**PHP**

**(with Apache server)**

### 

**Stage 1**

**PHP BUILDING BLOCKS**

**Lecturer: Alexander Adu- Sarkodie**

**MSc. Telematics (IT & Telecom), MSc. Eng., Dip. Russ. Lang., Teach. Cert, AMIAEng (UK), MBCS (UK), MIfL (UK)**

**Blog: http://www.blogger.com/profile/1480049019363278**

**J. Arrays**

Arrays are **complex variables**. An array stores a **group of values** under a **single variable name**. It is useful **for storing related values**. For instance you can store information about a shirt, such as size, colour and cost in a single array named $shirt.

**Types of Array in PHP**

There are three types of array in PHP

* **Numeric** : Arrays with a ***numeric index***
* **Associative**: An array where ***each ID*** key is associated with a value. Arrays do not necessarily have to have numeric indexes for elements. The key could be arbitrarily.
* **Multi-dimensional**: An array containing ***one or more arrays.***

The simplest way of creating an array is to assign a value to a variable with square brackets [] at the end of the name. For an array named $pet with three animals, the following statement will create the array (Numeric Arrays);

<?php

$pet[1] = “dragon”;

$pet[2] = “unicorn”;

$pet[3] = “tiger”;

?>

An array can be viewed as a list of **key** **value pairs**. To get a particular value you **specify** the **key** **in the brackets**. In the preceding example the keys are 1, 2 and 3. You can also use **words** (**Associative arrays**) for keys. The following statement creates an array of country capitals;

<?php

$capitals[‘PA’] = “Paris”;

$capitals[‘LN’] = “London”;

$capitals[‘NY’] = “New York”;

?>

Another format for the above structure is:

<?php

$capitals = array(‘PA’=>’Paris’,’LN’=>’London’,’NY’=>’New York’);

?>

Note the key values are all string and quoted.

***Removing array***

To remove totally an item from the array use the **unset function** keyword in the statement

<?php

unset($pet[2]);

?>

This will remove unicorn from the array of pet.

***Sorting array***

PHP sorts arrays in a variety of ways **assigning new keys,** that are the appropriate numbers (numerical). The following statement below will sort the $pets array

<?php

sort($pets);//using the **sort()** function

?>

Consider the pet array created previously;

<?php

$pet[0] = “dragon”;

$pet[1] = “unicorn”;

$pet[2] = “tiger”;

?>

After sorting, the array becomes,

<?php

$pet[0] = “dragon”;

$pet[1] = “tiger”;

$pet[2] = “unicorn”;

?>

***Assorting arrays***

To sort arrays that use **words** for keys use the **asort()** function. This will sort by values however retaining the original key for each value

In the example below

<?php

asort($capitals);

?>

Will evaluate to;

$capitals[‘LN’] = “London”;

$capitals[‘NY’] = “New York”;

$capitals[‘PA’] = “Paris”;

Also note these:

***rsort($arrayname****)-* This will sort by value in **reverse** order and assign **new numbers** as keys

***arsort($arrayname)***– This will sort by value in reverse order and **keeps the same keys**.

***ksort($arrayname)*** – Sorts by **key**

***krsort($arrayname)*** – Sorts by **key** in **reverse** order

***usort ($arrayname, functionname)*** *­*– Sorts by **function**

***Getting values from arrays***

i) Direct Access

You can retrieve any individual value in an array by **accessing it directly** using our previous example;

<?php

$CAcapital = $capitals[‘CA’];//assign the retrieving expression to a variable

echo $CAcapital;

?>

The output from this statement is

**Sacremento**

If you include the array value in a **longer echo statement that’s enclosed by double quotes then you may have to include the array value name in curly braces like below;**

<?php

echo “The capital of England is {$capitals[‘LN’]}<br />”;

?>

ii) Using list

You can also get several values from an array using the ***list*** statement. It gets values from an array and assigns them to variables. In the example below the third value in the array is not assigned any variable and hence not copied

<?php

$shirtinfo = array (“large”,”blue”,”12.00”);

sort ($shirtinfo);

list($firstvalue.$secondvalue) = $shirtinfo;

?>

The output will be in alphabetical order

<?php

echo “$shirtinfo <br />”;

?>

and will evaluate to:

**large**

**blue**

iii) Retrieving key value pairs. For example,

In some cases you may want to retrieve keys in the array

<?php

$shirtinfo = array (“size” =>”large”, ”colour” => “blue” ,”cost” => “12.00”);

$value = $shirtinfo([‘size’])

$key = key($shirtinfo); //Note **key** is a library function

echo “$key : $value <br />”;

?>

This will evaluate to;

**size : large**

iv) Retrieving all the values of an array

You can retrieve all the values of an array with words as keys and using the **extract()** function.

<?php

extract($shirtinfo);

echo “size is $size, colour is $colour, cost is $cost”;

?>

This will evaluate to

**size is large, colour is blue, cost is 12.00**

**K. Walking through arrays using control structures**

1. **Using foreach**

Foreach walks through the array **one value at a time** and **executes** the block of statements using each value in the array. The general format is;

<?php

foreach ($arrayname as $keyname =>$valuename) {

block of statements;

}

?>

In the example that follows, the following foreach statement walks through the sample array of state capitals and echoes a list.

<?php

$capitals = array(“CA” => “Saccramento”, “TX” => “Austin”, “OR” => “Salem”);

ksort($capitals); //sort by key

// You create variables in **name value pair relationship to store each array value**. Note the //**as**  keyword

// The <br /> tag will put each record on one line.

foreach($capitals as $state => $city) {

echo “$city , $state <br />”;

}

?>

The output will be:

**Sacramento CA,**

**Salem, OR**

**Austin, TX**

The previous example can be written to output just the values,

<?php

foreach ($arrayname as $valuename) {

block of statements;

}

?>

<?php

foreach($capitals as $city) {

echo “$city <br />”;

}

?>

The webpage output will simplify to:

**Sacramento**

**Salem**

**Austin,**

Inthe next example, suppose you want to output an array of links, using the *foreach* loop statement,

<?php

$links = array(“The apache web server” => “www.apache.org”, “Apres” => “www.apres.com”,”The PHP scripting Language” => “www.php.net”);

echo “<strong>Online resources<strong>:<br />”;

foreach($links as $title => $link) {

echo “<a href=\”http://$link\”>$title</a><br />”;

}

?>

**Note**: Escape inner double quotes (**Escape sequence**).

This will result in the following:

**Online resources:<br />**

**<a href=”http:// www.apache.org”> The apache web server </a><br />**

**<a href=”** **www.apres.com”> Apres</a><br />**

**<a href=”** **www.php.net”> The PHP scripting Language </a><br />**

1. **Multidimensional arrays**

Supposing we have the following data to process:

Shirt 20.00

Pants 22.50

Blanket 25.00

Bedspread 50.00

Lamp 44.00

Rug 75.00

We can model the above data by **aggregating** and **classifying** them into **groups**, say **clothing**, **linen** and **furniture**. This will allow **easy search** and data access than itemizing each individually as an array item. You can classify the products by putting the cost in a multi-dimensional array as follow. Since they are related by, a **common thing**. They are all **priced**.

<?php

$productPrices[‘clothing’][‘shirt’] = 20.00;

$productPrices[‘clothing’][‘pants’] = 22.50;

$productPrices[‘linens’][‘blanket’] = 25.00;

$productPrices[‘linens’][‘bedspread’] = 50.00;

$productPrices[‘furniture’][‘lamp’] = 44.00;

$productPrices[‘furniture’][‘rug’] = 75.00;

?>

The $productPrices has **three / key value pairs**. The keys are **clothing**, **linen** and **furniture**. The **value** for each **key** is an array with **two/key value pairs**. For instance, the value to the key clothing is an array with the two/key values ***shirt / 20.00*** and ***pants / 22.50***

This is a two dimensional area. The structure of this data set is illustrated below

**Product prices key value**

**Key value**

Clothing shirt 20.00

Pants 22.50

Linen blanket 30.00

Bedspread 50.00

Furniture lamp 44.00

Rug 75.00

**Extracting values from multi-dimensional arrays**

Values from multidimensional arrays can be extracted similarly to the procedure for a single dimensional arrays. Both syntaxes below can be used.

<?php

$shirtprice = $productPrices[‘clothing’][‘shirt’];

echo $shirtprice;

?>

<?php

echo $productPrices[‘clothing’][‘shirt’];

?>

However if you **combine the value within double quotes** then you will have to have **curly braces just before the dollar sign with NO SPACE!**

<?php

print “The price of a shirt is £ {$productPrices[‘clothing’][‘shirt’]}”;

?>

The output is: **The price of a shirt is £20**

**Traversing a multi-dimensional arrays**

You can walk through a multi-dimensional array using foreach statement. **One *foreach* statement for each dimensional array**. So for a two dimensional array like $productPrices you

will require **two** foreach statements. One nested in the other.

The following statements get the values from the multidimensional array and output them in an html table.

i)

<?php

$productPrices[‘clothing’][‘shirt’] = 20.00;

$productPrices[‘clothing’][‘pants’] = 22.50;

$productPrices[‘linens’][‘blanket’] = 25.00;

$productPrices[‘linens’][‘bedspread’] = 50.00;

$productPrices[‘furniture’][‘lamp’] = 44.00;

$productPrices[‘furniture’][‘rug’] = 75.00;

print “<table border=1>”;

foreach($productPrices as $productPrice) {

foreach ($productPrice as $product => $price){

$f\_price = sprintf(“%01.2f”,$price);//formats price to 2dp for 100 units

echo “<tr><td>$product : </td> <td>\$ $f\_price </td>”;”;

}

}

echo “</table>”;

?>

ii Using **count** to determine size of an array

<?php

$family = array(“Tom”,”Ian”,”Sophie”,”Alex”);

$family = array(“Pat”,”Frances”,”Athur”,”John”);

$rows = count($family);

$cols = count($family[0]);//**MUST** set first item in column field

//Column count

for($i<0;$i<$rows;$i++) {

for($j<0;$j<$cols;$j++) {

echo $family[$i][$j] . ‘ ‘;

}

echo “<br />”

}

?>

ii Using **rugged**  arrays

This is when the inner arrays are of different sizes. If this is the case you just need to check the number of columns, in each iteration.

<?php

$family = array(“Tom”,”Ian”,”Sophie”);

$family = array(“Pat”,”Frances”,”Athur”,”John”);

$rows = count($family);

for($i<0;$i<$rows;$i++) {

**$cols = count($family[i]);//**check for position for columns before looping

for($j<0;$j<$cols;$j++) {

echo $family[$i][$j] . ‘ ‘;

}

echo “<br />”;

}

?>

## PHP Form Handling

The most important thing to notice when dealing with **HTML forms** and PHP is that, any form element in an HTML page will **automatically** be available to your PHP scripts. So long as the form elements have **names**.

Good because you don’t have to use extra identifiers like IDs. You can use the **names** as **keys** in the **$\_POST** or **$\_GET** arrays**.** Bad in the sense that it can invoke or trigger **Cross Site Scripting** (XSS). **Malicious code can easily be injected to any of the form fields by users and submitted**. Hence the need for sanitization and validation required.

The PHP **$\_GET** and **$\_POST** variables are used to retrieve information from forms, like user input.

### **Example**

The example below contains an HTML form with two input fields and a submit button:

<html>  
<body>  
<form action="**welcome.php**" method="post">  
 Name: <input type="text" name="**fname**" />  
 Age: <input type="text" name="**age**" />  
 <input type="submit" />  
</form>  
</body>  
</html>

When a user fills out the form above and clicks on the submit button, the form data is sent to a PHP file, called "welcome.php":

"welcome.php" looks like this:

<html>  
<body>  
Welcome <?php echo $\_POST["**fname**"]; ?>!<br />  
You are <?php echo $\_POST["**age**"]; ?> years old.  
</body>  
</html>

Output could be something like this:

Welcome John!  
You are 28 years old.

## Form Validation

User input should be **validated** on the browser whenever possible (by **client scripts**). **Browser validation is faster and reduces the server load.**

You should consider server validation if the user input will be **inserted** into a **database**. A good way to **validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form**. This makes it **easier** to **discover the error**.

# PHP $\_GET Function

The built-in **$\_GET** function is used to collect values from a form sent with **method="get".**

Information sent from a form with the **GET** method is visible to everyone (it will be displayed in the **browser's address bar**) and has limits on the amount of information to send (**max. 100 characters**).

### **Example**

<form action="welcome.php" **method="get"**>  
 Name: <input type="text" name="**fname**" />  
 Age: <input type="text" name="**age**" />  
 <input type="submit" />  
</form>

When the user clicks the "Submit" button, the URL sent to the server could look something like this:

[http://www.w3schools.com/welcome.php**?fname=Peter&age=37**](http://www.w3schools.com/welcome.php?fname=Peter&age=37)

The "welcome.php" file can now use the $\_GET function to collect form data (**the names of the form fields will automatically be the keys in the $\_GET array**):

Welcome <?php echo $\_GET["fname"]; ?>.<br /> //**fname** and **age** are **keys** in the $\_GET array   
You are <?php echo $\_GET["age"]; ?> years old!

## When to use method="get"?

When using method="get" in HTML forms, all variable names and values are displayed in the URL.

**Note:** This method **should not be used when sending passwords or other sensitive information**!

However, because the variables are displayed in the URL, it is possible to **bookmark** the page. This can be useful in some cases. For search engine optimisation (SEO).

**Note:** The get method is not suitable for large variable values; the **value cannot exceed 100 characters in most browsers.**

# PHP $\_POST Function

## The $\_POST Function

The built-in **$\_POST** function is used to collect values from a form sent with **method="post".**

Information sent from a form with the POST method is **invisible** to others and has **no limits** on the amount of information to send.

**Note:** However, there is an **8 Mb max size** for the POST method, by **default** (can be changed by setting the **post\_max\_size in the php.ini file**).

### **Example**

<form action="welcome.php" **method="post">**  
 Name: <input type="text" name="**fname**" />  
 Age: <input type="text" name="**age**" />  
 <input type="submit" />  
</form>

When the user clicks the "Submit" button, the URL will look like this: <http://www.w3schools.com/welcome.php>

The "welcome.php" file can now use the $\_POST function to collect form data (the **names** of the form fields will automatically be the **keys** in the **$\_POST array**):

Welcome <?php echo $\_POST["**fname**"]; ?>!<br />  
You are <?php echo $\_POST["**age**"]; ?> years old.

## When to use method="post"?

Information sent from a form with the POST method is **invisible** to others and has no **limits** on the amount of information to send (Default value is 8MB, but this can be changed in the .ini file).

However, because the variables are not displayed in the URL, it is not **possible to bookmark** the page. For say SEO purposes.

## The PHP $\_REQUEST Function

The PHP built-in **$\_REQUEST** function contains the contents of both **$\_GET**, **$\_POST**, and **$\_COOKIE**.

The $\_REQUEST function can be used to collect form data sent with both the GET and POST methods.

### **Example**

Welcome <?php echo **$\_REQUEST**["fname"]; ?>!<br />  
You are <?php echo **$\_REQUEST**["age"]; ?> years old.

# PHP Sending E-mails

PHP allows you to send e-mails directly from a script.

## The PHP mail() Function

The PHP mail() function is used to send emails from inside a script.

**Syntax**

**mail(to,subject,message,headers,parameters)**

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| To | Required. Specifies the receiver / receivers of the email |
| subject | Required. Specifies the subject of the email. **Note:** This parameter cannot contain **any newline characters** |
| message | Required. Defines the message to be sent. Each line should be separated with a LF **(\n).** Lines should not exceed **70** characters |
| headers | Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n) |
| parameters | Optional. Specifies an additional parameter to the sendmail program |

## PHP Simple E-Mail

The simplest way to send an email with PHP is to send a text email.

In the example below we first declare the variables ($to, $subject, $message, $from, $headers), then we use the variables in the **mail()** function to send an e-mail:

<?php  
 $to = "someone@example.com";  
 $subject = "Test mail";  
 $message = "Hello! This is a simple email message.";  
 $from = "someonelse@example.com";  
 $headers = "From: $from";  
 mail($to,$subject,$message,$headers);  
 echo "Mail Sent.";  
?>

## PHP Mail Form

With PHP, you can create a feedback-form on your website. The example below sends a text message to a specified e-mail address:

<html>  
<body>  
  
<?php  
if (isset($\_REQUEST['email'])) {   
   //if "email" is filled out, send email  
   //send email  
   $email = $\_REQUEST['email'] ;  
   $subject = $\_REQUEST['subject'] ;  
   $message = $\_REQUEST['message'] ;  
   mail( "someone@example.com", "Subject: $subject",  
   $message, "From: $email" );  
   echo "Thank you for using our mail form";  
  }  
else   {  
 //if "email" is not filled out, display the form  
   echo "<form method='post' action='mailform.php'>  
   Email: <input name='email' type='text' /><br />  
   Subject: <input name='subject' type='text' /><br />  
   Message:<br />  
   <textarea name='message' rows='15' cols='40'></textarea><br />  
   <input type='submit' />  
   </form>";  
  }  
?>  
</body>  
</html>

This is how the example above works:

* First, check if the email input field is filled out
* If it is not set (like when the page is first visited); output the HTML form
* If it is set (after the form is filled out); send the email from the form
* **When submit is pressed after the form is filled out, the page reloads, sees that the email input is set, and sends the email.**

The problem with the code above is that unauthorized users can insert data into the **mail headers** via the input form and send or redirect the mail.

## PHP Stopping E-mail Injections

The best way to stop **e-mail injections** is to **validate** the **input**.

The code below is the same as in the previous chapter, but now we have added an input validator that checks the email field in the form:

In the code above we use **PHP filters** to validate input:

* The **FILTER\_SANITIZE\_EMAIL** filter removes all **illegal e-mail characters** from a string
* The **FILTER\_VALIDATE\_EMAIL** filter validates value as an e-mail address

<html>

<body>

<?php

function spamcheck($field){

//filter\_var() sanitizes the e-mail

//address using FILTER\_SANITIZE\_EMAIL

$field=filter\_var($field, FILTER\_SANITIZE\_EMAIL);

//filter\_var() validates the e-mail

//address using FILTER\_VALIDATE\_EMAIL

if(filter\_var($field, FILTER\_VALIDATE\_EMAIL)){

return TRUE;

}

else {

return FALSE;

}

}

if (isset($\_REQUEST['email'])){

//if "email" is filled out, proceed

//check if the email address is invalid

$mailcheck = spamcheck($\_REQUEST['email']);

if ($mailcheck==FALSE) {

echo "Invalid input";

}

else {

//send email

$email = $\_REQUEST['email'] ;

$subject = $\_REQUEST['subject'] ;

$message = $\_REQUEST['message'] ;

mail("someone@example.com", "Subject: $subject",

$message, "From: $email" );

echo "Thank you for using our mail form";

}

}

else{

//if "email" is not filled out, display the form

echo "<form method='post' action='mailform.php'>

Email: <input name='email' type='text' /><br />

Subject: <input name='subject' type='text' /><br />

Message:<br />

<textarea name='message' rows='15' cols='40'></textarea><br />

<input type='submit' />

</form>";

}

?>

</body>

</html>