in a table.
   3. For each guild, displays its names and its member heroes’ details.
      1. Details of the guilds’ member heroes are from the array $guilds.
      2. For each hero, display his/her name, and stars for strength and magic.
         1. Each hero’s attributes are obtainable from $heroes.
         2. E.g. Jam’s strength and magic are 4 and 1, thus, 4 stars and 1 star are displayed in the corresponding cells.
      3. Display the minimum and maximum strength and magic among the guild’s heroes’.
         1. E.g. For guild ‘Black Star’,
            1. The minimum strength is 1 (hero ‘Soo Ya’).
            2. The maximum strength is 4 (heroes ‘Jam’ and ‘Rose’).
            3. The minimum magic is 1 (hero ‘Jam’).
            4. The maximum magic is 5 (hero ‘Soo Ya’).



**Hints:**

1. **Breakdown the problem into smaller parts.**  
   Do question 14 ‘Guild’ first if you have not done so.
2. **Think through the logic.**
   1. For each user's selected alliance
      1. Calculate the number of rows that the alliance's table cell will span across.
      2. For each member guild of the alliance
         1. Calculate the number of rows that the guild's table cell will span across.
         2. For each member hero of the guild
            1. Display the hero's details
         3. Display the heroes' statistics for this guild
3. **Solve the smaller parts.**You may wish to create the following functions.
   1. **function getAllianceRowspan($guilds, $member\_guilds)**Calculates the number of rows that an alliance will span across.

Parameters

* + 1. $guilds: Dictionary of all guilds; key: name of guild. value: list of members aka heroes' name
    2. $member\_guilds: List of names of guilds that are members of the alliance.

Return the number of rows that the alliance table cell should span across.

* 1. **function genGuildStats($heroes, $guild\_heroes)**   
     Calculate the heroes' min and max attributes for a guild.

Parameters

* + 1. $heroes: Dictionary of all heroes;
       1. key: member's name.
       2. value: dictionary of member's attributes; E.g. [ 'Strength' => 3, 'Magic' => 3 ]
    2. $guild\_heroes: List of names of heroes that are members of guild.

Return a nested array representing the statistics . Structure is as follows:

[

'Min' => [

'Strength' => int,

'Magic' => int,

],

'Max' => [

'Strength' => int,

'Magic' => int

]

]

**Question 16 - Equation**

Go to **equation** directory**.**

Complete equation.html (contains a form that uses method post) and process\_equation.php (form handler) so that the following pages would be rendered:

|  |
| --- |
|  |

equation.html

|  |
| --- |
|  |

process\_equation.php

|  |
| --- |
| <!DOCTYPE html> <html>  <head>  <title>Equation</title>  </head>  <body>  This form will compute the following equation: Z = X^2 + Y<br/>  <!-- Please enter code here -->  <!-- End of code --> </body> </html> |

equation.html

|  |
| --- |
| <html>  <head>  <title>Equation</title>  </head>  <body>  <?php  // Please enter code here  // End of code  ?>  </body> </html> |

process\_equation.php

**Question 17 - Bus Fare (\*)**

Go to **bus\_fare** directory**.**

Complete bus\_fare.php below so that it can compute the bus fare of a passenger given information about distance (in km), age group (<12, 12-60, or >60), time (peak or non-peak). During non-peak hours, the fare of a bus ride is 10 cents per km. During peak-hours, the fare is 15 cents per km. Children (<12) receives a 50% discount, while senior citizens (>60) receives a 25% discount.

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Compute Bus Fare in Far Far Away Land</title>  </head>  <body>  <h2>Enter details</h2>  <form action="bus\_fare.php" method="post">  Distance (in km):   <input type="text" name="distance" size="5" required/><br/>  Age group:  <select name="age\_group">  <option value="children"><12</option>  <option value="others" selected>12-60</option>  <option value="senior">>60</option>  </select><br/>  Time:  <input type="radio" name="time" value="Peak" checked/>Peak  <input type="radio" name="time" value="Non-Peak"/>Non-Peak<br/>  <input type="submit" value="Compute Fare"/>  </form>  </body>  </html> |

bus.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Compute Bus Fare in Far Far Away Land</title>  </head>  <body>  <?php   $fare = 0.0;  // Please enter code here  // End of code  echo "Fare is $" . $fare;  ?>  </body> </html> |

bus\_fare.php

**Example:**

|  |
| --- |
|  |

ex1.html

|  |
| --- |
|  |

bus\_fare.php

**Question 18 - Prime Number (\*\*)**

Go to **prime\_number** directory**.**

Implement a PHP program that checks whether a given number is a prime number. A partially completed php page is shown below:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Check Prime</title>  </head>  <body>  <p>Please enter a whole number.</p>  <form> <!-- If action is unspecified, form submits to itself -->  <input name="number" type="text">  <input type="submit" value="Go!">   </form>  <?php  if (isset($\_GET['number'])){ // Loaded for first time?  if(isValid($\_GET['number'])){   $isPrime = true;  // Please enter code here  // End of code  if ($isPrime) {   echo "<p>".$i." is a prime number!</p>";   } else {  echo "<p>".$i." is not prime.</p>";  }     } else{  // User submitted something which is not a positive whole number   echo "<p>Please enter a positive whole number.</p>";   }  }    // check if the given number is a valid numeric value  // round() rounds a floating point value  function isValid($number) {  if(is\_numeric($number) && $number > 0  && $number == round($number, 0)) {  return true;  } else {  return false;  }  }  ?>  </body>  </html> |

prime.php

**Question 19 - Binary Search (\*\*\*)**

Go to **binary\_search** directory**.**

Implement a PHP program that performs **a binary search** (<https://en.wikipedia.org/wiki/Binary_search_algorithm>) to compute the length of a “hidden” string stored in the system. A partially completed php page is shown below:

|  |
| --- |
| <?php  if  (isset($\_GET["submit"]) && $\_GET["submit"]=="BinarySearch!") {  // predefine the search range [1, 100]  $hi = 100;  $lo = 1;  /\* add your code that performs binary search to find the length of  password string here. [Hint: You have to invoke the provided  isEqual() and/or isLess() in a loop.]  \*/    /\* end of code \*/  echo "<p> Length of the password is: " . $lo . "</p>";  }  //------ Do not modify the following two functions ---------------  function isEqual($length) {  $password = "This is a secret password!";  if(strlen($password) == $length) {  return true;  } else {  return false;  }  }  function isLess($length) {  $password = "This is a secret password!";  if($length < strlen($password)) {  return true;  } else {  return false;  }  }  ?>  <form>  <input type="submit" name="submit" value="BinarySearch!">  </form> |

binary.php

The following output will be shown if the inserted code is correct:

|  |
| --- |
|  |