UNIT 4

PHP FUNCTIONS

PHP has hundreds of base functions and thousands more via extensions. Besides built-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.

User defined function inPHP

- A user defined function declaration starts with the word ¯function.
- A function name can start with a letter or underscore (not a number).
- Function names are case-insensitive.

Syntax

```
Function functionName()
{
code to be executed;
}
```

Example

```
<?php
functionmsg( )
{
   echo "hello  world";
}

msg();
?>
```

OUTPUT

hello world

PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable. Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with two arguments ($fname and $year):

```
<?php
Function personal($fname,$dob)
{
   Echo "$fname.born in .$dob.<br>";
}
Personal("rahul",1990);
Personal("sam",1995);
?>
```

OUTOUT

Rahul born in 1990
Sam born in 1995

PHP Default Argument Value

When designing your functions, it is often helpful to be able to assign default values for parameters that aren't passed - PHP does this for most of its functions, and it saves you having to pass in parameters most of the time if they are usually the same.

To define your own default parameters for a function, simply add the constant value you would like them set to after the variables, like this:

Example

```
<?php
functionfun_default($Name = "Smith") {
return "Hai $Name!\n";
   }
fun_default();
fun_default (null);
fun_default("John");
?>
```

output
HaiPaul!
Hail
Hai John!

PHP Functions - Returning Values

PHP return statement immediately terminates the execution of a function when it is called from within that function. To let a function return a value, use the return statement.

```
<?php
function square($num)
{
  return $num * $num;
}
echo square(4);   ?>
```

Output
16

A function can not return multiple values, but similar results can be obtained by returning an array.

Variable Scope

The scope of a variable is the context within which it is defined. For the most part all PHP variables only have a single scope. We can also introduce local scope inside user-defined function. Any variable used inside a function is by default limited to the local function scope.

Argument Passing: Passing Arguments By References

We can pass a variable by reference to a function so the function can modify the variable.

Example

```
<? Php
functionfun_ref(&$i)
{
  $i++;
}
```

```
$n=10;
$r=fun_ref($n);
echo $r;
?>
```

Output

11

PHP Form

Forms are used to get input from the user and submit it to the web server for processing. When you login into a website or into your mail box, you are interacting with a form.A form is an HTML tag that contains graphical user interface items such as input box, check boxes radio buttons etc.

The form is defined using the <form>...</form> tags and GUI items are defined using form elements such as input.

Forms can be used to edit already existing data from the database.

Creating a form

We will use HTML tags to create a form. Below is the minimal list of things you need to create a form.

- Opening and closing form tags <form>...</form>
- Form submission type POST or GET
- Submission URL that will process the submitted data
- Input fields such as input boxes, text areas, buttons, checkboxes etc.

The code below creates a simple registration form

```
<body>
<h2>Registration Form</h2>

<form action="registration_form.php" method="POST"> First name:

<input type="text" name="firstname"><br> Last name:
<input type="text" name="lastname">
<input type="hidden" name="form_submitted" value="1" />
<input type="submit" value="Submit">

</form>
</body>
```

HERE,
action="registration_form.php" method="POST"> specifies the destination URL and the submission type.

<input type="hidden" name="form_submitted" value="1"/> is a hidden value that is used to check whether the form has been submitted or not.

Submitting the form data to the server

The action attribute of the form specifies the submission URL that processes the data. The method attribute specifies the submission type.

PHP POST method

This is the built in PHP super global array variable that is used to get values submitted via HTTP POST method.

The array variable can be accessed from any script in the program; it has a global scope.

This method is ideal when you do not want to display the form post values in the URL.

A good example of using post method is when submitting login details to the server.

It has the following syntax.

```
<?php
$_POST['variable_name'];
?>
```

HERE,

"$_POST[...]" is the PHP array

"'variable_name'" is the URL variable name.

PHP GET method

This is the built in PHP super global array variable that is used to get values submitted via HTTP GET method.

The array variable can be accessed from any script in the program; it has a global scope.

This method displays the form values in the URL.

It's ideal for search engine forms as it allows the users to book mark the results.

It has the following syntax.

```
<?php
$_GET['variable_name'];
?>
```

HERE,

"$_GET[...]" is the PHP array

"'variable_name'" is the URL variable name.

Comparison of GET and POST Methods

POST	GET
Values not visible in the URL	Values visible in the URL
Has not limitation of the length of the values since they are submitted via the body of HTTP	Has limitation on the length of the values usually 255 characters. This is because the values are displayed in the URL. Note the upper limit of the characters is dependent on the browser.
Has lower performance compared to Php_GET method due to time spent encapsulation the Php_POST values in the HTTP body	Has high performance compared to POST method dues to the simple nature of appending the values in the URL.
Supports many different data types such as string, numeric, binary etc.	Supports only string data types because the values are displayed in the URL
Results cannot be book marked	Results can be book marked due to the visibility of the values in the URL

Passing Information between Pages

There are different ways by which values of variables can be passed between pages. One of the ways is to use the URL to pass the values or data. Here the biggest advantage is we can pass data to a different site

even running at different servers. Any scripting language like ASP, JSP, PHP or Perl running at receiving end can process and collect the value from the query string or from the URL.

This is where a form-handling technology like PHP comes in. PHP will catch the variable tossed from one page to the next and make it available for further use. PHP happens to be unusually good at this type of data-passing function, which makes it fast and easy to employ for a wide variety of Web site tasks.

The *get* method

The get method passes arguments from one page to the next as part of the URL query string. When used for form handling, get appends the indicated variable name(s) and value(s) to the URL designated in the action attribute with a question mark separator and submits the whole thing to the processing agent (in this case a Web server).

```
<body>
<form action="http://localhost/course.php" method="get">
<p> Course name: </p>
<select name="course" size="2">
<option value="bca">BCA</option>
<option value="bcom">BCom</option>
<option value="bsc">BSc</option>
</select></p>
<p><input type="submit" name="Submit" value="Select" /></p>
</form>
</body>
```

When the user makes a selection and clicks the Submit button, the browser combine these elements in this order, with no spaces between the elements:

- The URL in quotes after the word action (http://localhost/course.php)
- A question mark (?) denoting that the following characters constitute a query string.

- A form control name, an equal sign, and the matching value (course=bca)
- An ampersand (&) and the next NAME-VALUE pair (Submit=Select); further name-value pairs separated by ampersands can be added as many times as the server querystring-length limit allows.

The browser thus constructs the URL string:

http://localhost/ course.php?course =bca&Submit=Select

The *get* method of form handling offers one big advantage over the post method: It constructs an actual new and differentiable URL query string. Users can now bookmark this page. The result of forms using the post method is not bookmarkable.

The get method is not suitable for logins because the username and password are fully visible onscreen as well as potentially stored in the client browser's memory as a visited page.

Every get submission is recorded in the Web server log, data set included.

Because the get method assigns data to a server environment variable, the length of the URL is limited.

The *post* method

Post is the preferred method of form submission today, such as adding information to a database. The form data set is included in the body of the form when it is forwarded to the processing agent (in this case, PHP). No visible change to the URL will result according to the different data submitted.

The post method has these advantages:

- It is more secure than get because user-entered information is never visible in the URL query string, in the server logs, or (if precautions, such as always using the password HTML input type for passwords, are taken) onscreen.
- There is a much larger limit on the amount of data that can be passed (a couple of kilobytes rather than a couple of hundred characters).

Post method has these disadvantages:

- The results at a given moment cannot be bookmarked.
- The results should be expired by the browser, so that an error will result if the user employs the Back button to revisit the page.
- This method can be incompatible with certain firewall setups, which strip the form data as a security measure.

Jgjgdrj

gkfjgfdhg

PHP $_REQUEST Method

$_REQUEST is a super global variable which is widely used to collect data after submitting html forms.

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. The PHP superglobal variables are:

1. $GLOBALS
2. $_SERVER
3. $_REQUEST
4. $_POST
5. $_GET
6. $_FILES
7. $_ENV
8. $_COOKIE
9. $_SESSION

The $_REQUEST variable is a variable with the contents of $_GET and $_POST and $_COOKIE variables.

Syntax

```
<?php
  $_REQUEST['form control name ']
?>
```

Example

```
<body>
<form name="contact" method="post" action="contact.php">
        Name: <input size=25 name="name">
        Contact number: <input size=25 name="phno">
        <input type="submit" name="send" value="Submit">
</from>
</body>
```

Now in contact.php we can collect the data entered by the user in different fields using $_RQUEST.

```
<?php
        $name=$_REQUEST['name'];
        $name=$_REQUEST['phno'];
        echo $name;
        echo $phno;
?>
```

PHP String Functions

A string is a collection of characters. String is one of the data types supported by PHP. The string variables can contain alphanumeric characters. PHP provides a number of built-in string functions which help in performing several operations while dealing with string data.

1. strlen()

PHP has a predefined function to get the length of a string. Strlen() displays the length of any string. It is more commonly used in validating input fields where the user is limited to enter a fixed length of characters.

Syntax
strlen(string);

Example

```
<?php
echostrlen("Welcome");
?>
```

OUTPUT
7

Example:

```
<?php
$str="welcome";
echostrlen($str);
?>
```

OUTPUT
7

2. **Str_word_count()**

This function is used to display of the number of words in any specific string is str_word_count(). This function is also useful in validation of input fields.

Syntax:

Str_word_count(string)

Example

```
<?php
echostr_word_count("Welcome to the world of php programming");
?>
Output
7
```

3. **strrev()**

strrev() is used for reversing a string. You can use this function to get the reverse version of any string.

Syntax

strev(string)

Example

```
<?php
$str="welcome";
echo $str;

echostrrev("$str");
?>
Output
emoclew
```

4. **Strpos()**

Strpos() enables searching particular text within a string. It works simply by matching the specific text in a string. If found, then it returns the specific position. If not found at all, then it will return "False". Strops() is most commonly used in validating input fields like email.

Syntax

strpos(*string,find,start*)

Parameter Description

string :Required. Specifies the string to search

Find :Required. Specifies the string to find

start :Optional. Specifies where to begin the search

```
<?php
echostrpos("Welcome to the world of phpprogramming","world");
?>
```

Output

15

5. Str_replace()

Str_replace() is a built-in function, basically used for replacing specific text within a string.

Syntax

Str_replace(string to be replaced,text,string)

Example

```
<?php
echostr_replace("php", "the programming world", "Welcome to php");
?>
```

Output

Welcome to the programming world

6. strtolower()

Used to convert all string characters to lower case letters.

```
<?php
echostrtolower( 'Welcome');
?>
```

OUTPUT

welcome

7. **strtoupper()**

Used to convert all string characters to upper case letters.

```
<?php
echostrtoupper('welcome');
?>
```

OUTPUT
WELCOME

8. **substr()**

The substr() function returns a part of a string.

Syntax
substr(string,start,length)
Parameter Description
String : Required. Specifies the string to return a part of
Start :Required. Specifies where to start in the string

- A positive number - Start at a specified position in the string
- A negative number - Start at a specified position from the end of the string
- 0 - Start at the first character in string length

Length :Optional. Specifies the length of the returned string. Default is to the end of the string.

- A positive number - The length to be returned from the start parameter
- Negative number - The length to be returned from the end of the string

Note: If the start parameter is a negative number and length is less than or equal to start, length becomes 0.

```
<?php
// Positive numbers:
    echosubstr("Hello world",10)."<br>";
    echosubstr("Hello world",1)."<br>";
    echosubstr("Hello world",3)."<br>";
    echosubstr("Hello world",7)."<br>";
```

```
        echo "<br>";
// Negative numbers:
        echosubstr("Hello world",-1)."<br>";
        echosubstr("Hello world",-10)."<br>";
        echosubstr("Hello world",-8)."<br>";
        echosubstr("Hello world",-4)."<br>";
?>
```

OUTPUT

d
ello world
lo world
orld

d
ello world
lo world
orld

9. explode()

Used to convert strings into an array variable

Syntax

explode ($delimiter , $string)

delimiter : The boundary string

```
<?php
  $str = "welcome to the world of php programming.";
print_r (explode(" ",$str));
?>
```

This will produce following result "

Array ([0] => welcome [1] => to [2] => the [3] => world[4]=> of [5]=>php [6]=> programming)

10. strcmp()

It is used to compare two strings. If this function returns 0, the two strings are equal.

Syntax

strcmp(string1,string2)

```
<?php
 $str1='hai';
 $str2='hai';
echostrcmp($str1, $str2);
?>
```

OUTPUT

0

11. **trim()**

It used to remove the whitespaces and other characters

Syntax

trim(string,charlist)

string : It is used to specifies string to check

charlist : It specifies which character to remove

```
<?php
 $str = "php programming";
echo $str . "<br>";
echo trim($str,"php");
?>
```

OUTPUT

php programming

programming

12. **ucfirst()**

Make the first character of a string value upper case

```
<?php
echoucfirst('respect');
?>
```

OUTPUT

Respect

13. lcfirst()

Make the first character of a string value lower case

```
<?php
echolcfirst('RESPECT');
?>
```

OUTPUT

rESPECT

include and require functions in php

In PHP, there are two functions that are used to put the contents of a file containing PHP source code into another PHP file. These function are Include() and Require(). These functions are the same if but they have one difference. The difference is that the include() function produces a warning, but the script will continue execution, while the require() function produces a warning and a fatal error i.e. the script will not continue execution. These two functions are used to put the data of a file into another PHP file before it is executed by the server.

include() Function

TheInclude() function is used to put data of one PHP file into another PHP file. If errors occur then theinclude() function produces a warning but does not stop the execution of the script i.e. the script will continue to execute.

```
<html>
<body>
<a href="web/home .php">Home</a>
<a href=" web/aboutus.php /">About us</a>
<a href=" web/news.php /">News</a>
<a href=" web/gallary.php /">Gallary</a>
</body>
</html>
```

The above file is save with the name link.php. Now we will create another PHP file. Suppose that we created main.php file. The above file is included in the main.php using include().

```
<html>
<body bgcolor="pink">
<?php include("link.php"); ?>
<h2><b>Welcome </b></h2>
</body>
</html>
```

require() Function

The Require() function is also used to put data of one PHP file to another PHP file. If there are any errors then therequire() function produces a warning and a fatal error and stops the execution of the script i.e. the script will continue to execute.

```
<html>
<body bgcolor="pink">
<?php require("link.php"); ?>
<h2><b>Welcome  </b></h2>
</body>
</html>
```

The difference between include and require arises when the file being included cannot be found: include will emit a warning (E_WARNING) and the script will continue, whereas require will emit a fatal error (E_COMPILE_ERROR) and halt the script. If the file being included is critical to the rest of the script running correctly then you need to use require.

Session and cookie management

Cookies

Cookies are the text files that you can store on the user's computer. Cookies are persist even when the user's computer is turned off, so you can store information about the user easily. Cookies are used for all kind of purposes (eg: can be used for tracking users responses and requests in a web access). A cookie is often used to identify a user. With php we can both create and retrieve cookie values.

Setting Cookies

PHP provided setcookie() function to set a cookie. This function requires upto six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

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

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–the value of the cookie.(this value is stroed on the clients computer, so do not store sensitive information)

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–the path on the server in which the cookie will be available on.

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 usingHTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Example Will Create Two Cookies Name And Age These Cookies Will Be Expired After One Hour.

```
<?php
setcookie("name", "John Watkin", time()+3600, "/","", 0);
setcookie("age", "36", time()+3600, "/", "", 0);
?>
<html>
<head>
<title>Setting Cookies with PHP</title>
</head>
<body>
<?php echo "Set Cookies"?>
</body>
</html>
```

Accessing Cookies with PHP

PHP provides many ways to access cookies. Simplest way is to use either $_COOKIE or $HTTP_COOKIE_VARS variables. Following example will access all the cookies set in above example.

```
<html>
<head>
<title>Accessing Cookies with PHP</title>
</head>
<body>
<?php
echo $_COOKIE["name"]. "<br />";
/* is equivalent to */
echo $HTTP_COOKIE_VARS["name"]. "<br />";
echo $_COOKIE["age"] . "<br />";
/* is equivalent to */
echo $HTTP_COOKIE_VARS["name"] . "<br />";
?>
</body>
</html>
```

You can use isset() function to check if a cookie is set or not.

```
<html>
<head>
<title>Accessing Cookies with PHP</title>
</head>
<body>
<?php
if(isset($_COOKIE["name"]))
echo "Welcome " . $_COOKIE["name"] . "<br />";
else
echo "Sorry... Not recognized" . "<br />";
?>
</body>
</html>
```

Deleting Cookie with PHP

Officially, to delete a cookie you should call setcookie() with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired "

```
<?php
setcookie( "name", "", time()- 60, "/","", 0);
setcookie( "age", "", time()- 60, "/","", 0);
?>
<html>
<head>
<title>Deleting Cookies with PHP</title>
</head>
<body>
<?php echo "Deleted Cookies" ?>
</body>
</html>
```

Session

Web pages hold only temporary data, unless you specifically store your data on the server. One way of storing data on the server is to use sessions. Using sessions you can store and retrieve data by name.

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit. The location of the temporary file is determined by a setting in the php.ini file.

- A session is a global variable stored on the server.
- Each session is assigned a unique id which is used to retrieve stored values.
- Whenever a session is created, a cookie containing the unique session id is stored on the user's computer and returned with every request to the server. If the client browser does not support cookies, the unique php session id is displayed in the URL
- Sessions have the capacity to store relatively large data compared to cookies.

- The session values are automatically deleted when the browser is closed. If you want to store the values permanently, then you should store them in the database.
- Just like the $_COOKIE array variable, session variables are stored in the $_SESSION array variable. Just like cookies, the session must be started before any HTML tags.
- You want to store important information such as the user id more securely on the server where malicious users cannot temper with them.
- You want to pass values from one page to another.
- You want the alternative to cookies on browsers that do not support cookies.
- You want to store global variables in an efficient and more secure way compared to passing them in the URL
- You are developing an application such as a shopping cart that has to temporary store information with a capacity larger than 4KB.

Creating a Session

In order to create a session, you must first call the PHP session_start function and then store your values in the $_SESSION array variable.

Let's suppose we want to know the number of times that a page has been loaded, we can use a session to do that. The code below shows how to create and retrieve values from sessions

```
<?php
session_start(); //start the PHP_session function
if(isset($_SESSION['page_count']))
{
   $_SESSION['page_count'] += 1;
}
else
{
   $_SESSION['page_count'] = 1;
}
echo 'You are visitor number ' . $_SESSION['page_count'];
?>
```

OUTPUT:

You are visitor number 1

- Session variables are stored in associative array called $_SESSION[].
  These variables can be accessed during lifetime of a session.
- The following example starts a session then register a variable called counter that is incremented each time the page is visited during the session.
- Make use of isset() function to check if session variable is already set or not.

Destroying Session Variables

The session_destroy() function is used to destroy the whole Php session variables.

If you want to destroy a single session variable then you can use unset() function to unset a session variable. The code below illustrates how to use both methods.

```
<?php
session_destroy(); //destroy entire session
?>

<?php
unset($_SESSION['product']); //destroy product session item
?>
```

Session_destroy removes all the session data including cookies associated with the session.

Unset only frees the individual session variables.Other data remains intact.

Destroy All Session Variable

This is often used to log out of applications that store the login information in a session. You can use the code below to destroy your session completely.

```
<?php
// Begin the session
session_start();
// Unset all of the session variables.
session_unset();
// Destroy the session.
session_destroy();
?>
```

Error handling

When creating scripts and web applications, error handling is an important part. Error handling is the process of catching errors raised by your program and then taking appropriate action.

An error is an unexpected program result that cannot be handled by the program itself. Errors are resolved by fixing the program. An example of an error would be an infinite loop that never stops executing.

An exception is unexpected program result that can be handled by the program itself.

Examples of exception include trying to open a file that does not exist. This exception can be handled by either creating the file or presenting the user with an option of searching for the file.

PHP Error handling

When an error occurs, depending on your configuration settings, PHP displays the error message in the web browser with information relating to the error that occurred.

PHP offers a number of ways to handle errors. Following are the commonly used methods.

1. **Die statements**– the die function combines the echo and exit function in one. It is very useful when we want to output a message and stop the script execution when an error occurs.
2. **Custom error handlers** – these are user defined functions that are called whenever an error occurs.
3. **PHP error reporting** – the error message depending on your PHP error reporting settings. This method is very useful in development environment when you have no idea what caused the error. The information displayed can help you debug your application.

die() function

The following example shows a simple script that opens a text file:

```
<?php
$f=fopen("test.txt","r");
?>
```

If the file does not exist shows an error:

Warning: fopen(test.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, we test whether the file exist before we try to access it:

```
<?php
if(!file_exists("test.txt")) {
 die("File not found");
} else {
 $f=fopen("test.txt","r");
}
?>
```

Now if the file does not exist you get an error like this:

File not found

The code above is more efficient than the earlier code, because it uses a simple error handling mechanism to stop the script after the error. However, simply stopping the script is not always the right way to go.

Custom Error Handling Function

You can write your own function to handling any error. PHP provides you a framework to define error handling function.

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 file name 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

Possible Error levels

These error report levels are the different types of error the user-defined error handler can be used for. These values cab used in combination using |operator.

Constant	Description	Value
.E_ERROR	Fatal run-time errors. Execution of the script is halted	1
E_WARNING	Non-fatal run-time errors. Execution of the script is not halted	2
E_PARSE	Compile-time parse errors. Parse errors should only be generated by the parser.	4
E_NOTICE	Run-time notices. The script found something that might be an error, but could also happen when running a script normally	8
E_CORE_ERROR	Fatal errors that occur during PHP's initial start-up.	16
E_CORE_WARNING	Non-fatal run-time errors. This occurs during PHP's initial start-up.	32
E_USER_ERROR	Fatal user-generated error. This is like an E_ERROR set by the programmer using the PHP function trigger_error()	256

E_USER_WARNING	Non-fatal user-generated warning. This is like an E_WARNING set by the programmer using the PHP function trigger_error()	512
E_USER_NOTICE	User-generated notice. This is like an E_NOTICE set by the programmer using the PHP function trigger_error()	1024
E_STRICT	Run-time notices. Enable to have PHP suggest changes to your code which will ensure the best interoperability and forward compatibility of your code.	2048
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())	4096
E_ALL	All errors and warnings, except level E_STRICT (E_STRICT will be part of E_ALL as of PHP 6.0)	8191

All the above error level can be set using following PHP built-in library function where level cab be any of the value defined in above table.

```
interror_reporting ( [int $level] )
```

Following is the way you can create one error handling function "

```
<?php
functionhandleError($errno, $errstr,$error_file,$error_line) {
echo "<b>Error:</b> [$errno] $errstr - $error_file:$error_line";
echo "<br />";
echo "Terminating PHP Script";
die();
  }
?>
```

Once you define your custom error handler you need to set it using PHP built-in library set_error_handler function. Now lets examine our example by calling a function which does not exist.

```
<?php
error_reporting( E_ERROR );

functionhandleError($errno, $errstr,$error_file,$error_line) {
echo "<b>Error:</b> [$errno] $errstr - $error_file:$error_line";
echo "<br />";
echo "Terminating PHP Script";

die();
 }

 //set error handler
set_error_handler("handleError");

 //trigger error
myFunction();
?>
```

Exceptions Handling

PHP 5 has an exception model similar to that of other programming languages. Exceptions are important and provides a better control over error handling.

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.

When an exception is thrown, code following the statement will not be executed, and PHP will attempt to find the first matching catch block. If an exception is not caught, a PHP Fatal Error will be issued with an "Uncaught Exception ...

- An exception can be thrown, and caught ("catched") within PHP. Code may be surrounded in a try block.

- Each try must have at least one corresponding catch block. Multiple catch blocks can be used to catch different classes of exceptions.
- Exceptions can be thrown (or re-thrown) within a catch block.

Example

Following is the piece of code, copy and paste this code into a file and verify the result.

```
<?php
try {
    $error = 'Always throw this error';
throw new Exception($error);

    // Code following an exception is not executed.
echo 'Never executed';
}catch (Exception $e) {
echo 'Caught exception: ', $e->getMessage(), "\n";
  }

  // Continue execution
echo 'Hello World';
?>
```

In the above example $e->getMessage function is used to get error message. There are following functions which can be used from Exception class.

- **getMessage()** " message of exception
- **getCode()** " code of exception
- **getFile()** " source filename
- **getLine()** " source line
- **getTrace()** " n array of the backtrace()
- **getTraceAsString()** " formated string of trace

Creating Custom Exception Handler

You can define your own custom exception handler. Use following function to set a user-defined exception handler function.

```
stringset_exception_handler ( callback $exception_handler )
```

Here **exception_handler** is the name of the function to be called when an uncaught exception occurs. This function must be defined before calling set_exception_handler().

Example

```
<?php
functionexception_handler($exception) {
echo "Uncaught exception: " , $exception->getMessage(), "\n";
 }

set_exception_handler('exception_handler');
throw new Exception('Uncaught Exception');

echo "Not Executed\n";
?>
```

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. See table below for possible error report levels
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)

Now lets create a function to handle errors:

```
functioncustomError($errno, $errstr) {
 echo "<b>Error:</b> [$errno] $errstr<br>";
 echo "Ending Script";
 die();
}
```

The code above is a simple error handling function. When it is triggered, it gets the error level and an error message. It then outputs the error level and message and terminates the script.

Now that we have created an error handling function we need to decide when it should be triggered.

Set Error Handler

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.

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");
```

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.

Example

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

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

//set error handler
set_error_handler("customError");

//trigger error
echo($test);
?>
```

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

Error: [8] Undefined variable: test

Object Oriented Programming using PHP

We can imagine our universe made of different objects like sun, earth, moon etc. Similarly we can imagine our car made of different objects like wheel, steering, gear etc. Same way there is object oriented programming concepts which assume everything as an object and implement a software using different objects.

Object Oriented Concepts

Before we go in detail, lets define important terms related to Object Oriented Programming.

- **Class** " This is a programmer-defined data type, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.
- **Object** " An individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.
- **Member Variable** " These are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called attribute of the object once an object is created.
- **Member function** " These are the function defined inside a class and are used to access object data.
- **Inheritance** " When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.
- **Parent class** " A class that is inherited from by another class. This is also called a base class or super class.
- **Child Class** " A class that inherits from another class. This is also called a subclass or derived class.
- **Polymorphism** " This is an object oriented concept where same function can be used for different purposes. For example function name will remain same but ittake different number of arguments and can do different task.

- **Overloading** " a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly functions can also be overloaded with different implementation.
- **Data Abstraction** " Any representation of data in which the implementation details are hidden (abstracted).
- **Encapsulation** " refers to a concept where we encapsulate all the data and member functions together to form an object.
- **Constructor** " refers to a special type of function which will be called automatically whenever there is an object formation from a class.
- **Destructor** " refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

Defining PHP Classes

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

```
<?php
classphpClass {
var $var1;
var $var2 = "constant string";

functionmyfunc ($arg1, $arg2) {
    [..]
  }
  [..]
 }
?>
```

Here is the description of each line "

- The special form **class**, followed by the name of the class that you want to define.
- A set of braces enclosing any number of variable declarations and function definitions.
- Variable declarations start with the special form **var**, which is followed by a conventional $ variable name; they may also have an initial assignment to a constant value.

- Function definitions look much like standalone PHP functions but are local to the class and will be used to set and access object data.

Example

Here is an example which defines a class of Books type "

```
<?php
class Books {
   /* Member variables */
var $price;
var $title;

   /* Member functions */
functionsetPrice($par){
      $this->price = $par;
   }

functiongetPrice(){
echo $this->price ."<br/>";
   }

functionsetTitle($par){
      $this->title = $par;
   }

functiongetTitle(){
echo $this->title ." <br/>";
   }
  }
?>
```

The variable **$this** is a special variable and it refers to the same object ie. itself.

Creating Objects in PHP

Once you defined your class, then you can create as many objects as you like of that class type. Following is an example of how to create object using **new** operator.

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

Here we have created three objects and these objects are independent of each other and they will have their existence separately. Next we will see how to access member function and process member variables.

Calling Member Functions

After creating your objects, you will be able to call member functions related to that object. One member function will be able to process member variable of related object only.

Following example shows how to set title and prices for the three books by calling member functions.

```
$physics->setTitle( "Physics for High School" );
$chemistry->setTitle( "Advanced Chemistry" );
$maths->setTitle( "Algebra" );

$physics->setPrice( 10 );
$chemistry->setPrice( 15 );
$maths->setPrice( 7 );
```

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

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

This will produce the following result "

```
Physics for High School
Advanced Chemistry
Algebra
10
15
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.

PHP provides a special function called **__construct()** to define a constructor. You can pass as many as arguments you like into the constructor function.

Following example will create one constructor for Books class and it will initialize price and title for the book at the time of object creation.

```
function __construct( $par1, $par2 ) {
  $this->title = $par1;
  $this->price = $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( "Physics for High School", 10 );
$maths = new Books ( "Advanced Chemistry", 15 );
$chemistry = new Books ("Algebra", 7 );

/* Get those set values */
$physics->getTitle();
$chemistry->getTitle();
$maths->getTitle();

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

This will produce the following result "

```
Physics for High School
 Advanced Chemistry
 Algebra
 10
 15
 7
```

Destructor

Like a constructor function you can define a destructor function using function **__destruct()**. You can release all the resources with-in a destructor.

Inheritance

PHP class definitions can optionally inherit from a parent class definition by using the extends clause. The syntax is as follows "

```
class Child extends Parent {
<definition body>
}
```

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.

Following example inherit Books class and adds more functionality based on the requirement.

```
class Novel extends Books {
var $publisher;

functionsetPublisher($par){
    $this->publisher = $par;
  }

functiongetPublisher(){
echo $this->publisher. "<br />";
  }
}
```

Now apart from inherited functions, class Novel keeps two additional member functions.

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.

In the following example getPrice and getTitle functions are overridden to return some values.

```
functiongetPrice() {
echo $this->price . "<br/>";
return $this->price;
}
functiongetTitle(){
echo $this->title . "<br/>";
return $this->title;
}
```

Public Members

Unless you specify otherwise, properties and methods of a class are public. That is to say, they may be accessed in three possible situations "

- From outside the class in which it is declared
- From within the class in which it is declared
- From within another class that implements the class in which it is declared

Till now we have seen all members as 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

By designating a member private, you limit its accessibility to the class in which it is declared. The private member cannot be referred to from classes that inherit the class in which it is declared and cannot be accessed from outside the class.

A class member can be made private by using **private** keyword infront of the member.

```
classMyClass {
private $car = "skoda";
 $driver = "SRK";
function __construct($par) {
   // Statements here run every time
   // an instance of the class
   // is created.
 }
functionmyPublicFunction() {
return("I'm visible!");
 }
private function myPrivateFunction() {
return("I'm  not visible outside!");
 }
}
```

When *MyClass* class is inherited by another class using extends, myPublicFunction() will be visible, as will $driver. The extending class will not have any awareness of or access to myPrivateFunction and $car, because they are declared private.

Protected members

A protected property or method is accessible in the class in which it is declared, as well as in classes that extend that class. Protected members are not available outside of those two kinds of classes. A class member can be made protected by using **protected** keyword in front of the member.

Here is different version of MyClass "

```
classMyClass {
protected $car = "skoda";
 $driver = "SRK";
```

```
function __construct($par) {
   // Statements here run every time
   // an instance of the class
   // is created.
  }
functionmyPublicFunction() {
return("I'm visible!");
  }
protected function myPrivateFunction() {
return("I'm  visible in child class!");
  }
}
```

Interfaces

Interfaces are defined to provide a common function names to the implementers. Different implementers can implement those interfaces according to their requirements. You can say, interfaces are skeletons which are implemented by developers.

It is possible to define an interface, like this "

```
interface Mail {
public function sendMail();
}
```

Then, if another class implemented that interface, like this "

```
class Report implements Mail {
  // sendMail() Definition goes here
}
```

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. Once you declare a constant, it does not change.

```
classMyClass {
constrequiredMargin = 1.7;

function __construct($incomingValue) {
  // Statements here run every time
  // an instance of the class
  // is created.
 }
}
```

In this class, requiredMargin is a constant. It is declared with the keyword const, and under no circumstances can it be changed to anything other than 1.7. Note that the constant's name does not have a leading $, as variable names do.

Abstract Classes

An abstract class is one that cannot be instantiated, only inherited. You declare an abstract class with the keyword **abstract**, like this –

```
abstract class MyAbstractClass
```

When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child; additionally, these methods must be defined with the same visibility.

```
abstract class MyAbstractClass {
abstract function myAbstractFunction() {
 } }
```

Note that function definitions inside an abstract class must also be preceded by the keyword abstract. It is not legal to have abstract function definitions inside a non-abstract class.

Static Keyword

Declaring class members or methods as static makes them accessible without needing an instantiation of the class. A member declared as static can not be accessed with an instantiated class object (though a static method can).

```
Try out following example "
<?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

The final keyword, which prevents child classes from overriding a method by prefixing the definition with final. If the class itself is being defined final then it cannot be extended.

Following example results in Fatal error: Cannot override final method BaseClass::moreTesting()

```
<?php
classBaseClass {
public function test() {
echo "BaseClass::test() called<br>";
    }
final public function moreTesting() {
echo "BaseClass::moreTesting() called<br>";
    }
  }

classChildClass extends BaseClass {
public function moreTesting() {
echo "ChildClass::moreTesting() called<br>";
    }
  }
?>
```

REVIEW QUESTIONS

Part A

1. How to define a function in PHP?
2. What is GET method?
3. What is POST method?
4. What is include function?
5. What is session?
6. What are cookies?
7. What are interfaces?
8. What is the use of final keyword?
9. What is $_REQUEST method?

Part B

1. What are the different methods of pass arguments to PHP functions?
2. Explain GET and POST methods.
3. Differentiate between GET and POST methods.
4. Explain any four PHP string functions.
5. Explain include and require functions with examples.
6. How to create a session in PHP with example?
7. Explain error handling in PHP.

Part C

1. Explain session and cookie management in PHP.
2. Explain object oriented programming concepts in PHP.