PHP

PHP Functions

The real power of PHP comes from its functions; it has more than 1000 built-in functions.

a)PHP User Defined Functions

In PHP functions, we can create our own functions.

A function is a block of statements that can be used repeatedly in a program.

A function will not execute immediately when a page loads.

A function will be executed by a call to the function.

- **Syntax**
- function *functionName*() {
  code to be executed;
  }
- E.g.

```
<?php
  function sum($x,$y) {
    $z=$x+$y;
    return $z;
  }
  echo "5 + 10 = " . sum(5,10) . "<br>";
  echo "2 + 4 = " . sum(2,4);
  ?>
```

- Output:- 5 + 10 = 15
  2 + 4 = 6

- **PHP 5 Global Variables - Superglobals**

a)Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

b)The PHP superglobal variables are:

- $GLOBALS, $_SERVER , $_REQUEST , $_POST , $_GET
- $_FILES, $_ENV, $_COOKIE, $_SESSION

- **PHP $GLOBALS**
- $GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).
- PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable.
- <?php
  $x = 75; $y = 25;
  function addition() {
    $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
  }
  addition();
  echo $z;
  ?> Output:- 100

- **ii) PHP $_SERVER**
- $_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.
- <?php
  echo $_SERVER['PHP_SELF'];
- echo "
";
  echo $_SERVER['SERVER_NAME'];
- echo "
";
  echo $_SERVER['HTTP_HOST'];
- echo "
";
  echo $_SERVER['HTTP_REFERER'];
- echo "
";
  echo $_SERVER['SCRIPT_NAME'];
  ?>

Element/Code	Description
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

- **iii) PHP $_REQUEST**
- PHP $_REQUEST is used to collect data after submitting an HTML form.
- <html><body>
  <form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit"> </form>
  <?php
  $name = $_REQUEST['fname'];
  echo $name;
  ?></body></html>

- **iv) PHP $_POST**
- PHP $_POST is widely used to collect form data after submitting an HTML form with method="post".
- <html><body>
  <form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit"> </form>
  <?php
  $name = $_POST['fname'];
  echo $name;
  ?></body> </html>

- **PHP $_GET**
- PHP $_GET can also be used to collect form data after submitting an HTML form with method="get".
- $_GET can also collect data sent in the URL.
- Assume we have an HTML page that contains a hyperlink with parameters:
- <html><body>
  Test $GET
  </body></html>
- When a user clicks on the link "Test $GET", the parameters "subject" and "web" is sent to "test_get.php", and you can then acces their values in "test_get.php" with $_GET.
- The example below shows the code in "test_get.php":
- **Example**
- <html><body>
  <?php
  echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
  ?>
  </body></html>

- **PHP Date and Time**
- The PHP date() function is used to format a date and/or a time.
- **a)The PHP Date() Function**
- The PHP date() function formats a timestamp to a more readable date and time.
- **Syntax**
- date(*format*,*timestamp*)
- <?php
  echo "Today is " . date("Y/m/d") . "
";
  echo "Today is " . date("Y.m.d") . "
";
  echo "Today is " . date("l");
  ?>
- Output:- Today is 2014/08/05
  Today is 2014.08.05
  Today is Tuesday

- **Create a Date With PHP mktime()**
- The optional *timestamp* parameter in the date() function specifies a timestamp. If you do not specify a timestamp, the current date and time will be used (as shown in the examples above).
- <?php
  $d=mktime(11, 14, 54, 8, 12, 2014);
  echo "Created date is " . date("Y-m-d h:i:sa", $d);
  ?>
- Output:- Created date is 2014-08-12 11:14:54am.

PHP Include Files

- **Syntax:-**
- include '*filename*'; or
  require '*filename*';
- It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement.

- **PHP include Examples**
- Assume we have a standard footer file called "footer.php", that looks like this:
- <?php
  echo " Copyright © 1999-" . date("Y") . "
  Example.com ";
  ?>
- To include the footer file in a page, use the include statement:
- **Example**
- <html><body>
  <h1>Welcome to my home page!</h1>
  Some text.
  Some more text.
  <?php include 'footer.php';?>
  </body></html>
- **Output: Welcome to my home page!**
- Some text.
- Some more text.
- Copyright © 1999-2014 W3Schools.com

PHP File Handling

- The readfile() function reads a file and writes it to the output buffer.
- Assume we have a text file called "webdictionary.txt", stored on the server,
- that looks like this:
- AJAX = Asynchronous JavaScript and XML
  CSS = Cascading Style Sheets
  HTML = Hyper Text Markup Language
  PHP = PHP Hypertext Preprocessor
  XML = EXtensible Markup Language

- **Example**
- <?php
  echo readfile("webdictionary.txt");
  ?>
- The readfile() function is useful if all you want to do is open up a file and read its contents.
- The next chapters will teach you more about file handling.

- ```
  <?php
  $myfile = fopen("webdictionary.txt", "r") or
  die("Unable to open file!");
  echo
  fread($myfile,filesize("webdictionary.txt"));
  fclose($myfile);
  ?>
  ```
  ```

Modes	Description
r	**Open a file for read only**. File pointer starts at the beginning of the file
w	**Open a file for write only**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	**Open a file for write only**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	**Creates a new file for write only**. Returns FALSE and an error if file already exists
r+	**Open a file for read/write**. File pointer starts at the beginning of the file
w+	**Open a file for read/write**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	**Open a file for read/write**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	**Creates a new file for read/write**. Returns FALSE and an error if file already exists

- **PHP Read File - fread()**

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

```
fread($myfile,filesize("webdictionary.txt"));
```

- **PHP Close File - fclose()**

The fclose() function is used to close an open file.

It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

```
<?php
    $myfile = fopen("webdictionary.txt", "r");
    // some code to be executed....
    fclose($myfile);
    ?>
```

- **PHP Read Single Line - fgets()**

The fgets() function is used to read a single line from a file.

The example below outputs the first line of the "webdictionary.txt" file:

Example

```
<?php
    $myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
    echo fgets($myfile);
    fclose($myfile);
    ?>
```

Note: After a call to the fgets() function, the file pointer has moved to the next line.

- **PHP Check End-Of-File - feof()**

The feof() function checks if the "end-of-file" (EOF) has been reached.

The feof() function is useful for looping through data of unknown length.

The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

Example

```
<?php
    $myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
    // Output one line until end-of-file
    while(!feof($myfile)) {
      echo fgets($myfile) . "<br>";
    }
    fclose($myfile);
    ?>
```

- **PHP Read Single Character - fgetc()**

The fgetc() function is used to read a single character from a file.

The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

Example

```
<?php
    $myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
    // Output one character until end-of-file
    while(!feof($myfile)) {
      echo fgetc($myfile);
    }
    fclose($myfile);
    ?>
```

Note: After a call to the fgetc() function, the file pointer moves to the next character.

- **PHP Write to File - fwrite()**

i)The fwrite() function is used to write to a file.

ii)The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.

iii)ex.

```
<?php
    $myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
    $txt = "Jspm's abacus";
    fwrite($myfile, $txt);
    fclose($myfile);
    ?>
```

PHP Error Handling:-

- When creating scripts and web applications, error handling is an important part. If your code lacks error checking code, your program may look very unprofessional and you may be open to security risks.
- We will show different error handling methods:
- i)Simple "die()" statements.
- ii)Custom errors and error triggers
- iii)Error reporting

- **I)Basic Error Handling: Using the die() function**
- The first example shows a simple script that opens a text file:
- <?php
  $file=fopen("welcome.txt","r");
  ?>
- If the file does not exist you might get an error like this:
- **Warning**: fopen(welcome.txt) [function.fopen]: failed to open stream:
  No such file or directory in **C:\webfolder\test.php** on line **2**
- To prevent the user from getting an error message like the one above, we test whether the file exist before we try to access it and if not present give you your error message .
- <?php
  if(!file_exists("welcome.txt")) {
  die("File not found");
  } else {
  $file=fopen("welcome.txt","r");
  }
  ?>

ii) Creating a Custom Error Handler

Creating a custom error handler is quite simple. We simply create a special function that can be called when an error occurs in PHP.

This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context):

Syntax

error_function(error_level,error_message, error_file,error_line,error_context)

Parameter

Description

error_level

Required. Specifies the error report level for the user-defined error. Must be a value number.

error_message

Required. Specifies the error message for the user-defined error

error_file

Optional. Specifies the filename in which the error occurred

error_line

Optional. Specifies the line number in which the error occurred

error_context

Optional. Specifies an array containing every variable, and their values, in use when the error occurred

Error Report levels

These error report levels are the different types of error the user-defined error handler can be used for:

Value	Constant	Description
2	E_WARNING	Non-fatal run-time errors. Execution of the script is not halted
8	E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script normally
256	E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()
512	E_USER_WARNING	Non-fatal user-generated warning. This is like an E_WARNING set by the programmer using the PHP function trigger_error()
1024	E_USER_NOTICE	User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()
4096	E_RECOVERABLE_ERROR	Catchable fatal error. This is like an E_ERROR but can be caught by a user defined handle (see also set_error_handler())
8191	E_ALL	All errors and warnings (E_STRICT became a part of E_ALL in PHP 5.4)

aa) Set Error Handler

i)The default error handler for PHP is the built in error handler. We are going to make the function above the default error handler for the duration of the script.

ii)It is possible to change the error handler to apply for only some errors, that way the script can handle different errors in different ways. However, in this example we are going to use our custom error handler for all errors:

```
set_error_handler("customError");
```

iii)Since we want our custom function to handle all errors, the set_error_handler() only needed one parameter, a second parameter could be added to specify an error level.

e.g Testing the error handler by trying to output variable that does not exist:

```
<?php
    //error handler function
    function customError($errno, $errstr) {
      echo "<b>Error:</b> [$errno] $errstr";
    }
    set_error_handler("customError");
            echo($test);     ?>
```

The output of the code above should be something like this: **Error:** [8] Undefined variable: test

iii)Trigger an Error

In a script where users can input data it is useful to trigger errors when an illegal input occurs. In PHP, this is done by the trigger_error() function.

Example

In this example an error occurs if the "test" variable is bigger than "1":

```
<?php
$test=2;
if ($test>1) {
  trigger_error("Value must be 1 or below");
}
?>
```

Example

In this example an E_USER_WARNING occurs if the "test" variable is bigger than "1". If an E_USER_WARNING occurs we will use our custom error handler and end the script:

```
<?php
    //error handler function
    function customError($errno, $errstr) {
      echo "<b>Error:</b> [$errno] $errstr<br>";
      echo "Ending Script";
      die();
    }
    //set error handler
    set_error_handler("customError",E_USER_WARNING);
    //trigger error
    $test=2;
    if ($test>1) {
      trigger_error("Value must be 1 or below",E_USER_WARNING);
    }
    ?>
```

The output of the code above should be something like this:

Error: [512] Value must be 1 or below
Ending Script

c)Try, throw and catch

To avoid the error from the example above, we need to create the proper code to handle an exception. Proper exception code should include:

Try - A function using an exception should be in a "try" block. If the exception does not trigger, the code will continue as normal. However if the exception triggers, an exception is "thrown"

Throw - This is how you trigger an exception. Each "throw" must have at least one "catch"

Catch - A "catch" block retrieves an exception and creates an object containing the exception information

Lets try to trigger an exception with valid code:

```
<?php
    function checkNum($number) {
      if($number>1) {
        throw new Exception("Value must be 1 or below");
      }
      return true;
    }
    //trigger exception in a "try" block
    try {
      checkNum(2);
      //If the exception is thrown, this text will not be shown
      echo 'If you see this, the number is 1 or below';
    }
    catch(Exception $e) {
      echo 'Message: ' .$e->getMessage();  }
    ?>
```

PHP Cookies:-

A cookie is often used to identify a user.

The setcookie() function is used to set a cookie.

Note: The setcookie() function must appear BEFORE the <html> tag.

Syntax

setcookie(name, value, expire, path, domain);

Here is the detail of all the arguments:

Name - This sets the name of the cookie and is stored in an environment variable called HTTP_COOKIE_VARS. This variable is used while accessing cookies.

Value -This sets the value of the named variable and is the content that you actually want to store.

Expiry - This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.

- **Path -**This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.
- **Domain -** This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.
- **Security -** This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Example

You can also set the expiration time of the cookie in another way. It may be easier than using seconds. cookie will expire after 30 days. 60*60*24=86400 = 1 day

```
<?php
	$expire=time()+60*60*24*30;
	setcookie("user", "Alex ", $expire);
	?>
```

In the example above the expiration time is set to a month (*60 sec * 60 min * 24 hours * 30 days*).

c)How to Retrieve a Cookie Value?

The PHP $_COOKIE variable is used to retrieve a cookie value.

In the example below, we retrieve the value of the cookie named "user" and display it on a page:

```
<?php
	// Print a cookie
	echo $_COOKIE["user"];
	// A way to view all cookies
	print_r($_COOKIE);
	?>
```

In the following example we use the isset() function to find out if a cookie has been set:

```
<html><body>
	<?php
	if (isset($_COOKIE["user"]))
	  echo "Welcome " . $_COOKIE["user"] . "!<br>";
	else
	  echo "Welcome guest!<br>";
	?>
	</body></html>
```

d)How to Delete a Cookie?

When deleting a cookie you should assure that the expiration date is in the past.

Delete example:

```
<?php
	// set the expiration date to one hour ago
	setcookie("user", "", time()-3600);
	?>
```

PHP Sessions

- **Starting a PHP Session**

Before you can store user information in your PHP session, you must first start up the session.

Note: The session_start() function must appear BEFORE the <html> tag:

<?php session_start(); ?>

The code above will register the user's session with the server, allow you to start saving user information, and assign a UID for that user's session.

- **Storing a Session Variable:**

Session variables are stored in associative array called $_SESSION[]. These variables can be accessed during lifetime of a session.

```
<?php
session_start();    // Start the session
?>
<!DOCTYPE html>
<html><body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?></body></html>
```

- **Get Session Variable Values**

```
<?php
    session_start();
    ?>
    <!DOCTYPE html>
    <html><body>
    <?php
    // Echo session variables that were set on previous example
    echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
    echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
    ?>
    </body></html>
```

Another way to show all the session variable values for a user session is to run the following code:

```
<?php
    session_start();
    ?>
    <!DOCTYPE html>
    <html><body>
    <?php
    print_r($_SESSION);
    ?>
    </body></html>
();
    ?>
```

- **Modify a PHP Session Variable**

To change a session variable, just overwrite it:

```
<?php
    session_start();
    ?>
    <!DOCTYPE html>
    <html><body>
    <?php
    // to change a session variable, just overwrite it
    $_SESSION["favcolor"] = "yellow";
    print_r($_SESSION);
    ?>
    </body></html>
```

- The correct way to store and retrieve session variables is to use the PHP $_SESSION variable:

```
<?php
    session_start();
    if(isset($_SESSION["count"])) {
        $accesses = $_SESSION["count"] + 1;
    } else {
        $accesses = 1;
    }
     $_SESSION["count"] = $accesses;
 ?>
 <html>  <head>  <title>Access counter</title>  </head>
  <body>
<h1>Access counter</h1>
 <p>You have accessed this page <?php echo $accesses; ?> times today.</p>
 </body>  </html>
```

- **Destroying a Session**

If you wish to delete some session data, you can use the unset() or the session_destroy() function.

The unset() function is used to free the specified session variable:

```
<?php
    session_start();
    if(isset($_SESSION['views']))
      unset($_SESSION['views']);
    ?>
```

You can also completely destroy the session by calling the session_destroy() function:

```
<?php
    session_destroy
```

Sending email:-

- PHP makes use of **mail()** function to send an email. This function requires three mandatory arguments that specify the recipient's email address, the subject of the the message and the actual message additionally there are other two optional parameters.
- 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. 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

```
<html><head><title>Sending email using PHP</title></head><body>
<?php
   $to = "xyz@somedomain.com";
   $subject = "This is subject";
   $message = "This is simple text message.";
   $header = "From:abc@somedomain.com \r\n";
   $retval = mail ($to,$subject,$message,$header);
   if( $retval == true )
   {     echo "Message sent successfully...";
   }
   else
   {     echo "Message could not be sent...";
   }
?>
</body></html>
```

Object Oriented Concepts in PHP:-

a)Defining PHP Classes:

The general form for defining a new class in PHP is as follows:

```
<?php
class  Books{    /* Member variables */
   var $price;    var $title;
   /* Member functions */
   function setPrice($par){
     $this->price = $par;
   }
   function getPrice(){
     echo $this->price ."<br/>";
   }
   }
?>
```

Creating Objects in PHP

```
$physics = new Books;
  $maths = new Books;
```

Calling Member Functions

```
 $physics->setPrice( 10 );
  $maths->setPrice( 7 );
```

Now you call another member functions to get the values set by in above example:

```
  $physics->getPrice();
  $maths->getPrice();
```

This will produce follwoing result: 10, 7

Constructor Functions:

Constructor Functions are special type of functions which are called automatically whenever an object is created. So we take full advantage of this behaviour, by initializing many things through constructor functions.

```
function __construct( $par1, $par2 ){
  $this->price = $par1;
  $this->title = $par2;
}
```

Now we don't need to call set function separately to set price and title. We can initialize these two member variables at the time of object creation only. Check following example below:

```
  $physics = new Books( 10 );
  $maths = new Books (15 );
  /* Get those set values */

  $physics->getPrice();
    $maths->getPrice();
```

Destructor:

Example using constructor and desructor function

```
<?php
class  Books{
   /* Member variables */
   var $price;
   var $title;
   /* Member functions */
function __construct( $par1, $par2 ){
  $this->title = $par1;
  $this->price = $par2;
}
       function getPrice(){
      echo $this->price ."<br/>";
   }
      function getTitle(){
      echo $this->title ." <br/>";
   }
function     destruct()
```

Inheritance:

The effect of inheritance is that the child class (or subclass or derived class) has the following characteristics:

Automatically has all the member variable declarations of the parent class.

Automatically has all the same member functions as the parent, which (by default) will work the same way as those functions do in the parent.

```
class Novel extends Books{
  var publisher;
  function setPublisher($par){
    $this->publisher = $par;
  }
  function getPublisher(){
    echo $this->publisher. "<br />";
  }
}
```

Function Overriding:	**Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.**	**function getPrice(){
		echo $this->price . "
		";
		return $this->price;
		}
		function getTitle(){
		echo $this->title . "
		";
		return $this->title;
		}**

Public Members:	If you wish to limit the accessibility of the members of a class then you define class members as **private** or **protected**.	**Private members:** **Protected members:**
Interfaces:	Interfaces are defined to provide a common function names to the implementors.	interface Mail { public function sendMail(); }

Constants:	**A constant is somewhat like a variable, in that it holds a value, but is really more like a function because a constant is immutable**	**class MyClass {
		  const requiredMargin = 1.7;
		  function __construct($incomingValue)
		{**

Abstract Classes:	An abstract class is one that cannot be instantiated, only inherited. You declare an abstract class with the keyword **abstract**	abstract class MyAbstractClass
		{
		  abstract function myAbstractFunction() {
		  }
		}

Static Keyword:	**Declaring class members or methods as static makes them accessible without needing an instantiation of the class**	**<?php
		class Foo
		{ public static $my_static = 'foo';
		public function staticValue() {
		return self::$my_static;
		}
		}
		print Foo::$my_static . "\n";
		$foo = new Foo();
		print $foo->staticValue() . "\n";**

Final Keyword	PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with final	BaseClass::moreTesting()