## Question3.Write a PHP script to determine and display the minimum between 3 numbers $a, $b and $c. For example: if $a=5, $b=7 and $c=3 then the minimum=3.

## Question4. Given the score of a student, write a PHP script to display the grade of the student according to the following rules:

* If score >= 90 then grade=’A’
* If 80 <= score <90 then grade=’B’
* If 70 <= score < 80 then grade=’C’
* If 60 <= score < 70 then grade=’D’
* If score < 60 then grade=’F’

**Training Questions**

**Question1. Write a PHP script to calculate and display the sum of odd and even numbers between 1 and 1000 (Answer this question using while, for and do-while loops).**

**Question2. Write a PHP script to display table of 2 (until 2\*10) on a page. (Note, you may display the table simply using a loop – no need to create actual HTML Table! Simply echo the value each time multiplication is done !.**

**Part 2 – Arrays and Functions**

**Part1. Functions**

Functions are powerful because they enable you to create a section of code that can, if well-written, be copied and pasted from program to program saving you writing it each time. Here is a simple function to display the string “My first function!”:

<?php   
 function myfirstfunction() {   
 echo "My first function!";   
 }

myfirstfunction(); // call the function   
?>

**Question1.** Write a function called “minimum” that returns the minimum of two numbers.

**Question2.** Write a function called “grade” that accept a student score and display the grade according to the following rules:

* If score >= 90 then grade=’A’
* If 80 <= score <90 then grade=’B’
* If 70 <= score < 80 then grade=’C’
* If 60 <= score < 70 then grade=’D’
* If score < 60 then grade=’F’

**Question1. Write a PHP script to do the following:**

1. Create the associative array $country with the following elements.

|  |  |
| --- | --- |
| **Key** | **Value** |
| FR | France |
| OM | Oman |
| KW | Kuwait |

1. Use the function unset to remove the element FR=>France from the array $country.
2. Add the element QR=>Qatar to the array $country.
3. Sort the array $country by key in the reverse order.
4. Use the function print\_r to display the array $country.

**Question2. Write a PHP script to do the following:**

1. Create the following two dimensional array $students:

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Score** |
| 100 | Ali | 76 |
| 200 | Khaleed | 33 |
| 300 | Fatema | 89 |
| 400 | Sumaya | 45 |

1. Use foreach to display the array $students in an HTML table.
2. Display the number of pass and fail students (pass if the score is greater than 50 and fail otherwise).

## Part4. Superglobal Arrays

PHP uses associative arrays called superglobal arrays, to provide information about the PHP script’s environment. These variables are automatically available.

|  |  |
| --- | --- |
| **Superglobal array** | **Description** |
| $\_REQUEST | Parameters passed to any type of request |
| $\_GET,$\_POST | Parameters passed to GET and POST request |
| $\_SERVER,$\_ENV | Information about the web server |
| $\_SESSION,$\_COOKIE | Cookies used to identify the user |
| $\_FILES | Files uploaded |

## The following code sample displays all superglobal arrays

<html>

<head>

<title>Superglobal Arrays</title>

</head>

<body>

<h1>Superglobal Arrays</h1>

<h2>$\_COOKIE</h2>

<ol>

<?php

**foreach ($\_COOKIE as $key => $item)**

**{**

**echo "<li><b>$key:</b> $item<br/></li>";**

**}**

?>

</ol>

<hr/>

<h2>$\_ENV</h2>

<ol>

<?php

**foreach ($\_ENV as $key => $item)**

**{**

**echo "<li><b>$key:</b> $item<br/></li>";**

**}**

?>

</ol>

<hr/>

<h2>$\_FILES</h2>

<ol>

<?php

**foreach ($\_FILES as $key => $item)**

**{**

**echo "<li><b>$key:</b> $item<br/></li>";**

**}**

?>

</ol>

<hr/>

<h2>$\_GET</h2>

<ol>

<?php

**foreach ($\_GET as $key => $item)**

**{**

**echo "<li><b>$key:</b> $item<br/></li>";**

**}**

?>

</ol>

<hr/>

<h2>$\_POST</h2>

<ol>

<?php

**foreach ($\_POST as $key => $item)**

**{**

**echo "<li><b>$key:</b> $item<br/></li>";**

**}**

?>

</ol>

<hr/>

<h2>$\_REQUEST</h2>

<ol>

<?php

**foreach ($\_REQUEST as $key => $item)**

**{**

**echo "<li><b>$key:</b> $item<br/></li>";**

**}**

?>

</ol>

<hr/>

<h2>$\_SESSION</h2>

<ol>

<?php

**foreach ($\_SESSION as $key => $item)**

**{**

**echo "<li><b>$key:</b> $item<br/></li>";**

**}**

?>

</ol>

<hr/>

<h2>$\_SERVER</h2>

<ol>

<?php

**foreach ($\_SERVER as $key => $item)**

**{**

**echo "<li><b>$key:</b> $item<br/></li>";**

**}**

?>

</ol>

</body>

</html>

**Question1.** Write a PHP script to display the number of elements for each superglobal array.

|  |
| --- |
| **Part 3 – Passing Data to PHP** |

***Part1. Passing variables to a PHP script (GET method)***

A common way to pass values from the browser to the server is by **appending them to the URL** as follows:

**Syntax:**

[**http://server/path/<script\_name>?<query\_string**](http://server/path/%3cscript_name%3e?%3cquery_string)**>**

The ***<query\_string>*** part is composed of one or more name-value pairs. Each name-value pair is separated by an ampersand (&). The processing page can read these name-value pairs and use them to determine its response. Values passed in the query string are part of the ***$\_GET*** array and can be accessed using the following syntax: ***$\_GET['fieldname']***.

**Question1. Try the following example.**

|  |  |
| --- | --- |
| **Greeting.html** | **Greetingscript.php** |
| <html>  <head>  <title>Preferred Greeting</title>  </head>  <body>  <p><b>Which type of greeting you prefer? </b></p>  <ul>  <li><a href="greetingscript.php?greet=Hello">Formal</a></li>  <li><a href="greetingscript.php?greet=Hi">Informal</a></li>  <li><a href="greetingscript.php?greet=How are you">Friendly</a></li>  </ul>  </body>  </html> |  |
|  |  |

***Part2.*** ***Passing variables to a PHP script (POST method)***

Create and validate the form shown below (***helloform.html***). Then, create a PHP file that takes the text input of the form and adds it to a simple “Hello” message.



* The name attribute of the input field should have the value user
* The action of the form should be ***processhello.php***
* The method of the form should be ***POST*** so that the query string **is not encoded** on the URL.

Now, create a PHP file called ***processhello.php***. It should be mostly HTML, and contain the following:

<p> Hello

<?php

$user = htmlentities($\_POST['user']);

echo $user;

?>

**Explanation:** Your form is submitting a query string as a ***POST*** variable, the PHP program is automatically creating an array called ***$\_POST*** from it and each named field in the form can have its value accessed by acting as a key in the array.

What if someone tries to go straight to ***processhello.php*** without entering any form data?.

In order to re-direct the user to the form, try putting the following PHP code right at the top of ***processhello.php*** (even before the HTML declaration):

if (!isset($\_POST['user']))

{

header("Location: helloform.html");

exit;

}

Now try typing the URL for ***processhello.php*** directly into your browser. You should get redirected off to ***helloform.html***.

The ***isset*** function checks to see whether a variable has any value associated with it. The ! operator negates the function, so you can read the line as, “if the ***POST*** variable called ***“user”*** is not set. . . ”.

The header function modifies HTTP headers, which is why it has to go at the top of the page—it has to happen before any HTML body is sent, even whitespace.

***Part3.* *Putting form and PHP script in one PHP file***

There are several common patterns in processing forms that are worth getting to know. You have seen one already: put the form in one file, the PHP to generate the response in a second file, and specify the second file as the action of the first.

Another common pattern is for the form and response to be the same PHP file. A test is made to see if the main form variable is set; if it isn’t, the HTML form is generated as output, but if it is, the form is processed and a response is generated. The general format looks like this:

if (!isset($\_POST['some\_form\_name']))

{

// generate the form here

// The action should be the same name as THIS file

}

else

{

// do the form processing and generate the response here

}

The idea is to put as much HTML as you can outside the main block of PHP code, and generate as little as possible inside.

**Question1*.*** Create a file called ***helloform.php*** that both generates the form from the previous exercise and processes it.

**Hint:** Use **$\_SERVER['PHP\_SELF']** to get the path of the current script ***helloform.php***

***Part4. Mixed mode***

You can turn on PHP, start up an if-statement, begin the first conditional block. . . then turn PHP off and output as much HTML as you want, then turn PHP back on, finish the block, write the else branch, turn PHP back off, and so on.

**Example:**

<?php

if(!isset($\_POST['user']))

{

?>

<p>Create an HTML form here, using plain HTML</p>

<?php

}

else

{

?>

<p>Do form response here</p>

<?php

} //end of else-block

?>

<p>Put last HTML bits here

**Question1*.*** Re-write your ***helloform.php*** program as ***hellomixed.php***.

You may find it useful to be able to add smalls snippets of PHP while in pass-through mode, for instance in the action attribute of your form. It is perfectly fine to write something like:

**<form method="post" action="<?php echo "some PHP goes here"; ?>">**

This allows you to get a little piece of quick fire PHP inside a large bulk of HTML.