

9. PHP – Arrays

An array is a data structure that stores one or more similar type of values in a single value. For example, if you want to store 100 numbers, then instead of defining 100 variables, it is easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID which is called array index.

- **Numeric array** - An array with a numeric index. Values are stored and accessed in linear fashion
- **Associative array** - An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.
- **Multidimensional array** - An array containing one or more arrays and values are accessed using multiple indices

NOTE: Built-in array functions is given in function reference [PHP Array Functions](#)

Numeric Array

These arrays can store numbers, strings and any object but their index will be represented by numbers. By default, the array index starts from zero.

Example

The following example demonstrates how to create and access numeric arrays.

Here we have used **array()** function to create array. This function is explained in function reference.

```
<html>
<body>
<?php
/* First method to create array. */
$numbers = array( 1, 2, 3, 4, 5);
foreach( $numbers as $value )
{
    echo "Value is $value <br />";
}
/* Second method to create array. */
$numbers[0] = "one";
$numbers[1] = "two";
$numbers[2] = "three";
```

```

$numbers[3] = "four";
$numbers[4] = "five";

foreach( $numbers as $value )
{
    echo "Value is $value <br />";
}
?>

</body>
</html>

```

This will produce the following result:

```

Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
Value is one
Value is two
Value is three
Value is four
Value is five

```

Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

NOTE: Don't keep associative array inside double quote while printing, otherwise it would not return any value.

Example

```

<html>
<body>
<?php
/* First method to associate create array. */

```

```

$salaries = array(
    "mohammad" => 2000,
    "qadir" => 1000,
    "zara" => 500
);

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ". $salaries['qadir']. "<br />";
echo "Salary of zara is ". $salaries['zara']. "<br />";

/* Second method to create array. */
$salaries['mohammad'] = "high";
$salaries['qadir'] = "medium";
$salaries['zara'] = "low";

echo "Salary of mohammad is ". $salaries['mohammad'] . "<br />";
echo "Salary of qadir is ". $salaries['qadir']. "<br />";
echo "Salary of zara is ". $salaries['zara']. "<br />";
?>
</body>
</html>

```

This will produce the following result:

```

Salary of mohammad is 2000
Salary of qadir is 1000
Salary of zara is 500
Salary of mohammad is high
Salary of qadir is medium
Salary of zara is low

```

Multidimensional Arrays

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple index.

Example

In this example, we create a two dimensional array to store marks of three students in three subjects:

This example is an associative array, you can create numeric array in the same fashion.

```
<html>
<body>
<?php
    $marks = array(
        "mohammad" => array
        (
            "physics" => 35,
            "maths" => 30,
            "chemistry" => 39
        ),
        "qadir" => array
        (
            "physics" => 30,
            "maths" => 32,
            "chemistry" => 29
        ),
        "zara" => array
        (
            "physics" => 31,
            "maths" => 22,
            "chemistry" => 39
        )
    );

    /* Accessing multi-dimensional array values */
    echo "Marks for mohammad in physics : " ;
    echo $marks['mohammad']['physics'] . "<br />";
    echo "Marks for qadir in maths : ";
    echo $marks['qadir']['maths'] . "<br />";
    echo "Marks for zara in chemistry : " ;
    echo $marks['zara']['chemistry'] . "<br />";
?>
</body>
</html>
```

This will produce the following result:

```
Marks for mohammad in physics : 35  
Marks for qadir in maths : 32  
Marks for zara in chemistry : 39
```

10. PHP – Strings

They are sequences of characters, like "PHP supports string operations".

NOTE: Built-in string functions is given in function reference [PHP String Functions](#)
Some valid examples of strings are as follows:

```
$string_1 = "This is a string in double quotes";  
$string_2 = "This is a somewhat longer, singly quoted string";  
$string_39 = "This string has thirty-nine characters";  
$string_0 = ""; // a string with zero characters
```

Singly quoted strings are treated almost literally, whereas doubly quoted strings replace variables with their values as well as specially interpreting certain character sequences.

```
<?  
$variable = "name";  
$literally = 'My $variable will not print!\n';  
print($literally);  
$literally = "My $variable will print!\n";  
print($literally);  
?>
```

This will produce the following result:

```
My $variable will not print!\n  
My name will print
```

There are no artificial limits on string length - within the bounds of available memory, you ought to be able to make arbitrarily long strings.

Strings that are delimited by double quotes (as in "this") are preprocessed in both the following two ways by PHP:

- Certain character sequences beginning with backslash (\) are replaced with special characters
- Variable names (starting with \$) are replaced with string representations of their values.

The escape-sequence replacements are:

- \n is replaced by the newline character
- \r is replaced by the carriage-return character
- \t is replaced by the tab character

- \\$ is replaced by the dollar sign itself (\$)
- \" is replaced by a single double-quote (")
- \\ is replaced by a single backslash (\)

String Concatenation Operator

To concatenate two string variables together, use the dot (.) operator:

```
<?php
$string1="Hello World";
$string2="1234";
echo $string1 . " " . $string2;
?>
```

This will produce the following result:

```
Hello World 1234
```

If you look at the code above, you see that we used the concatenation operator two times. This is because we had to insert a third string.

Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

Using the strlen() function

The strlen() function is used to find the length of a string.

Let's find the length of our string "Hello world!":

```
<?php
echo strlen("Hello world!");
?>
```

This will produce the following result:

```
12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string).

Using the strpos() function

The strpos() function is used to search for a string or character within a string.

If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string:

```
<?php
echo strpos("Hello world!","world");
?>
```

This will produce the following result:

```
6
```

As you can see, the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

11. PHP – Web Concepts

This session demonstrates how PHP can provide dynamic content according to browser type, randomly generated numbers or User Input. It also demonstrated how the client browser can be redirected.

Identifying Browser & Platform

PHP creates some useful **environment variables** that can be seen in the phpinfo.php page that was used to setup the PHP environment.

One of the environment variables set by PHP is **HTTP_USER_AGENT** which identifies the user's browser and operating system.

PHP provides a function getenv() to access the value of all the environment variables. The information contained in the HTTP_USER_AGENT environment variable can be used to create dynamic content appropriate to the browser.

The following example demonstrates how you can identify a client browser and operating system.

NOTE: The function preg_match() is discussed in PHP Regular expression session.

```
<html>
<body>
<?php
    $viewer = getenv( "HTTP_USER_AGENT" );
    $browser = "An unidentified browser";
    if( preg_match( "/MSIE/i", "$viewer" ) )
    {
        $browser = "Internet Explorer";
    }
    else if( preg_match( "/Netscape/i", "$viewer" ) )
    {
        $browser = "Netscape";
    }
    else if( preg_match( "/Mozilla/i", "$viewer" ) )
    {
        $browser = "Mozilla";
    }
    $platform = "An unidentified OS!";
    if( preg_match( "/Windows/i", "$viewer" ) )
```

```

{
    $platform = "Windows!";
}
else if ( preg_match( "/Linux/i", "$viewer" ) )
{
    $platform = "Linux!";
}
echo("You are using $browser on $platform");
?>
</body>
</html>

```

This is producing the following result on my machine. This result may be different for your computer, depending on what browser you are using.

```

Your browser: Google Chrome 51.0.2704.103 on windows reports:
Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/51.0.2704.103 Safari/537.36

```

Display Images Randomly

The PHP **rand()** function is used to generate a random number. This function can generate numbers with-in a given range. The random number generator should be seeded to prevent a regular pattern of numbers being generated. This is achieved using the **srand()** function that specifies the seed number as its argument.

The following example demonstrates how you can display different image each time out of four images:

```

<html>
<body>
<?php
    srand( microtime() * 1000000 );
    $num = rand( 1, 4 );

    switch( $num )

```

```

{
case 1: $image_file = "/home/images/alfa.jpg";
        break;
case 2: $image_file = "/home/images/ferrari.jpg";
        break;
case 3: $image_file = "/home/images/jaguar.jpg";
        break;
case 4: $image_file = "/home/images/porsche.jpg";
        break;
}
echo "Random Image : <img src=$image_file />";
?>
</body>
</html>

```

It will produce the following result:



Using HTML Forms

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.

Try out the following example by putting the source code in test.php script.

```

<?php
if( $_POST["name"] || $_POST["age"] )
{
    echo "Welcome ". $_POST['name']. "<br />";
    echo "You are ". $_POST['age']. " years old.";
    exit();
}

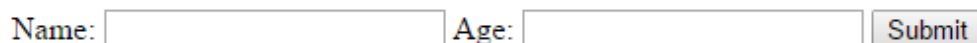
```

```

?>
<html>
<body>
  <form action="<?php $_PHP_SELF ?>" method="POST">
    Name: <input type="text" name="name" />
    Age: <input type="text" name="age" />
    <input type="submit" />
  </form>
</body>
</html>

```

It will produce the following result:



- The PHP default variable **\$_PHP_SELF** is used for the PHP script name and when you click "submit" button, the same PHP script will be called and will produce following result:
- The method = "POST" is used to post user data to the server script. There are two methods of posting data to the server script which are discussed in PHP GET & POST chapter.

Browser Redirection

The PHP **header()** function supplies raw HTTP headers to the browser and can be used to redirect it to another location. The redirection script should be at the very top of the page to prevent any other part of the page from loading.

The target is specified by the **Location:** header as the argument to the **header()** function. After calling this function the **exit()** function can be used to halt parsing of rest of the code.

The following example demonstrates how you can redirect a browser request to another web page. Try out this example by putting the source code in test.php script.

```
<?php
```

```

if( $_POST["location"] )
{
    $location = $_POST["location"];
    header( "Location:$location" );
    exit();
}
?>
<html>
<body>
    <p>Choose a site to visit :</p>
    <form action="<?php $_PHP_SELF ?>" method="POST">
    <select name="location">
        <option value="http://w3c.org">
            World Wide Web Consortium
        </option>
        <option value="http://www.google.com">
            Google Search Page
        </option>
    </select>
    <input type="submit" />
    </form>
</body>
</html>

```

It will produce the following output:

Choose a site to visit :

Tutorialspoint.com ▼

Submit

Displaying "File Download" Dialog Box

Sometime it is desired that you want to give option where a user will click a link and it will pop up a "File Download" box to the user instead of displaying actual content. This is very easy and will be achieved through HTTP header.

The HTTP header will be different from the actual header where we send **Content-Type** as **text/html\n\n**. In this case content type will be **application/octet-stream** and actual file name will be concatenated along with it.

For example, if you want make a **FileName** file downloadable from a given link, then its syntax will be as follows.

```
#!/usr/bin/perl

# HTTP Header
print "Content-Type:application/octet-stream; name=\"FileName\"\\r\\n";
print "Content-Disposition: attachment; filename=\"FileName\"\\r\\n\\n";

# Actual File Content
open( FILE, "<FileName" );
while(read(FILE, $buffer, 100) )
{
    print("$buffer");
}
```

12. PHP – GET and POST Methods

There are two ways the browser client can send information to the web server.

- The GET Method
- The POST Method

Before the browser sends the information, it encodes it using a scheme called URL encoding. In this scheme, name/value pairs are joined with equal signs and different pairs are separated by the ampersand.

```
name1=value1&name2=value2&name3=value3
```

Spaces are removed and replaced with the + character and any other non-alphanumeric characters are replaced with a hexadecimal values. After the information is encoded, it is sent to the server.

The GET Method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character.

```
http://www.test.com/index.htm?name1=value1&name2=value2
```

- The GET method produces a long string that appears in your server logs, in the browser's Location: box.
- The GET method is restricted to send up to 1024 characters only.
- Never use GET method if you have password or other sensitive information to be sent to the server.
- GET can't be used to send binary data, like images or word documents, to the server.
- The data sent by GET method can be accessed using QUERY_STRING environment variable.
- The PHP provides **\$_GET** associative array to access all the sent information using GET method.

Try out the following example by putting the source code in test.php script.

```
<?php
if( $_GET["name"] || $_GET["age"] )
{
    echo "Welcome ". $_GET['name']. "<br />";
    echo "You are ". $_GET['age']. " years old.";
```

```
        exit();
    }
?>
<html>
<body>
    <form action="<?php $_PHP_SELF ?>" method="GET">
        Name: <input type="text" name="name" />
        Age: <input type="text" name="age" />
        <input type="submit" />
    </form>
</body>
</html>
```

It will produce the following result:

Name: Age:

The POST Method

The POST method transfers information via HTTP headers. The information is encoded as described in case of GET method and put into a header called QUERY_STRING.

- The POST method does not have any restriction on data size to be sent.
- The POST method can be used to send ASCII as well as binary data.
- The data sent by POST method goes through HTTP header so security depends on HTTP protocol. By using Secure HTTP you can make sure that your information is secure.
- The PHP provides **\$_POST** associative array to access all the sent information using POST method.

Try out the following example by putting the source code in test.php script.

```
<?php
    if( $_POST["name"] || $_POST["age"] )
    {
        echo "Welcome ". $_POST['name']. "<br />";
        echo "You are ". $_POST['age']. " years old.";
        exit();
    }
?>
<html>
<body>
    <form action="<?php $_PHP_SELF ?>" method="POST">

    Name: <input type="text" name="name" />
    Age: <input type="text" name="age" />

    <input type="submit" />
    </form>
</body>
</html>
```

It will produce the following result:

Name: Age:

The \$_REQUEST variable

The PHP \$_REQUEST variable contains the contents of both \$_GET, \$_POST, and \$_COOKIE. We will discuss \$_COOKIE variable when we will explain about cookies.

The PHP \$_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Try out the following example by putting the source code in test.php script.

```
<?php
if( $_REQUEST["name"] || $_REQUEST["age"] )
{
    echo "Welcome ". $_REQUEST['name']. "<br />";
    echo "You are ". $_REQUEST['age']. " years old.";
    exit();
}
?>
<html>
<body>
    <form action="<?php $_PHP_SELF ?>" method="POST">

    Name: <input type="text" name="name" />
    Age: <input type="text" name="age" />

    <input type="submit" />
    </form>
</body>
</html>
```

Here `$_PHP_SELF` variable contains the name of self script in which it is being called.

It will produce the following result:

Name: Age:

13. PHp – File Inclusion

You can include the content of a PHP file into another PHP file before the server executes it. There are two PHP functions which can be used to included one PHP file into another PHP file.

- The include() Function
- The require() Function

This is a strong point of PHP which helps in creating functions, headers, footers, or elements that can be reused on multiple pages. This will help developers to make it easy to change the layout of complete website with minimal effort. If there is any change required, then instead of changing thousands of files just change included file.

The include() Function

The include() function takes all the text in a specified file and copies it into the file that uses the include function. If there is any problem in loading a file, then the **include()** function generates a warning but the script will continue execution.

Assume you want to create a common menu for your website. Then create a file menu.php with the following content.

```
<a href="http://www.tutorialspoint.com/index.htm">Home</a> -  
<a href="http://www.tutorialspoint.com/ebxml">ebXML</a> -  
<a href="http://www.tutorialspoint.com/ajax">AJAX</a> -  
<a href="http://www.tutorialspoint.com/perl">PERL</a> <br />
```

Now create as many pages as you like and include this file to create header. For example now your test.php file can have the following content.

```
<html>  
<body>  
  
<?php include("menu.php"); ?>  
  
<p>This is an example to show how to include PHP file!</p>  
  
</body>  
</html>
```

This will produce the following result:

```
Home -  
ebXML -  
AJAX -  
PERL  
  
This is an example to show how to include PHP file.
```

The require() Function

The `require()` function takes all the text in a specified file and copies it into the file that uses the `include` function. If there is any problem in loading a file, then the **`require()`** function generates a fatal error and halt the execution of the script.

So there is no difference in `require()` and `include()` except they handle error conditions. It is recommended to use the `require()` function instead of `include()`, because scripts should not continue executing if files are missing or misnamed.

You can try using above example with `require()` function and it will generate same result. But if you will try the following two examples where file does not exist, then you will get different results.

```
<html>
<body>
<?php include("xxmenu.php"); ?>
<p>This is an example to show how to include wrong PHP file!</p>
</body>
</html>
```

This will produce the following result:

```
This is an example to show how to include wrong PHP file!
```

Now let us try same example with `require()` function.

```
<html>
<body>
<?php require("xxmenu.php"); ?>
<p>This is an example to show how to include wrong PHP file!</p>
</body>
</html>
```

This time file execution halts and nothing is displayed.

NOTE: You may get plain warning messages or fatal error messages or nothing at all. This depends on your PHP Server configuration.

14. PHP – Files & I/O

This chapter will explain the following functions related to files:

- Opening a file
- Reading a file
- Writing a file
- Closing a file

Opening and Closing Files

The PHP **fopen()** function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate.

Files modes can be specified as one of the six options in this table.

Mode	Purpose
r	Opens the file for reading only. Places the file pointer at the beginning of the file.
r+	Opens the file for reading and writing. Places the file pointer at the beginning of the file.
w	Opens the file for writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If the file does not exist, then it attempts to create a file.
w+	Opens the file for reading and writing only. Places the file pointer at the beginning of the file. and truncates the file to zero length. If the file does not exist, then it attempts to create a file.
a	Opens the file for writing only. Places the file pointer at the end of the file. If the file does not exist, then it attempts to create a file.
a+	Opens the file for reading and writing only. Places the file pointer at the end of the file. If the file does not exist, then it attempts to create a file.

If an attempt to open a file fails, then **fopen** returns a value of **false** otherwise it returns a **file pointer** which is used for further reading or writing to that file.

After making a changes to the opened file it is important to close it with the **fclose()** function. The **fclose()** function requires a file pointer as its argument and then returns **true** when the closure succeeds or **false** if it fails.

Reading a file

Once a file is opened using **fopen()** function it can be read with a function called **fread()**. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.

The file's length can be found using the **filesize()** function which takes the file name as its argument and returns the size of the file expressed in bytes.

So here are the steps required to read a file with PHP.

- Open a file using **fopen()** function.
- Get the file's length using **filesize()** function.
- Read the file's content using **fread()** function.
- Close the file with **fclose()** function.

The following example assigns the content of a text file to a variable and then displays those contents on the web page.

```
<html>
<head>
<title>Reading a file using PHP</title>
</head>
<body>

<?php
$filename = "/home/user/guest/tmp.txt";
$file = fopen( $filename, "r" );
if( $file == false )
{
    echo ( "Error in opening file" );
    exit();
}
$filesize = filesize( $filename );
$filetext = fread( $file, $filesize );

fclose( $file );

echo ( "File size : $filesize bytes" );
```

```

echo ( "<pre>$filetext</pre>" );
?>

</body>
</html>

```

It will produce the following result:

File size : 278 bytes

```

The PHP Hypertext Preprocessor (PHP) is a programming
language that allows web developers to create dynamic
content that interacts with databases.
PHP is basically used for developing web based software
applications. This tutorial helps you to build your base
with PHP.

```

Writing a File

A new file can be written or text can be appended to an existing file using the PHP **fwrite()** function. This function requires two arguments specifying a **file pointer** and the string of data that is to be written. Optionally a third integer argument can be included to specify the length of the data to write. If the third argument is included, writing would stop after the specified length has been reached.

The following example creates a new text file and then writes a short text heading inside it. After closing this file its existence is confirmed using **file_exists()** function which takes file name as an argument

```

<?php
$filename = "/home/user/guest/newfile.txt";
$file = fopen( $filename, "w" );
if( $file == false )
{
    echo ( "Error in opening new file" );
    exit();
}
fwrite( $file, "This is a simple test\n" );
fclose( $file );
?>

```

```
<html>
<head>
<title>Writing a file using PHP</title>
</head>
<body>

<?php
if( file_exist( $filename ) )
{
    $filesize = filesize( $filename );
    $msg = "File  created with name $filename ";
    $msg .= "containing $filesize bytes";
    echo ($msg );
}
else
{
    echo ("File $filename does not exit" );
}
?>
</body>
</html>
```

It will produce the following result:

Error in opening new file

We have covered all the function related to file input and out in the [PHP File System Function](#) chapter.

15. PHP – Functions

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

You already have seen many functions like **fopen()** and **fread()** etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you:

- Creating a PHP Function
- Calling a PHP Function

In fact, you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

Please refer to [PHP Function Reference](#) for a complete set of useful functions.

Creating PHP Function

It is very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it.

The following example creates a function called `writeMessage()` and then calls it just after creating it.

Note that while creating a function its name should start with keyword **function** and all the PHP code should be put inside { and } braces as shown in the following example below:

```
<html>
<head>
<title>Writing PHP Function</title>
</head>
<body>

<?php
/* Defining a PHP Function */
function writeMessage()
{
    echo "You are really a nice person, Have a nice time!";
}
/* Calling a PHP Function */
```

```
writeMessage();  
?>  
</body>  
</html>
```

This will display the following result:

```
You are really a nice person, Have a nice time!
```

PHP Functions with Parameters

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters you like. These parameters work like variables inside your function. The following example takes two integer parameters and adds them together and then prints them.

```
<html>  
<head>  
<title>Writing PHP Function with Parameters</title>  
</head>  
<body>  
  
<?php  
function addFunction($num1, $num2)  
{  
    $sum = $num1 + $num2;  
    echo "Sum of the two numbers is : $sum";  
}  
addFunction(10, 20);  
?>  
</body>  
</html>
```

This will display the following result:

```
Sum of the two numbers is : 30
```

Passing Arguments by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

The following example depicts both the cases.

```
<html>
<head>
<title>Passing Argument by Reference</title>
</head>
<body>
<?php
function addFive($num)
{
    $num += 5;
}

function addSix(&$num)
{
    $num += 6;
}
$orignum = 10;
addFive( &$orignum );
echo "Original Value is $orignum<br />";
addSix( $orignum );
echo "Original Value is $orignum<br />";
?>
</body>
</html>
```

This will display the following result:

```
Original Value is 10
Original Value is 16
```

PHP Functions returning value

A function can return a value using the **return** statement in conjunction with a value or object. **return** stops the execution of the function and sends the value back to the calling code.

You can return more than one value from a function using **return array(1,2,3,4)**.

The following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that **return** keyword is used to return a value from a function.

```
<html>
<head>
<title>Writing PHP Function which returns value</title>
</head>
<body>
```

```
<?php
function addFunction($num1, $num2)
{
    $sum = $num1 + $num2;
    return $sum;
}
$return_value = addFunction(10, 20);
echo "Returned value from the function : $return_value";
?>
</body>
</html>
```

This will display the following result:

```
Returned value from the function : 30
```

Setting Default Values for Function Parameters

You can set a parameter to have a default value if the function's caller doesn't pass it.

The following function prints NULL in case use does not pass any value to this function.

```
<html>
<head>
<title>Writing PHP Function which returns value</title>
</head>
<body>

<?php
function printMe($param = NULL)
{
    print $param;
}
printMe("This is test");
printMe();
?>

</body>
</html>
```

This will produce the following result:

```
This is test
```

Dynamic Function Calls

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself. The following example depicts this behavior.

```
<html>
<head>
<title>Dynamic Function Calls</title>
</head>
<body>
<?php
function sayHello()
{
    echo "Hello<br />";
}
$function_holder = "sayHello";
$function_holder();
?>
</body>
</html>
```

This will display the following result:

```
Hello
```

16. PHP – Cookies

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users:

- Server script sends a set of cookies to the browser. For example, name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- Next time, when the browser sends any request to the web server, it sends those cookies information to the server and server uses that information to identify the user.

This chapter will teach you how to set cookies, how to access them and how to delete them.

The Anatomy of a Cookie

Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser). A PHP script that sets a cookie might send headers that look something like this:

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
           path=/; domain=tutorialspoint.com
Connection: close
Content-Type: text/html
```

As you can see, the Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server. The browser's headers might look something like this:

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
```

```
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

A PHP script will then have access to the cookie in the environmental variables `$_COOKIE` or `$HTTP_COOKIE_VARS[]` which holds all cookie names and values. Above cookie can be accessed using `$HTTP_COOKIE_VARS["name"]`.

Setting Cookies with PHP

PHP provided **setcookie()** function to set a cookie. This function requires up to 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** - This sets the value of the named variable and is the content that you actually want to store.
- **Expiry** - This specifies 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 means cookie can be sent by regular HTTP.

The following example will create two cookies **name** and **age**. These cookies will expire after an 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. The simplest way is to use either `$_COOKIE` or `$HTTP_COOKIE_VARS` variables. The 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["age"] . "<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>
```

17. PHP – Sessions

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session.

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 called **session.save_path**. Before using any session variable make sure you have setup this path. When a session is started, the following actions take place:

- PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443.
- A cookie called **PHPSESSID** is automatically sent to the user's computer to store unique session identification string.
- A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess_ ie sess_3c7foj34c3jj973hjkop2fc937e3443.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

Starting a PHP Session

A PHP session is easily started by making a call to the **session_start()** function. This function first checks if a session is already started and if none is started, then it starts one. It is recommended to put the call to **session_start()** at the beginning of the page.

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 and then registers 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.

Put this code in a test.php file and load this file many times to see the result:

```
<?php
    session_start();
    if( isset( $_SESSION['counter'] ) )
    {
        $_SESSION['counter'] += 1;
    }
    else
    {
        $_SESSION['counter'] = 1;
    }
    $msg = "You have visited this page ". $_SESSION['counter'];
    $msg .= "in this session.";
?>
<html>
<head>
<title>Setting up a PHP session</title>
</head>
<body>
<?php echo ( $msg ); ?>
</body>
</html>
```

It will produce the following result:

You have visited this page 1 in this session.

Destroying a PHP Session

A PHP session can be destroyed by **session_destroy()** function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable, then you can use **unset()** function to unset a session variable.

Here is the example to unset a single variable:

```
<?php
    unset($_SESSION['counter']);
?>
```

Here is the call which will destroy all the session variables:

```
<?php
    session_destroy();
?>
```

Turning on Auto Session

You don't need to call `start_session()` function to start a session when a user visits your site if you can set **session.auto_start** variable to 1 in **php.ini** file.

Sessions without cookies

There may be a case when a user does not allow to store cookies on their machine. So there is another method to send session ID to the browser.

Alternatively, you can use the constant `SID` which is defined if the session started. If the client did not send an appropriate session cookie, it has the form `session_name=session_id`. Otherwise, it expands to an empty string. Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using `SID`.

```
<?php
    session_start();

    if (isset($_SESSION['counter'])) {
        $_SESSION['counter'] = 1;
    } else {
        $_SESSION['counter']++;
    }

    ?>

    $msg = "You have visited this page ". $_SESSION['counter'];
```

```
$msg .= "in this session.";
echo ( $msg );
<p>
To continue click following link <br />
<a href="nextpage.php?<?php echo htmlspecialchars(SID); >">
</p>
```

It will produce the following result:

You have visited this page 1 in this session.

To continue click following link

The **htmlspecialchars()** may be used when printing the SID in order to prevent XSS related attacks.

18. PHP – Sending Emails

PHP must be configured correctly in the **php.ini** file with the details of how your system sends email. Open php.ini file available in **/etc/** directory and find the section headed **[mail function]**.

Windows users should ensure that two directives are supplied. The first is called SMTP that defines your email server address. The second is called sendmail_from which defines your own email address.

The configuration for Windows should look something like this:

```
[mail function]
; For Win32 only.
SMTP = smtp.secureserver.net

; For win32 only
sendmail_from = webmaster@tutorialspoint.com
```

Linux users simply need to let PHP know the location of their sendmail application. The path and any desired switches should be specified to the sendmail_path directive.

The configuration for Linux should look something like this:

```
[mail function]
; For Win32 only.
SMTP =

; For win32 only
sendmail_from =

; For Unix only
sendmail_path = /usr/sbin/sendmail -t -i
```

Now you are ready to go.

Sending plain text 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 message and the actual message additionally there are other two optional parameters.

```
mail( to, subject, message, headers, parameters );
```

Here is the description for each 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

As soon as the mail function is called, PHP will attempt to send the email, then it will return true if successful or false if it is failed.

Multiple recipients can be specified as the first argument to the mail() function in a comma separated list.

Example

The following example will send an HTML email message to xyz@somedomain.com. You can code this program in such a way that it should receive all content from the user and then it should send an email.

```
<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>

```

Sending HTML email

When you send a text message using PHP, then all the content will be treated as simple text. Even if you will include HTML tags in a text message, it will be displayed as simple text and HTML tags will not be formatted according to HTML syntax. But PHP provides option to send an HTML message as actual HTML message.

While sending an email message, you can specify a Mime version, content type and character set to send an HTML email.

Example

The following example will send an HTML email message to xyz@somedomain.com copying it to afgh@somedomain.com. You can code this program in such a way that it should receive all content from the user and then it should send an email.

```

<html>
<head>
<title>Sending HTML email using PHP</title>
</head>
<body>
<?php
    $to = "xyz@somedomain.com";
    $subject = "This is subject";
    $message = "<b>This is HTML message.</b>";
    $message .= "<h1>This is headline.</h1>";
    $header = "From:abc@somedomain.com \r\n";
    $header = "Cc:afgh@somedomain.com \r\n";
    $header .= "MIME-Version: 1.0\r\n";
    $header .= "Content-type: text/html\r\n";
    $retval = mail ($to,$subject,$message,$header);
    if( $retval == true )
    {
        echo "Message sent successfully...";
    }
    else
    {

```



```

        echo "Message could not be sent...";
    }
?>
</body>
</html>

```

Sending attachments with email

To send an email with mixed content requires to set **Content-type** header to **multipart/mixed**. Then text and attachment sections can be specified within **boundaries**.

A boundary is started with two hyphens followed by a unique number which can not appear in the message part of the email. A PHP function **md5()** is used to create a 32 digit hexadecimal number to create unique number. A final boundary denoting the email's final section must also end with two hyphens.

Attached files should be encoded with the **base64_encode()** function for safer transmission and are best split into chunks with the **chunk_split()** function. This adds **\r\n** inside the file at regular intervals, normally every 76 characters.

Following is the example which will send a file **/tmp/test.txt** as an attachment. you can code your program to receive an uploaded file and send it.

```

<html>
<head>
<title>Sending attachment using PHP</title>
</head>
<body>
<?php
    $to = "xyz@somedomain.com";
    $subject = "This is subject";
    $message = "This is test message.";
    # Open a file
    $file = fopen( "/tmp/test.txt", "r" );
    if( $file == false )
    {
        echo "Error in opening file";
        exit();
    }
    # Read the file into a variable
    $size = filesize("/tmp/test.txt");
    $content = fread( $file, $size);

```

```

# encode the data for safe transit
# and insert \r\n after every 76 chars.
$encoded_content = chunk_split( base64_encode($content));

# Get a random 32 bit number using time() as seed.
$num = md5( time() );

# Define the main headers.
$header = "From:xyz@somedomain.com\r\n";
$header .= "MIME-Version: 1.0\r\n";
$header .= "Content-Type: multipart/mixed; ";
$header .= "boundary=$num\r\n";
$header .= "--$num\r\n";

# Define the message section
$header .= "Content-Type: text/plain\r\n";
$header .= "Content-Transfer-Encoding:8bit\r\n\r\n";
$header .= "$message\r\n";
$header .= "--$num\r\n";

# Define the attachment section
$header .= "Content-Type: multipart/mixed; ";
$header .= "name=\"test.txt\"\r\n";
$header .= "Content-Transfer-Encoding:base64\r\n";
$header .= "Content-Disposition:attachment; ";
$header .= "filename=\"test.txt\"\r\n\r\n";
$header .= "$encoded_content\r\n";
$header .= "--$num--";

# Send email now
$retval = mail ( $to, $subject, "", $header );
if( $retval == true )
{
    echo "Message sent successfully...";
}
else
{

```

```
        echo "Message could not be sent...";  
    }  
?>  
</body>  
</html>
```

You try all the above examples. If you face any problems, then you can post it in our discussion forum.

19. PHP – File Uploading

A PHP script can be used with a HTML form to allow users to upload files to the server. Initially files are uploaded into a temporary directory and then relocated to a target destination by a PHP script.

Information in the **phpinfo.php** page describes the temporary directory that is used for file uploads as **upload_tmp_dir** and the maximum permitted size of files that can be uploaded is stated as **upload_max_filesize**. These parameters are set into PHP configuration file **php.ini**

The process of uploading a file follows these steps:

- The user opens the page containing a HTML form featuring a text field, a browse button and a submit button.
- The user clicks the browse button and selects a file to upload from the local PC.
- The full path to the selected file appears in the text field, then the user clicks the submit button.
- The selected file is sent to the temporary directory on the server.
- The PHP script that was specified as the form handler in the form's action attribute checks that the file has arrived and then copies the file into an intended directory.
- The PHP script confirms the success to the user.

As usual, while writing files, it is necessary for both temporary and final locations to have permissions set that enable file writing. If either is set to be read-only, then process will fail.

An uploaded file could be a text file or image file or any document.

Creating an Upload Form

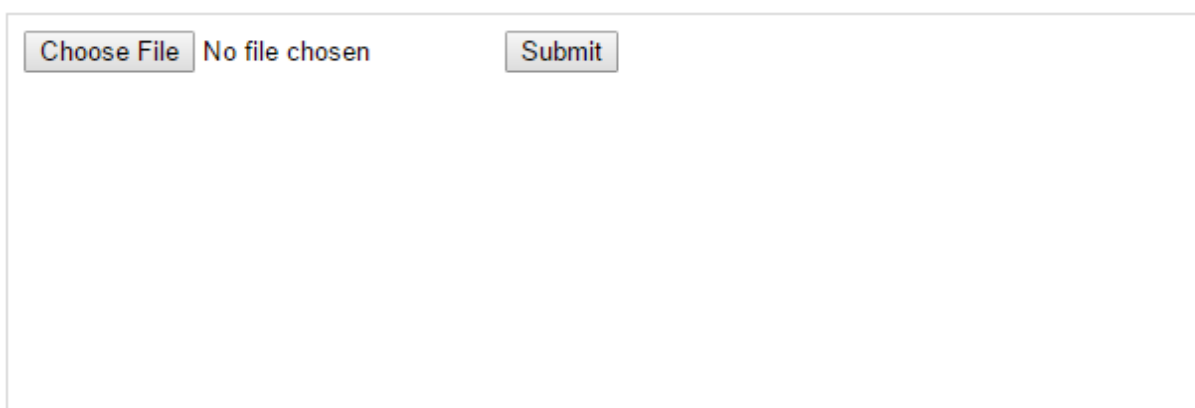
The following HTML code creates an uploader form. This form is having method attribute set to **post** and enctype attribute is set to **multipart/form-data**

```
<html>

<head>
<title>File Uploading Form</title>
</head>
<body>
<h3>File Upload:</h3>
Select a file to upload: <br />
```

```
<form action="/php/file_uploader.php" method="post"
        enctype="multipart/form-data">
<input type="file" name="file" size="50" />
<br />
<input type="submit" value="Upload File" />
</form>
</body>
</html>
```

This will display the following result:



The screenshot shows a web form with a light gray border. At the top, there is a 'Choose File' button, followed by the text 'No file chosen', and then a 'Submit' button. The rest of the form area is empty.

Creating an upload script

There is one global PHP variable called **\$_FILES**. This variable is an associate double dimension array and keeps all the information related to uploaded file. So if the value assigned to the input's name attribute in uploading form was **file**, then PHP would create the following five variables:

- **\$_FILES['file']['tmp_name']**- the uploaded file in the temporary directory on the web server.
- **\$_FILES['file']['name']** - the actual name of the uploaded file.
- **\$_FILES['file']['size']** - the size in bytes of the uploaded file.
- **\$_FILES['file']['type']** - the MIME type of the uploaded file.
- **\$_FILES['file']['error']** - the error code associated with this file upload.

The following example should allow upload images and return the uploaded file information.

```
<?php
if( $_FILES['file']['name'] != "" )
{
    copy( $_FILES['file']['name'], "/var/www/html" ) or
        die( "Could not copy file!");
}
else
{
    die("No file specified!");
}
?>
<html>
<head>
<title>Uploading Complete</title>
</head>
<body>
<h2>Uploaded File Info:</h2>
<ul>
<li>Sent file: <?php echo $_FILES['file']['name']; ?>
<li>File size: <?php echo $_FILES['file']['size']; ?> bytes
<li>File type: <?php echo $_FILES['file']['type']; ?>
</ul>
</body>
</html>
```

It will produce the following result:

No file chosen

- Sent file:
- File size:
- File type:

20. PHP – Coding Standard

Every company follows a different coding standard based on their best practices. Coding standard is required because there may be many developers working on different modules. If they start inventing their own standards, then the source will become very unmanageable and it will become difficult to maintain that source code in future.

Here are several reasons why to use coding specifications:

- Your peer programmers have to understand the code you produce. A coding standard acts as the blueprint for all the team to decipher the code.
- Simplicity and clarity achieved by consistent coding saves you from common mistakes.
- If you revise your code after some time, then it becomes easy to understand that code.

There are a few guidelines which can be followed while coding in PHP.

- **Indenting and Line Length** - Use an indent of 4 spaces and don't use any tab because different computers use different setting for tab. It is recommended to keep lines at approximately 75-85 characters long for better code readability.
- **Control Structures** - These include if, for, while, switch, etc. Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls. You are strongly encouraged to always use curly braces even in situations where they are technically optional.

Example

```
if ((condition1) || (condition2)) {  
    action1;  
} elseif ((condition3) && (condition4)) {  
    action2;  
} else {  
    default action;  
}
```

You can write switch statements as follows:

```
switch (condition) {  
case 1:  
    action1;  
    break;
```

```

case 2:
    action2;
    break;

default:
    defaultaction;
    break;
}

```

Function Calls - Functions should be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon. Here's an example:

```
$var = foo($bar, $baz, $quux);
```

Function Definitions - Function declarations follow the "BSD/Allman style":

```

function fooFunction($arg1, $arg2 = '')
{
    if (condition) {
        statement;
    }
    return $val;
}

```

Comments - C style comments (`/* */`) and standard C++ comments (`//`) are both fine. Use of Perl/shell style comments (`#`) is discouraged.

PHP Code Tags - Always use `<?php ?>` to delimit PHP code, not the `<? ?>` shorthand. This is required for PHP compliance and is also the most portable way to include PHP code on differing operating systems and setups.

Variable Names -

- Use all lower case letters
- Use `'_'` as the word separator.
- Global variables should be prepended with a `'g'`.
- Global constants should be all caps with `'_'` separators.
- Static variables may be prepended with `'s'`.

Make Functions Reentrant - Functions should not keep static variables that prevent a function from being reentrant.

Alignment of Declaration Blocks - Block of declarations should be aligned.

One Statement Per Line - There should be only one statement per line unless the statements are very closely related.

Short Methods or Functions - Methods should limit themselves to a single page of code. There could be many more points which should be considered while writing your PHP program. Over all intension should be to be consistent throughout of the code programming and it will be possible only when you will follow any coding standard. You can device your own standard if you like something different.

Part 2: Advanced PHP

21. PHP – Predefined Variables

PHP provides a large number of predefined variables to any script which it runs. PHP provides an additional set of predefined arrays containing variables from the web server the environment, and user input. These new arrays are called superglobals:

All the following variables are automatically available in every scope.

PHP Superglobals

Variable	Description
<code>\$GLOBALS</code>	Contains a reference to every variable which is currently available within the global scope of the script. The keys of this array are the names of the global variables.
<code>\$_SERVER</code>	This is an array containing information such as headers, paths, and script locations. The entries in this array are created by the web server. There is no guarantee that every web server will provide any of these. See next section for a complete list of all the SERVER variables.
<code>\$_GET</code>	An associative array of variables passed to the current script via the HTTP GET method.
<code>\$_POST</code>	An associative array of variables passed to the current script via the HTTP POST method.
<code>\$_FILES</code>	An associative array of items uploaded to the current script via the HTTP POST method.
<code>\$_REQUEST</code>	An associative array consisting of the contents of <code>\$_GET</code> , <code>\$_POST</code> , and <code>\$_COOKIE</code> .
<code>\$_COOKIE</code>	An associative array of variables passed to the current script via HTTP cookies.
<code>\$_SESSION</code>	An associative array containing session variables available to the current script.
<code>\$_PHP_SELF</code>	A string containing PHP script file name in which it is called.
<code>\$php_errormsg</code>	<code>\$php_errormsg</code> is a variable containing the text of the last error message generated by PHP.

Server variables: \$_SERVER

\$_SERVER is an array containing information such as headers, paths, and script locations. The entries in this array are created by the web server. There is no guarantee that every web server will provide any of these.

Variable	Description
\$_SERVER['PHP_SELF']	The filename of the currently executing script, relative to the document root
\$_SERVER['argv']	Array of arguments passed to the script. When the script is run on the command line, this gives C-style access to the command line parameters. When called via the GET method, this will contain the query string.
\$_SERVER['argc']	Contains the number of command line parameters passed to the script if run on the command line.
\$_SERVER['GATEWAY_INTERFACE']	What revision of the CGI specification the server is using; i.e. 'CGI/1.1'.
\$_SERVER['SERVER_ADDR']	The IP address of the server under which the current script is executing.
\$_SERVER['SERVER_NAME']	The name of the server host under which the current script is executing. If the script is running on a virtual host, this will be the value defined for that virtual host.
\$_SERVER['SERVER_SOFTWARE']	Server identification string, given in the headers when responding to requests.
\$_SERVER['SERVER_PROTOCOL']	Name and revision of the information protocol via which the page was requested; i.e. 'HTTP/1.0';
\$_SERVER['REQUEST_METHOD']	Which request method was used to access the page; i.e. 'GET', 'HEAD', 'POST', 'PUT'.
\$_SERVER['REQUEST_TIME']	The timestamp of the start of the request. Available since PHP 5.1.0.
\$_SERVER['QUERY_STRING']	The query string, if any, via which the page was accessed.
\$_SERVER['DOCUMENT_ROOT']	The document root directory under which the current script is executing, as defined in the server's configuration file.

<code>\$_SERVER['HTTP_ACCEPT']</code>	Contents of the Accept: header from the current request, if there is one.
<code>\$_SERVER['HTTP_ACCEPT_CHARSET']</code>	Contents of the Accept-Charset: header from the current request, if there is one. Example: 'iso-8859-1,* ,utf-8'.
<code>\$_SERVER['HTTP_ACCEPT_ENCODING']</code>	Contents of the Accept-Encoding: header from the current request, if there is one. Example: 'gzip'.
<code>\$_SERVER['HTTP_ACCEPT_LANGUAGE']</code>	Contents of the Accept-Language: header from the current request, if there is one. Example: 'en'.
<code>\$_SERVER['HTTP_CONNECTION']</code>	Contents of the Connection: header from the current request, if there is one. Example: 'Keep-Alive'.
<code>\$_SERVER['HTTP_HOST']</code>	Contents of the Host: header from the current request, if there is one.
<code>\$_SERVER['HTTP_REFERER']</code>	The address of the page (if any) which referred the user agent to the current page.
<code>\$_SERVER['HTTP_USER_AGENT']</code>	This is a string denoting the user agent being which is accessing the page. A typical example is: Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586).
<code>\$_SERVER['HTTPS']</code>	Set to a non-empty value if the script was queried through the HTTPS protocol.
<code>\$_SERVER['REMOTE_ADDR']</code>	The IP address from which the user is viewing the current page.
<code>\$_SERVER['REMOTE_HOST']</code>	The Host name from which the user is viewing the current page. The reverse dns lookup is based off the REMOTE_ADDR of the user.
<code>\$_SERVER['REMOTE_PORT']</code>	The port being used on the user's machine to communicate with the web server.
<code>\$_SERVER['SCRIPT_FILENAME']</code>	The absolute pathname of the currently executing script.
<code>\$_SERVER['SERVER_ADMIN']</code>	The value given to the SERVER_ADMIN (for Apache) directive in the web server configuration file.
<code>\$_SERVER['SERVER_PORT']</code>	The port on the server machine being used by the web server for communication. For default setups, this will be '80'.

<code>\$_SERVER['SERVER_SIGNATURE']</code>	String containing the server version and virtual host name which are added to server-generated pages, if enabled.
<code>\$_SERVER['PATH_TRANSLATED']</code>	Filesystem based path to the current script.
<code>\$_SERVER['SCRIPT_NAME']</code>	Contains the current script's path. This is useful for pages which need to point to themselves.
<code>\$_SERVER['REQUEST_URI']</code>	The URI which was given in order to access this page; for instance, '/index.html'.
<code>\$_SERVER['PHP_AUTH_DIGEST']</code>	When running under Apache as module doing Digest HTTP authentication this variable is set to the 'Authorization' header sent by the client.
<code>\$_SERVER['PHP_AUTH_USER']</code>	When running under Apache or IIS (ISAPI on PHP 5) as module doing HTTP authentication this variable is set to the username provided by the user.
<code>\$_SERVER['PHP_AUTH_PW']</code>	When running under Apache or IIS (ISAPI on PHP 5) as module doing HTTP authentication this variable is set to the password provided by the user.
<code>\$_SERVER['AUTH_TYPE']</code>	When running under Apache as module doing HTTP authenticated this variable is set to the authentication type.

22. PHP – Regular Expression

Regular expressions are nothing more than a sequence or pattern of characters itself. They provide the foundation for pattern-matching functionality.

Using regular expression you can search a particular string inside another string, you can replace one string by another string and you can split a string into many chunks.

PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression. You can use any of them based on your comfort.

- POSIX Regular Expressions
- PERL Style Regular Expressions

POSIX Regular Expressions

The structure of a POSIX regular expression is not dissimilar to that of a typical arithmetic expression: various elements (operators) are combined to form more complex expressions.

The simplest regular expression is one that matches a single character, such as `g`, inside strings such as `g`, `haggle`, or `bag`.

Let us discuss a few concepts being used in POSIX regular expression. After that, we will introduce you to regular expression related functions.

Brackets

Brackets (`[]`) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Expression	Description
<code>[0-9]</code>	It matches any decimal digit from 0 through 9.
<code>[a-z]</code>	It matches any character from lowercase a through lowercase z.
<code>[A-Z]</code>	It matches any character from uppercase A through uppercase Z.
<code>[a-Z]</code>	It matches any character from lowercase a through uppercase Z.

The ranges shown above are general; you could also use the range `[0-3]` to match any decimal digit ranging from 0 through 3, or the range `[b-v]` to match any lowercase character ranging from b through v.

Quantifiers

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character having a specific connotation. The +, *, ?, {int. range}, and \$ flags all follow a character sequence.

Expression	Description
p+	It matches any string containing at least one p.
p*	It matches any string containing zero or more p's.
p?	It matches any string containing zero or more p's. This is just an alternative way to use p*.
p{N}	It matches any string containing a sequence of N p's
p{2,3}	It matches any string containing a sequence of two or three p's.
p{2, }	It matches any string containing a sequence of at least two p's.
p\$	It matches any string with p at the end of it.
^p	It matches any string with p at the beginning of it.

Examples

Following examples will clear your concepts about matching characters.

Expression	Description
[^a-zA-Z]	It matches any string not containing any of the characters ranging from a through z and A through Z.
p.p	It matches any string containing p, followed by any character, in turn followed by another p.
^. {2}\$	It matches any string containing exactly two characters.
(.*	It matches any string enclosed within and .
p(hp)*	It matches any string containing a p followed by zero or more instances of the sequence hp.

Predefined Character Ranges

For your programming convenience several predefined character ranges, also known as character classes, are available. Character classes specify an entire range of characters, for example, the alphabet or an integer set:

Expression	Description
<code>[:alpha:]</code>	It matches any string containing alphabetic characters aA through zZ.
<code>[:digit:]</code>	It matches any string containing numerical digits 0 through 9.
<code>[:alnum:]</code>	It matches any string containing alphanumeric characters aA through zZ and 0 through 9.
<code>[:space:]</code>	It matches any string containing a space.

PHP's Regexp POSIX Functions

PHP currently offers seven functions for searching strings using POSIX-style regular expressions:

Function	Description
<code>ereg()</code>	The <code>ereg()</code> function searches a string specified by string for a string specified by pattern, returning true if the pattern is found, and false otherwise.
<code>ereg_replace()</code>	The <code>ereg_replace()</code> function searches for string specified by pattern and replaces pattern with replacement if found.
<code>eregi()</code>	The <code>eregi()</code> function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive.
<code>eregi_replace()</code>	The <code>eregi_replace()</code> function operates exactly like <code>ereg_replace()</code> , except that the search for pattern in string is not case sensitive.
<code>split()</code>	The <code>split()</code> function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.
<code>spliti()</code>	The <code>spliti()</code> function operates exactly in the same manner as its sibling <code>split()</code> , except that it is not case sensitive.
<code>sql_regcase()</code>	The <code>sql_regcase()</code> function can be thought of as a utility function, converting each character in the input parameter string into a bracketed expression containing two characters.

PHP – Function `ereg()`

Syntax

```
int ereg(string pattern, string originalstring, [array regs]);
```

Definition and Usage

The `ereg()` function searches a string specified by `string` for a string specified by `pattern`, returning `true` if the pattern is found, and `false` otherwise. The search is case sensitive in regard to alphabetical characters.

The optional input parameter `regs` contains an array of all matched expressions that were grouped by parentheses in the regular expression.

Return Value

- It returns `true` if the pattern is found, and `false` otherwise.

Example

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

```
<?php
$email_id = "admin@tutorialspoint.com";
$retval = ereg("(\\.) (com$)", $email_id);

if( $retval == true )
{
    echo "Found a .com<br>";
}
else
{
    echo "Could not found a .com<br>";
}

$retval = ereg("(\\.) (com$)", $email_id, $regs);

if( $retval == true )
{
    echo "Found a .com and reg = ". $regs[0];
}
else
{
    echo "Could not found a .com";
}

?>
```

This will produce the following result –

```
Found a .com  
Found a .com and reg = .com
```

PHP – Function `ereg_replace()`

Syntax

```
string ereg_replace (string pattern, string replacement, string  
originalstring);
```

Definition and Usage

The `ereg_replace()` function searches for string specified by pattern and replaces pattern with replacement if found. The `ereg_replace()` function operates under the same premises as `ereg()`, except that the functionality is extended to finding and replacing pattern instead of simply locating it.

Like `ereg()`, `ereg_replace()` is case sensitive.

Return Value

- After the replacement has occurred, the modified string will be returned.
- If no matches are found, the string will remain unchanged.

Example

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

```
<?php  
    $copy_date = "Copyright 1999";  
    $copy_date = ereg_replace("([0-9]+)", "2000", $copy_date);  
  
    print $copy_date;  
?>
```

This will produce the following result –

Copyright 2000

PHP — Function eregi()

Syntax

```
int eregi(string pattern, string string, [array regs]);
```

Definition and Usage

The eregi() function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive. Eregi() can be particularly useful when checking the validity of strings, such as passwords.

The optional input parameter regs contains an array of all matched expressions that were grouped by parentheses in the regular expression.

Return Value

- It returns true if the pattern is validated, and false otherwise.

Example

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

```
<?php
$password = "abc";

if (! eregi ("[:alnum:]{8,10}", $password))
{
    print "Invalid password! Passwords must be from 8 - 10 chars";
}
else
{
    print "Valid password";
}
```

```
}
?>
```

This will produce the following result –

```
Invalid password! Passwords must be from 8 - 10 chars
```

PHP — Function eregi_replace()

Syntax

```
string eregi_replace (string pattern, string replacement, string
originalstring);
```

Definition and Usage

The eregi_replace() function operates exactly like ereg_replace(), except that the search for pattern in string is not case sensitive.

Return Value

- After the replacement has occurred, the modified string will be returned.
- If no matches are found, the string will remain unchanged.

Example

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

```
<?php
    $copy_date = "Copyright 2000";
    $copy_date = eregi_replace("([a-z]+)", "&Copy;", $copy_date);

    print $copy_date;
?>
```

This will produce the following result –

```
&Copy; 2000
```

PHP – Function split()

Syntax

```
array split (string pattern, string string [, int limit])
```

Definition and Usage

The split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.

The optional input parameter limit is used to signify the number of elements into which the string should be divided, starting from the left end of the string and working rightward. In cases where the pattern is an alphabetical character, split() is case sensitive.

Return Value

- Returns an array of strings after splitting up a string.

Example

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

```
<?php

$ip = "123.456.789.000"; // some IP address
$iparr = split ("\.", $ip);

print "$iparr[0] <br />";
print "$iparr[1] <br />" ;
print "$iparr[2] <br />" ;
print "$iparr[3] <br />" ;

?>
```

This will produce the following result –

```
123
456
789
000
```

PHP — Function spliti()

Syntax

```
array spliti (string pattern, string string [, int limit])
```

Definition and Usage

The spliti() function operates exactly in the same manner as its sibling split(), except that it is not case sensitive. Case-sensitive characters are an issue only when the pattern is alphabetical. For all other characters, spliti() operates exactly as split() does.

Return Value

- Returns an array of strings after splitting up a string.

Example

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

```
<?php
    $ip = "123.456.789.000"; // some IP address
    $iparr = spliti ("\\.", $ip, 3);

    print "$iparr[0] <br />";
    print "$iparr[1] <br />" ;
    print "$iparr[2] <br />" ;
    print "$iparr[3] <br />" ;

?>
```

This will produce only three strings as we have limited number of strings to be produced.

```
123
456
789.000
```

PHP — Function sql_regcase()

Syntax

```
string sql_regcase (string string)
```

Definition and Usage

The sql_regcase() function can be thought of as a utility function, converting each character in the input parameter string into a bracketed expression containing two characters.

If the alphabetical character has both an uppercase and a lowercase format, the bracket will contain both forms; otherwise the original character will be repeated twice.

Return Value

- Returns a string of bracketed expression along with the converted character.

Example

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

```
<?php
    $version = "php 4.0";

    print sql_regcase($version);
?>
```

This will produce the following result –

```
[Pp][Hh][Pp] 4.0
```

PERL Style Regular Expressions

Perl-style regular expressions are similar to their POSIX counterparts. The POSIX syntax can be used almost interchangeably with the Perl-style regular expression functions. In fact, you can use any of the quantifiers introduced in the previous POSIX section.

Let us discuss a few concepts being used in PERL regular expressions. After that, we will introduce you with regular expression related functions.

Metacharacters

A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

For instance, you can search for large money sums using the '\d' metacharacter: `/([\d]+)000/`, Here `\d` will search for any string of numerical character.

Following is the list of metacharacters which can be used in PERL Style Regular Expressions.

Character	Description
.	a single character
\s	a whitespace character (space, tab, newline)
\S	non-whitespace character
\d	a digit (0-9)
\D	a non-digit
\w	a word character (a-z, A-Z, 0-9, _)
\W	a non-word character
[aeiou]	matches a single character in the given set
[^aeiou]	matches a single character outside the given set
(foo bar baz)	matches any of the alternatives specified

Modifiers

Several modifiers are available that can make your work with regexps much easier, like case sensitivity, searching in multiple lines etc.

Modifier	Description
i	Makes the match case insensitive
m	Specifies that if the string has newline or carriage return characters, the ^ and \$ operators will now match against a newline boundary, instead of a string boundary
o	Evaluates the expression only once
s	Allows use of . to match a newline character
x	Allows you to use white space in the expression for clarity
g	Globally finds all matches
cg	Allows a search to continue even after a global match fails

PHP's Regexp PERL Compatible Functions

PHP offers the following functions for searching strings using Perl-compatible regular expressions:

Function	Description
<code>preg_match()</code>	The <code>preg_match()</code> function searches string for pattern, returning true if pattern exists, and false otherwise.
<code>preg_match_all()</code>	The <code>preg_match_all()</code> function matches all occurrences of pattern in string.
<code>preg_replace()</code>	The <code>preg_replace()</code> function operates just like <code>ereg_replace()</code> , except that regular expressions can be used in the pattern and replacement input parameters.
<code>preg_split()</code>	The <code>preg_split()</code> function operates exactly like <code>split()</code> , except that regular expressions are accepted as input parameters for pattern.
<code>preg_grep()</code>	The <code>preg_grep()</code> function searches all elements of <code>input_array</code> , returning all elements matching the regexp pattern.
<code>preg_quote()</code>	Quote regular expression characters

PHP — Function `preg_match()`

Syntax

```
int preg_match (string pattern, string string [, array pattern_array], [, int $flags [, int $offset]]);
```

Definition and Usage

The `preg_match()` function searches string for pattern, returning true if pattern exists, and false otherwise.

If the optional input parameter `pattern_array` is provided, then `pattern_array` will contain various sections of the subpatterns contained in the search pattern, if applicable.

If this flag is passed as `PREG_OFFSET_CAPTURE`, for every occurring match the appendant string offset will also be returned

Normally, the search starts from the beginning of the subject string. The optional parameter `offset` can be used to specify the alternate place from which to start the search.

Return Value

- Returns true if pattern exists, and false otherwise.

Example

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

```
<?php
    $line = "Vi is the greatest word processor ever created!";
    // perform a case-Insensitive search for the word "Vi"

    if (preg_match("/\bVi\b/i", $line, $match)) :
        print "Match found!";
    endif;
?>
```

This will produce the following result –

Match found!

PHP — Function preg_match_all()

Syntax

```
int preg_match_all (string pattern, string string, array pattern_array [, int
order]);
```

Definition and Usage

The preg_match_all() function matches all occurrences of pattern in string.

It will place these matches in the array pattern_array in the order you specify using the optional input parameter order. There are two possible types of order –

- **PREG_PATTERN_ORDER** –REG_PATTERN_ORDERe matches in the array pattern_array in the od. PREG_PATTERN_ORDER specifies the order in the way that you might think most logical; \$pattern_array[0] is an array of all complete pattern matches, \$pattern_array[1] is an array of all strings matching the first parenthesized regexp, and so on.
- **PREG_SET_ORDER** –REG_SET_ORDERRe matches in the array pattern_array in the od. PREG_PATTERN_ORDER specifies the order in the way

that you might parenthesize a regexp, `$pattern_array[1]` will contain elements matched by the second parenthesized regexp, and so on.

Return Value

- Returns the number of matchings.

Example

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

```
<?php
    $userinfo = "Name: <b>John Poul</b> <br> Title: <b>PHP Guru</b>";
    preg_match_all ("/<b>(.*?)</b>/U", $userinfo, $pat_array);

    print $pat_array[0][0].<br> ".$pat_array[0][1].<br>";
?>
```

This will produce the following result –

```
John Poul
PHP Guru
```

PHP — Function preg_replace()

Syntax

```
mixed preg_replace (mixed pattern, mixed replacement, mixed string [, int limit  
[, int &$count]] );
```

Definition and Usage

The `preg_replace()` function operates just like POSIX function `ereg_replace()`, except that regular expressions can be used in the pattern and replacement input parameters.

The optional input parameter `limit` specifies how many matches should take place.

If the optional parameter `$count` is passed, then this variable will be filled with the number of replacements done.

Return Value

- After the replacement has occurred, the modified string will be returned.
- If no matches are found, the string will remain unchanged.

Example

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

```
<?php
    $copy_date = "Copyright 1999";
    $copy_date = preg_replace("([0-9]+)", "2000", $copy_date);

    print $copy_date;
?>
```

This will produce the following result –

Copyright 2000

PHP — Function preg_split()

Syntax

```
array preg_split (string pattern, string string [, int limit [, int flags]]);
```

Definition and Usage

The preg_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern.

If the optional input parameter limit is specified, then only limit number of substrings are returned.

flags can be any combination of the following types –

- **PREG_SPLIT_NO_EMPTY** –REG_SPLIT_NO_EMPTYmbination of the following fied, then only limit number of s
- **PREG_SPLIT_DELIM_CAPTURE** –REG_SPLIT_DELIM_CAPTUREtion of the following fied, then only limit number of substrings are returned.as input p
- **PREG_SPLIT_OFFSET_CAPTURE** –REG_SPLIT_Oag is set, for every occurring match the appendant string offset will also be returned.

Return Value

- Returns an array of strings after splitting up a string.

Example

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

```
<?php
    $ip = "123.456.789.000"; // some IP address
    $iparr = split ("/\\./", $ip);

    print "$iparr[0] <br />";
    print "$iparr[1] <br />" ;
    print "$iparr[2] <br />" ;
    print "$iparr[3] <br />" ;

?>
```

This will produce the following result –

```
123.456.789.000
```

PHP – Function preg_grep()

Syntax

```
array preg_grep ( string $pattern, array $input [, int $flags] );
```

Definition and Usage

Returns the array consisting of the elements of the input array that match the given pattern.

If flag is set to PREG_GREP_INVERT, this function returns the elements of the input array that do not match the given pattern.

Return Value

- Returns an array indexed using the keys from the input array.

Example

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

```
<?php
    $foods = array("pasta", "steak", "fish", "potatoes");

    // find elements beginning with "p", followed by one or more letters.
    $p_foods = preg_grep("/p(\w+)/", $foods);

    print "Found food is " . $p_foods[0];
    print "Found food is " . $p_foods[1];
?>
```

This will produce the following result –

```
Found food is pastaFound food is
```

PHP — Function preg_quote()

Syntax

```
string preg_quote ( string $str [, string $delimiter] );
```

Definition and Usage

preg_quote() takes str and puts a backslash in front of every character that is part of the regular expression syntax.

Return Value

- Returns the quoted string.

Example

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

```
<?php
    $keywords = '$40 for a g3/400';
    $keywords = preg_quote($keywords, '/');

    echo $keywords;
?>
```

This will produce the following result –

```
\$40 for a g3\400
```


23. PHP – Error and Exception Handling

Error handling is the process of catching errors raised by your program and then taking appropriate action. If you would handle errors properly, then it may lead to many unforeseen consequences. It is very simple in PHP to handle errors.

Using die() function

While writing your PHP program you should check all possible error condition before going ahead and take appropriate action when required.

Try the following example without having **/tmp/test.txt** file and with this file.

```
<?php
if(!file_exists("/tmp/test.txt"))
{
    die("File not found");
}
else
{
    $file=fopen("/tmp/test.txt","r");
    print "Opend file sucessfully";
}
// Test of the code here.
?>
```

You can thus write an efficient code. Using the above technique, you can stop your program whenever it errors out and display more meaningful and user-friendly message.

Defining 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 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

Possible Error levels

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

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

2048	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.
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, except level E_STRICT (E_STRICT will be part of E_ALL as of PHP 6.0)

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

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

Here is how you can create an error handling function:

```
<?php
function handleError($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 let's examine our example by calling a function which does not exist.

```
<?php
error_reporting( E_ERROR );
function handleError($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.

Let's now explain the new keyword related to exceptions.

- **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, the 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 ("caught") 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

Copy and paste the following piece of 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. The following functions 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()** - formatted string of trace

Creating Custom Exception Handler

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

```
string set_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
function exception_handler($exception) {
    echo "Uncaught exception: " , $exception->getMessage(), "\n";
}

set_exception_handler('exception_handler');

throw new Exception('Uncaught Exception');

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

Check the complete set of error handling functions at [PHP Error Handling Functions](#).

24. PHP – Bugs Debugging

Programs rarely work correctly the first time. Many things can go wrong in your program that cause the PHP interpreter to generate an error message. You have a choice about where those error messages go. The messages can be sent along with other program output to the web browser. They can also be included in the web server error log.

To make error messages display in the browser, set the **display_errors** configuration directive to **On**. To send errors to the web server error log, set **log_errors** to **On**. You can set them both to **On** if you want error messages in both places.

PHP defines some constants you can use to set the value of **error_reporting** such that only errors of certain types get reported: **E_ALL** (for all errors except strict notices), **E_PARSE** (parse errors), **E_ERROR** (fatal errors), **E_WARNING** (warnings), **E_NOTICE** (notices), and **E_STRICT** (strict notices).

While writing your PHP program, it is a good idea to use PHP-aware editors like **BBEdit** or **Emacs**. One of the special features of these editors is syntax highlighting. It changes the color of different parts of your program based on what those parts are. For example, strings are pink, keywords such as **if** and **while** are blue, comments are grey, and variables are black.

Another feature is quote and bracket matching, which helps to make sure that your quotes and brackets are balanced. When you type a closing delimiter such as **}**, the editor highlights the opening **{** that it matches.

You need to verify the following points while debugging your program.

- **Missing Semicolons** - Every PHP statement ends with a semicolon (**;**). PHP doesn't stop reading a statement until it reaches a semicolon. If you leave out the semicolon at the end of a line, PHP continues reading the statement on the following line.
- **Not Enough Equal Signs** - When you ask whether two values are equal in a comparison statement, you need two equal signs (**==**). Using one equal sign is a common mistake.
- **Misspelled Variable Names** - If you misspelled a variable, then PHP understands it as a new variable. Remember: To PHP, **\$test** is not the same variable as **\$Test**.
- **Missing Dollar Signs** - A missing dollar sign in a variable name is really hard to see, but at least it usually results in an error message so that you know where to look for the problem.
- **Troubling Quotes** - You can have too many, too few, or the wrong kind of quotes. So check for a balanced number of quotes.
- **Missing Parentheses and curly brackets** - They should always be in pairs.
- **Array Index** - All the arrays should start from zero instead of 1.

Moreover, handle all the errors properly and direct all trace messages into system log file so that if any problem occurs, then it will be logged into system log file and you will be able to debug that problem.

25. PHP – Date and Time

Dates are so much part of everyday life that it becomes easy to work with them without thinking. PHP also provides powerful tools for date arithmetic that make manipulating dates easy.

Getting the Time Stamp with `time()`

PHP's **`time()`** function gives you all the information that you need about the current date and time. It requires no arguments but returns an integer.

The integer returned by `time()` represents the number of seconds elapsed since midnight GMT on January 1, 1970. This moment is known as the UNIX epoch, and the number of seconds that have elapsed since then is referred to as a time stamp.

```
<?php
print time();
?>
```

It will produce the following result:

```
1468926906
```

This is something difficult to understand. But PHP offers excellent tools to convert a time stamp into a form that humans are comfortable with.

Converting a Time Stamp with `getdate()`

The function **`getdate()`** optionally accepts a timestamp and returns an associative array containing information about the date. If you omit the time stamp, it works with the current time stamp as returned by `time()`.

The following table lists the elements contained in the array returned by `getdate()`.

Key	Description	Example
seconds	Seconds past the minutes (0-59)	20
minutes	Minutes past the hour (0 - 59)	29
hours	Hours of the day (0 - 23)	22
mday	Day of the month (1 - 31)	11
wday	Day of the week (0 - 6)	4
mon	Month of the year (1 - 12)	7
year	Year (4 digits)	1997
yday	Day of year (0 - 365)	19
weekday	Day of the week	Thursday
month	Month of the year	January
0	Timestamp	948370048

Now you have complete control over date and time. You can format this date and time in whatever format you want.

Example

Try out the following example.

```
<?php
$date_array = getdate();
foreach ( $date_array as $key => $val )
{
    print "$key = $val<br />";
}
$formated_date = "Today's date: ";
$formated_date .= $date_array[mday] . "/";
$formated_date .= $date_array[mon] . "/";
$formated_date .= $date_array[year];

print $formated_date;
?>
```

It will produce the following result:

```
seconds = 6
minutes = 15
hours = 11
mday = 19
wday = 2
mon = 7
year = 2016
yday = 200
weekday = Tuesday
month = July
O = 1468926906
Today's date: 19/7/2016
```

Converting a Time Stamp with date()

The **date()** function returns a formatted string representing a date. You can exercise an enormous amount of control over the format that date() returns with a string argument that you must pass to it.

```
date(format,timestamp)
```

The date() optionally accepts a time stamp if omitted, then current date and time will be used. Any other data you include in the format string passed to date() will be included in the return value.

The following table lists the codes that a format string can contain:

Format	Description	Example
a	'am' or 'pm' lowercase	pm
A	'AM' or 'PM' uppercase	PM
d	Day of month, a number with leading zeroes	20
D	Day of week (three letters)	Thu
F	Month name	January
h	Hour (12-hour format - leading zeroes)	12
H	Hour (24-hour format - leading zeroes)	22
g	Hour (12-hour format - no leading zeroes)	12
G	Hour (24-hour format - no leading zeroes)	22
i	Minutes (0 - 59)	23

j	Day of the month (no leading zeroes)	20
l (Lower 'L')	Day of the week	Thursday
L	Leap year ('1' for yes, '0' for no)	1
m	Month of year (number - leading zeroes)	1
M	Month of year (three letters)	Jan
r	The RFC 2822 formatted date	Thu, 21 Dec 2000 16:01:07 +0200
n	Month of year (number - no leading zeroes)	2
s	Seconds of hour	20
U	Time stamp	948372444
y	Year (two digits)	06
Y	Year (four digits)	2006
z	Day of year (0 - 365)	206
Z	Offset in seconds from GMT	+5

Example

Try out the following example.

```
<?php
print date("m/d/y G.i:s<br>", time());
print "Today is ";
print date("j of F Y, \a\\t g.i a", time());
?>
```

It will produce following result:

```
07/19/16 11.15:06Today is 19 2016f July 2016, at 11.15 am
```

Hope you have a good understanding of how to format date and time according to your requirement. For your reference a complete list of all the date and time functions is given in [PHP Date & Time Functions](#).