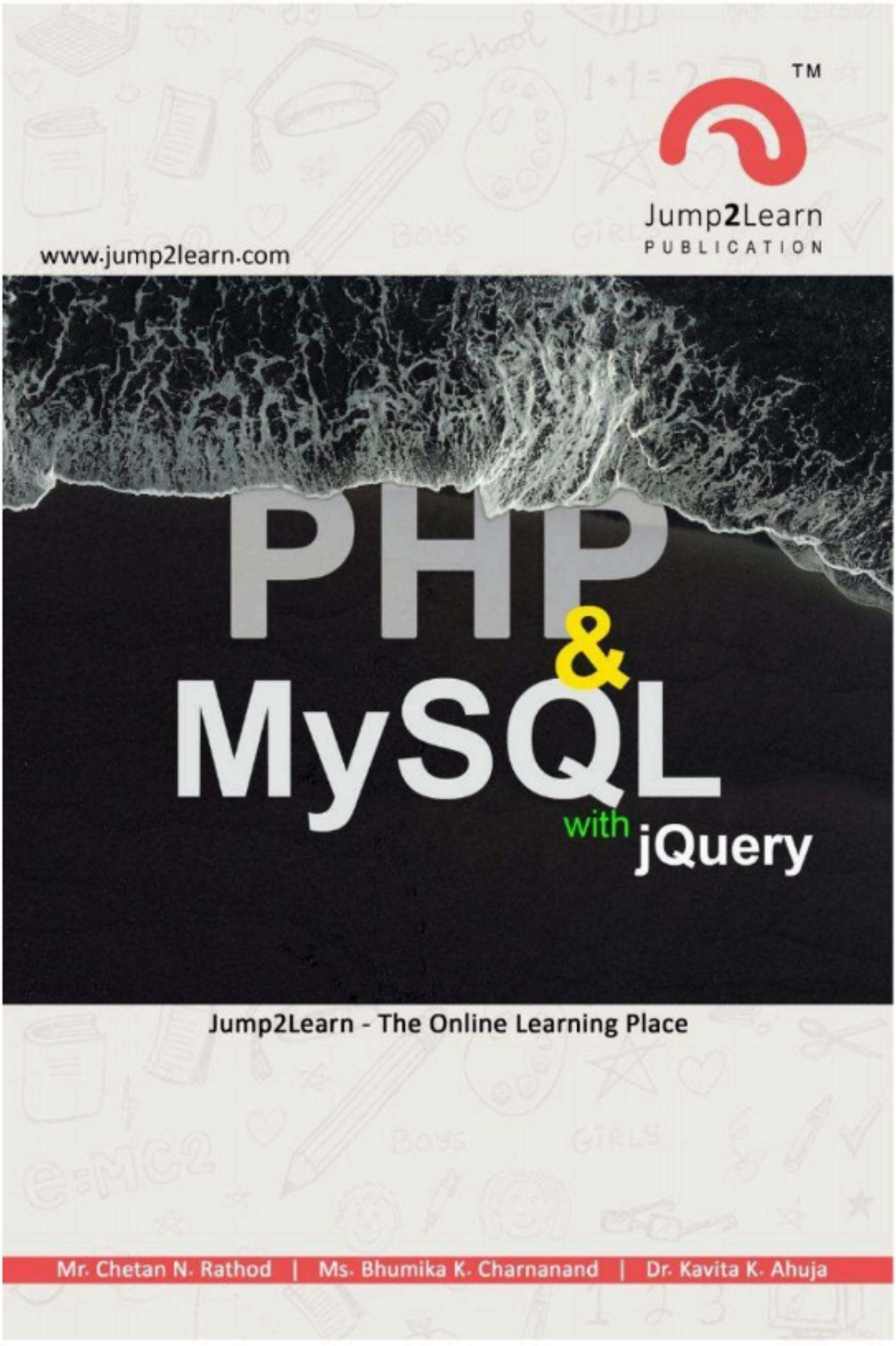

Jump2Learn
PUBLICATION
www.jump2learn.com
PHP
&
MySQL
with jQuery
Jump2Learn - The Online Learning Place
Mr. Chetan N. Rathod | Ms. Bhumika K. Charnanand | Dr. Kavita K. Ahuja

Unit - 2

Working with Data and Functions

FORM element, INPUT elements
Validating the user Input
Passing variables between pages through GET, POST and REQUEST
Built-in Functions
String Functions: chr, ord, strtolower, strtoupper, strlen, ltrim, rtrim, substr, strcmp, strcasecmp, strpos, strrpos, strstr, stristr, str_replace, strrev, echo, print
Math Functions: abs, ceil, floor, round, fmod, min, max, pow, sqrt, rand
Array Functions: count, list, in_array, current, next, previous, end, each, sort, rsort, assort, array_merge, array_reverse
Date Functions: date, getdate, DateTime::setDate, checkdate, time, mktime
File System Functions
Miscellaneous Functions
User-defined Functions

FORM

The Form element is used to create form area on the page. The form element is the container that defines the start and end points for a form that a site visitor can fill in. Any fields that are located between the opening <form> and closing </form> tags will be associated with this form.

It is possible to have multiple form elements on a page. It holds the fields you want in your web site for users to fill in. Form element contains all input element and other element like label, textbox, check box, submit button, etc.

Information can be captured in the form using many different form-specific elements. These controls include the input element, as well as textarea, select, and button. There are also labeling and grouping controls, which include the fieldset, legend, and label elements. Form has its own attributes like **action, method, accept-charset, enctype, name and target.**

FORM ELEMENT, INPUT ELEMENTS

ATTRIBUTES OF THE FORMS

action: It is a page recipient address to when the value of variable is sent.

method: It specify the name of the method (POST, GET, REQUEST) for passing the value. By it we can send data to the recipient.

accept-charset: It specifies the character encodings the server must accept.

enctype: It defines the way form data should be encoded when sent.

name: It specifies a name by which the form will be referenced.

target: It defines the frame (in frameset-based designs) that will process and display form submission results.

Syntax:

```
<form action="uri" enctype=" { application/x-www-form-urlencoded | multipart /form-data | text/plain } " method=" { get | post } ">
.....
.....
.....
</form>
```

Example:

```
<form action="form-to-email.php" method="post">
        <div>
                <label for="txtname">Name:</label>
                <input type="text" name="txtname" id="txtname"/>
        </div>
        <div>
                <label for="txtcontacttel">Contact Tel:</label>
                <input type="text" name="txtcontacttel" id="txtcontacttel"/>
        </div>
        <div>
                <input type="submit" name="cmdSubmit" id="cmdSubmit"
value="Send booking details"/>
        </div>
</form>
```

TEXT FIELD:

The input text type is standard single line text box. Allow the user to input a text.

Syntax:

```
< input type=”text” name=”name” / >
```

The PHP $_GET and $_POST variables are used to retrieve information from the Text Box of forms.

Example:

For example, suppose the form contains a text-field value named email that looks like this:

```
<input type="text" id="email" name="email" />
```

Once this form is submitted, you can reference that text-field value like so:

```
$_POST['email']
```

Note: If you want several lines you should use **textarea.**

PASSWORD FIELD:

Password fields are similar to text fields. The difference is that what is inserted into a password field shows up as dots on the screen. This prevents others from reading the password on the screen.

Syntax:

< input type="password" name="name" / >

Example:

< input type="password" name="UserPwd" / >

Once this form is submitted, you can reference that password-field value like so:

$_POST['UserPwd']

TEXT AREA:

Text areas are text fields that can span several lines. Unlike most other form fields, text areas are not defined with an <input> tag.

Instead you enter a <textarea> tag where you want the text area to start and a closing </textarea> tag where you want the area to end. Everything written between these tags will be presented in the text area box.

Syntax:

```
<textarea cols="columns" rows ="rows" name="name">
    TEXT.....
</textarea>
```


Example:

```
<textarea cols="40" rows ="5" name="MyTxtArea">
    We are inside the text area.....
</textarea>
```

Once this form is submitted, you can reference that text-area value like so:

$_POST['MyTxtArea']

HIDDEN FIELD:

Hidden fields are similar to text fields, with one very important difference. The difference is that the hidden field does not show on the page. Therefore the visitor can't type anything into a hidden field, which leads the purpose of the field:

To submit information that is not entered by the user.

Syntax:

```
< input type="hidden" name="name" value="value" / >
```

Example:

```
< input type="hidden" name="Language" value="English" / >
```

LIST BOX:

List Box is used for selection purpose. List Boxes are used when you want to let the visitor select one or more options from a set of alternatives.

Syntax:

```
<select name="name" multiple="multiple" id="id">
        <option value="value"> Text </option>
        <option value=" value"> Text </option>
        <option value=" value"> Text </option>
        <option value=" value "> Text </option>
</select>
```

Example:

For example, suppose the form contains a Multi Selection List Box which contain a list of Names that looks like this:

```
<select name="select[]" multiple="multiple" id="select[]">
        <option value="1">Chetan</option>
        <option value="2">Badal</option>
        <option value="3">Gaurang</option>
        <option value="4">Sandip</option>
</select>
```

Once this form is submitted, you can reference the value of list box like so:

```
$list = $_POST['select'];
        foreach ($list as $a)
        {
                echo $a;
        }
```

CHECK BOX:

Check Boxes are used when you want to let the visitor select one or more options from a set of alternatives.

Syntax:

```
<input type="checkbox" name="name" value="value" /> Text
```

Example:

```
<input type="checkbox" name="chk[]" value="A" /> BCA <br/>
<input type="checkbox" name="chk[]" value="B" /> BBA<br/>
<input type="checkbox" name="chk[]" value="C" /> BCom <br/>
```

Once this form is submitted, you can reference the value of check box like so:

```
$check = $_POST['chk'];
        foreach ($check as $b)
        {
                echo $b;
        }
```


RADIO BUTTON:

Radio Buttons are used when you want to let the visitor select just one option from a set of alternatives.

Syntax:

```
<input type="radio" name="name" value="value" checked="checked" />Text
```

Example:

```
<input type="radio" name="rdSex" value="Male" checked="checked" />Male
<input type="radio" name="rdSex" value="Female" /> Female
```

Once this form is submitted, you can reference the value of Radio Button like so:

```
echo "The Selected Gender is ". $_REQUEST['rdSex'];
```

SUBMIT BUTTON:

When a visitor clicks a submit button, the form is sent to the address specified in the action setting of the <form> tag.

Syntax:

```
<input type="submit" name="name" value="Submit" />
```

Example:

```
<input type="submit" name="btnSubmit" value="Submit" />
```

RESET BUTTON:

When a visitor clicks a Reset Button, all controls (form or input elements) are reset to the default value.

Syntax:

```
<input type="reset" name="name" value="Reset" />
```

Example:

```
<input type="reset" name="BtnReset" value="Reset!" />
```

IMAGE BUTTON:

Image buttons have the same effect as the submit buttons. When a visitor clicks a submit button, the form is sent to the address specified in the action setting of the <form> tag. Only difference is that we can put any image(picture) instead of normal button.

Syntax:

<input type="image" src="url" name="name">

Example:

<input type="image" src="rainbow.gif" name="BtnImage" width="60" height="60">

Label:

Specifies descriptive text for the associated form control.

Syntax:

<label for="Control_ID">Text</label>

Example:

```
<input type="text" id="TxtName" name="Name" />
<label for="TxtName">Name:</label>
```

LEGEND:

Defines caption text for form controls that are grouped by the fieldset element.

Syntax:

<legend>Group Name</legend>

Example:

<legend>Faculty</legend>

For example you are checking student roll no is integer or not if not then convert it to an integer. And you also check that your roll no is set or not?

For this you have to write a code like:

```
<?php
        if (!isset($_GET['roll_no']))
        {
                die ("Error, Roll No was not set");
        }
        $Roll_No = (int) $_GET['roll_no'];
?>
```

Note: If roll no is not an integer then it is converted by 0.

If you want to force the user to enter marks in number and also marks is between 0 to 100 then write following script.

```
        if (!is_numeric($GET['mark']))
        {
                $error[]= 'Please enter a numeric value.';
        }
        else
                if (($GET['mark']) < 0 || ($GET['mark']) > 100)
                {
                        $error[]= 'Please enter marks between 0 to 100.';
                }
```

SOME VALIDATING FUNCTION ARE:

is_array(): Checks if the variable holds an array or not.

is_binary(): Checks if the variable holds a native binary string or not .

is_bool(): Checks for Boolean - type values (TRUE, FALSE, 0, or 1) .

is_callable(): Checks if the variable ' s value can be called as a function or not.

is_float(): Checks if the variable holds a decimal value or not .

is_int(): Checks if the variable holds an integer value or not.

is_null(): Checks if the variable ' s value is null or not.

For example you are checking student roll no is integer or not if not then convert it to an integer. And you also check that your roll no is set or not?

For this you have to write a code like:

```
<?php
        if (!isset($_GET['roll_no']))
        {
                die ("Error, Roll No was not set");
        }
        $Roll_No = (int) $_GET['roll_no'];
?>
```

Note: If roll no is not an integer then it is converted by 0.

If you want to force the user to enter marks in number and also marks is between 0 to 100 then write following script.

```
        if (!is_numeric($GET['mark']))
        {
                $error[]= 'Please enter a numeric value.';
        }
        else
                if (($GET['mark']) < 0 || ($GET['mark']) > 100)
                {
                        $error[]= 'Please enter marks between 0 to 100.';
                }
```

SOME VALIDATING FUNCTION ARE:

is_array(): Checks if the variable holds an array or not.

is_binary(): Checks if the variable holds a native binary string or not .

is_bool(): Checks for Boolean - type values (TRUE, FALSE, 0, or 1) .

is_callable(): Checks if the variable ' s value can be called as a function or not.

is_float(): Checks if the variable holds a decimal value or not .

is_int(): Checks if the variable holds an integer value or not.

is_null(): Checks if the variable ' s value is null or not.

is_numeric(): Checks if the variable holds a number or not .

is_object(): Checks if the variable stores an object or not .

is_resource(): Checks to see if the variable is a resource or not.

is_string(): Checks to see if the value is a string or not.

is_unicode(): Checks to see if the value is a Unicode string or not.

PASSING VARIABLES BETWEEN PAGES THROUGH GET, POST AND REQUEST (Form Handling)

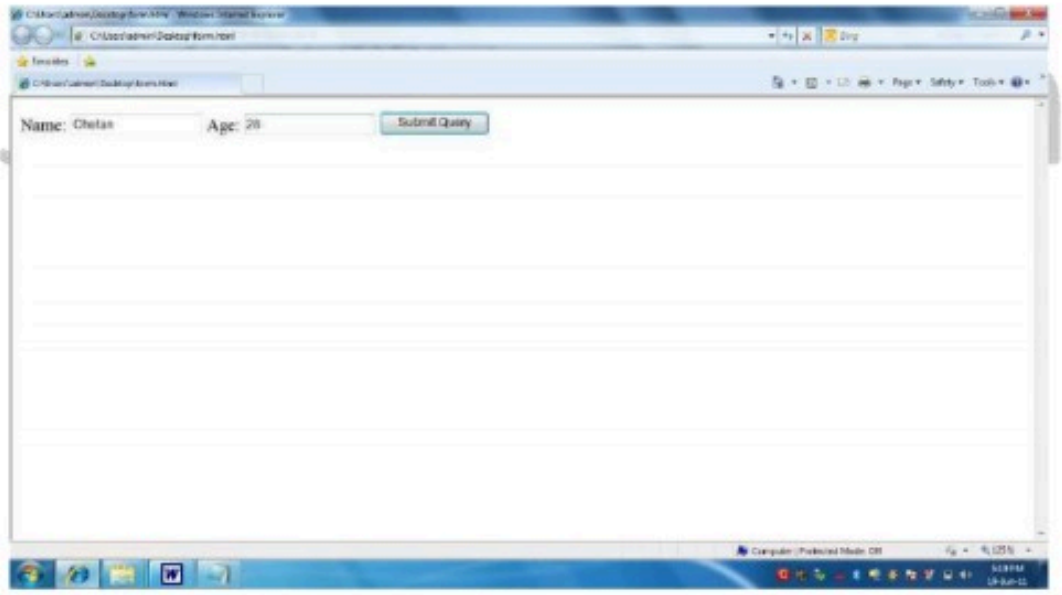
The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will automatically be available to your PHP scripts.

Example:

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

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

Output:



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

"welcome.php" looks like this:


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

Output:

Welcome Chetan!
You are 28 years old.

THE $_GET FUNCTION

The $_GET function is used to passing variable to another page. The built-in $_GET function is used to collect values from a form sent with method="get".

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

Example:


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

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

http://localhost:8080/welcome.php?fname=Chetan&age=28

The "welcome.php" file can now use the $_GET function to collect form data.

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

Output:

Welcome Chetan!
You are 28 years old.

WHEN TO USE METHOD="GET"?

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

Note: This method should not be used when sending passwords or other sensitive information!
However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

Note: The get method is not suitable for very large variable values. It should not be used with values exceeding 2000 characters.

THE $_POST FUNCTION

The $_POST function is used to passing variable to another page. The built-in $_POST function is used to collect values from a form sent with method="post".

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

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

Example:

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

When the user clicks the "Submit" button, the URL will look like this:

http://localhost:8080/welcome.php

The "welcome.php" file can now use the $_POST function to collect form data.

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

Output:

Welcome Chetan!
You are 28 years old.

WHEN TO USE METHOD ="POST"?

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

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

DIFFERENCE BETWEEN $_POST AND $_GET METHOD:

Information sent from a form with the POST method is invisible to others. While Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar.)

POST has no limits on the amount of information to send. GET has limits on the amount of information to send.(It should not be used with values exceeding 2000 characters.)

It is not possible to bookmark the page in POST method because the variables are not displayed in the URL. It is possible to bookmark the page in GET method because the variables are displayed in the URL.

THE $_REQUEST FUNCTION

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

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

Example:

```
Welcome <?php echo $_REQUEST["fname"]; ?>!<br />
```

BUILT-IN FUNCTIONS

PHP has wide range of built in function which are use for their convenient use to make programs.

1. **String Functions**
2. **Math Functions**
3. **Date Functions**
4. **Array Functions**
5. **File Handling Functions**
6. **Miscellaneous functions**

String Functions

The string functions allow you to manipulate strings.

chr()

The chr() function returns a character from the **specified ASCII value**. You have to pass ASCII value (as an integer) as a parameter; it means it required an ASCII value. It will return a one-character string containing the character specified by ASCII.

Syntax:

string **chr** (int $ascii)

Example:

```
<html>
<head>
    <title>USE OF CHR()</title>
</head>
<body>
    <?php
        echo (chr(34))."<br />";   // Prints a quotation mark "
        echo chr(65)."<br />";     //it will print "A"
        echo chr(105)."<br />";    //it will print "i"
    ?>
</body>
</html>
```

ord()

The ord() function returns the **ASCII value of the first character of a string**. ord() is chr()'s complement. It receives a character and returns the corresponding ASCII code as an **integer.**

Syntax:

int **ord** (string $string)

Example:

```
<html>
<head>
	<title>USE OF ORD()</title>
</head>
<body>
	<?php
		$str = "\n";	// here ASCII value of "\n" is 10
		if (ord($str) == 10)
		{
			echo "The first character of \$str is a line feed.\n";
		}
		else
		{
			echo "The first character of \$str is not a line feed.\n";
		}
	?>
</body>
</html>
```


Example:

```
<html>
<head>
	<title>USE OF ORD()</title>
</head>
<body>
```

```
<?php
        echo ord("h")."<br />"; // it will print 104
        echo ord("hello")."<br />";    // it will print 104
        //because ord() will return the ASCII value of only 1st character from
the string
?>
</body>
</html>
```


strtolower()

The strtolower() function converts a string to **all lowercase letters**, returning the modified string.

Note: Non-alphabetical characters are not affected.

Syntax:

string **strtolower**(string $str)

Example:

```
<html>
<head>
    <title>Use Of Strtolower()</title>
</head>
<body>
    <?php
        $str = "Hello HOW Are YOU !!!";
        echo strtolower($str);
    ?>
</body>
</html>
```

Output:

hello how are you !!!

strtoupper()

The strtoupper() function converts a string to **uppercase**, returning the modified string.

Syntax:

string **strtoupper**(string $str)

Example:

```
<html>
<head>
	<title>USE OF STRTOUPPER()</title>
</head>
<body>
	<?php
		$str = "HELLO world";
		echo strtoupper($str);
	?>
</body>
</html>
```

Output:

HELLO WORLD

strlen()

This function returns the **length of given string**, where each character in the string is equivalent to one unit.

Syntax:

int **strlen**(string $str)

Note: It return the length of the *string* on success, and *0* if the *string* is empty.

Example:

```
<html>
<head>
	<title>USE OF STRLEN()</title>
</head>
<body>
	<?php
		$num = "12345678";
```

```
        if (strlen($num) < 10)
                echo "Mobile number is too short!";
        else
                echo "Mobile number is valid!";
    ?>
</body>
</html>
```

Example:

```
<html>
<head>
    <title>USE OF STRLEN()</title>
</head>
<body>
    <?php
        $str = 'HELLO';
        echo strlen($str)."<br />"; // 5
        $str = 'HOW ARE YOU';   // space will also be count
        echo strlen($str); // 11
    ?>
</body>
</html>
```


ltrim()

The ltrim() function will remove white spaces or other predefined character from the **left side of a string.** It means it removes white spaces or other predefined characters from the beginning of a string.

Syntax:

string **ltrim** (string $str [, string $charlist])

Parameters:

str - The input string.

charlist

It is optional. Specifies which characters to **remove from the string**. If omitted, all of the following characters are removed:

- "\0" – NULL (ASCII 0)
- "\t" - tab (ASCII 9)
- "\n" - new line (ASCII 10)
- "\x0B" - vertical tab (ASCII 11)
- "\r" - carriage return (ASCII 13)
- " " - ordinary white space (ASCII 32)

Example:

```
<html>
<head>
        <title>USE OF LTRIM()</title>
</head>
<body>
        <?php
                $str = "                        Hello World!";
                echo "Without ltrim: " . $str;
                echo "<br />";
                echo "With ltrim: " . ltrim($str);
        ?>
</body>
</html>
```

Output:

```
Without ltrim:                  Hello World!
With ltrim: Hello World!
```

Example:

```
<html>
<head>
        <title>USE OF LTRIM()</title>
</head>
<body>
        <?php
                $str = "        Hi How Are You";
                echo "Without ltrim: " . $str;
                echo "<br />";
```

```
        echo "With ltrim: " . ltrim($str,"\t" );
    ?>
</body>
</html>
```

Output:

```
Without ltrim:          Hi How Are You
With ltrim : Hi How Are You
```

rtrim()

The rtrim() function operates just like to ltrim(), but the difference is that it removes the designated characters from the **right side of a string**. It means it removes white spaces or other predefined characters from the End of a string.

Syntax:

string **rtrim** (string $str [, string $charlist])

Parameters:

str - The input string.

charlist-

It is optional. Specifies which characters to remove from the string. If omitted, all of the following characters are removed:

- "\0" – NULL(ASCII 0)
- "\t" - tab(ASCII 9)
- "\n" - new line(ASCII 10)
- "\x0B" - vertical tab(ASCII 11)
- "\r" - carriage return(ASCII 13)
- " " - ordinary white space(ASCII 32)

In short, The rtrim() function will remove white spaces or other predefined character from the right side of a string.

Example:

```
<html>
<head>
```

```
        <title>USE OF RTRIM()</title>
</head>
<body>
        <?php
                $str = "Hello World!            ";
                echo "Without rtrim: " . $str ."!";
                echo "<br />";
                echo "With rtrim: " . rtrim($str) . "!";
        ?>
</body>
</html>
```

Output:

```
Without rtrim: Hello World!            !
With rtrim: Hello World!!
```


substr()

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

Syntax:

string **substr** (string *source*, int *begin*, int [*length*]);

PARAMETER	DESCRIPTION
String	Specifies the string to return a part of the input string. String must be one character or longer.
Start	Specifies where to start in the string A positive number - Start at a specified position in the string A negative number - Start at a specified position from the end of the string 0 - Start at the first character in string
Length	It is optional. Specifies the length of the returned string. Default is to the end of the string. A positive number - The length to be returned from the start parameter

	Negative number - The length to be returned from the end of the string

Note: If start is a negative number and length is less than or equal to start, length becomes 0. Returns the extracted part of string, or FALSE on failure or an empty string.

echo (substr("Good Morning", 1));

will print ood Morning.

If the second argument is negative, substr() will count backwards from the end of the source string:

echo (substr("Good Morning", -2));

will print ng. This basically says "Return the piece of the string beginning at the second-to-last character".

The third argument is optional. It is an integer that specifies the length of the substring to be returned.

echo (substr("Good Morning", -5, 3)); // Prints 'rni'

Example:

```
<html>
<head>
        <title>USE OF SUBSTR()</title>
</head>
<body>
        <?php
                echo (substr("Good Morning\n", 1));   // returns "ood Morning"
                echo (substr("\nGood Morning", -2));// returns "ng"
                echo (substr("\nGood Morning", -5, 3)); // returns "rni"
        ?>
</body>
</html>
```


strcmp()

The strcmp() function takes two strings as arguments and returns 0 if the strings are **exactly equivalent**.

Syntax:

int **strcmp** (string $str1 , string $str2)

The strcmp() function compares two strings. This function returns:

- 0 - if the two strings are equal
- <0 - if string1 is less than string2
- >0 - if string1 is greater than string2

Note: The strcmp() function is binary safe and case-sensitive.

If strcmp() encounters a difference, it returns a negative number if the first byte is a smaller ASCII value in the first string, and a positive number if the smaller byte is found in the second string.

Note: The strcmp() function performs a binary-safe, case-sensitive comparison of two strings.

Syntax:

int **strcmp** (string $str1 , string $str2)

Example:

```
<html>
<head>
    <title>USE OF SUBSTR()</title>
</head>
<body>
    <?php
        echo strcmp("Good Morning","Good Morning");
    ?>
</body>
</html>
```


Output:

0

Example:

```
<html>
<head>
	<title>USE OF SUBSTR()</title>
</head>
<body>
	<?php
		$str1 = "Hello";
		$str2 = "HiHello";
		if (strcmp($str1, $str2) != 0)
		{
			echo "Both strings are not Same";
		}
		else
		{
			echo "Both strings Same";
		}
	?>
</body>
</html>
```

Output:

Both strings are not Same

Example:

```
<html>
<head>
	<title>USE OF SUBSTR()</title>
</head>
<body>
	<?php
		$str1 = "Hello";
		$str2 = "HiHello";
		echo strcmp(str1,str2);
	?>
</body>
</html>
```

Output:

-1

strcasecmp()

The strcasecmp() function **compares two strings**. This function returns:

- 0 - if the two strings are equal
- <0 - if string1 is less than string2
- >0 - if string1 is greater than string2

Syntax:

int **strcasecmp** (string $str1 , string $str2)

Note: The strcasecmp() function is binary safe and case-insensitive.

Example:

```
<html>
<body>
        <?php
                echo strcasecmp("Hello world!","HELLO WORLD!"); //0
        ?>
</body>
</html>
```

Output:

0

strops()

The strpos() function returns the **position of the first occurrence of a string2 in string1**. If the string is not found, this function returns FALSE.

Syntax:

int **strops** (string string1, string string2 [, int offset=0])

Here string 1 is a source string, string2 is to find, and third argument is optional where to begin the search.

Note: The strpos() function is case-sensitive.

Example:

```
<html>
<head>
        <title>USE OF STRPOS()</title>
</head>
<body>
        <?php
          echo strpos("Good Morning","or");   // 6
        echo strpos("Good Morning","G");   // 0
        echo strpos("Good Morning","o",5);   // 6  (here it start from 5th position)
        echo strpos("Good Morning","M");   //5
        ?>
</body>
</html>
```


strrpos:

The strrpos() function finds the **position of the last occurrence of a string2 inside string1** and returning its **numerical position**.

strrpos() accepts a string and a character and searches for the character, starting from the end of the string, forwards up to the start of the string, then returns the position of the first occurrence it finds.

strrpos() is a cousin of strpos() , which does the same, but strrpos() starts searching from the ending of the string. This function returns the position on success, otherwise it returns FALSE.

Syntax:

int **strrpos** (string *string1*, string *string2* [, int *offset=0*])

Here string 1 is a source string, string2 is to find, and third argument is optional where to begin the search.

Note: The strrpos() function is case-sensitive.

Example:

```
<html>
<head>
	<title>Use of Strrpos()</title>
</head>
<body>
	<?php
		echo strrpos("Good Morning","o");//6
	?>
</body>
```

strstr()

The strstr() function searches for the **first occurrence of a string inside another string**. This function returns the rest of the string (from the matching point), or FALSE, if the string to search for is not found.

Syntax:

string **strstr** (string str , mixed search [, bool before_search = false])

str: Specifies the string to search.

search: Specifies the string to search for. If this parameter is a number, it will search for the character matching the ASCII value of the number.

before_search: If TRUE, strstr() returns the part of the str before the first occurrence of the search.

Note: This function is binary safe and case-sensitive. For a case-insensitive search, use stristr().

Example:

```
<html>
<head>
	<title>Use of StrStr()</title>
</head>
<body>
	<?php
		echo strstr("GOOD MORNING","MOR"); //MORNING
```

```
            echo strstr("GOOD MORNING","MOR", true); //GOOD
        ?>
</body>
</html>
```

Example:

In this example we will search a string for the ASCII value of "d":

```
<html>
<head>
        <title>Use of StrStr()</title>
</head>
<body>
        <?php
                echo strstr("Good Morning!",100);// d Morning!
        ?>
</body>
</html>
```


stristr()

The stristr() function searches for the **first occurrence of a string inside another string**. This function returns the rest of the string (from the matching point), or FALSE, if the string to search for is not found.

Syntax:

string **stistr** (string str , mixed search [, bool before_search = false])

str: Specifies the string to search.

search: Specifies the string to search for. If this parameter is a number, it will search for the character matching the ASCII value of the number.

before_search: If TRUE, strstr() returns the part of the str before the first occurrence of the search.

Note: This function is case-insensitive. For a case-sensitive search, use strstr().

Example:

```
<html>
<head>
        <title>Use of StriStr()</title>
</head>
<body>
        <?php
                echo stristr("Good Morning!","MORNING");// Morning!
        ?>
</body>
</html>
```

str_replace()

The str_replace() function **replaces some characters with some other characters** in a string. This function works by the following rules:

- If the string to be searched is an array, it returns an array
- If the string to be searched is an array, find and replace is performed with every array element
- If both find and replace are arrays, and replace has fewer elements than find, an empty string will be used as replace
- If find is an array and replace is a string, the replace string will be used for every find value

Syntax:

str_replace (find,replace,string,count)

find: Required. Specifies the value to find.

replace: Required. Specifies the value to replace the value in find.

string: Required. Specifies the string to be searched.

count: Optional. A variable that counts the number of replacements.

Note: This function is case-sensitive. Use str_ireplace() to perform a case-insensitive search. This function is binary-safe.

Example:

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

Output:

```
Hello Peter
```

Example:

```
<?php
        $arr = array("blue","red","green","yellow");
        print_r(str_replace("red","pink",$arr,$i));
        echo "Replacements: $i";
?>
```

Output:

```
Array
(
[0] => blue
[1] => pink
[2] => green
[3] => yellow
)
Replacements: 1
```

Example:

```
<?php
        $find = array("Hello","world");
        $replace = array("B");
        $arr = array("Hello","world","!");
        print_r(str_replace($find,$replace,$arr));
?>
```


Output:

```
Array
(
[0] => B
[1] =>
[2] => !
)
```

strrev()

The strrev() function **reverses a string**.

Syntax:

string **strrev** (string $string)

string: Required. Specifies the string to reverse.

Example:

```
<?php
    echo strrev("Rathod Chetan");
?>
```

Output:

natehC dohtaR

Math Functions

These math functions will only handle values within the range of the integer and float types on your computer.

abs()

Returns the absolute value of **number Short for absolute value.**

- If the single numerical argument is negative, the corresponding positive number is returned.
- If the argument is positive, the argument itself is returned.

Syntax:

number **abs** (mixed $value)

Parameters:

$value - The numeric value to process

If the number is of type float, the return type is also float, otherwise it is integer.

Example:

```
<head>
    <title>USE OF abs</title>
</head>
<body>
    <?php
        echo(abs(-10.10) . "<br />"); // $abs = 10.1; (double/float)
        echo(abs(-10) . "<br />");  //$abs = 10; (integer)
        echo(abs(10));  // $abs2 = 10; (integer)
    ?>
</body>
</html>
```

Output:

```
10.1
10
10
```

ceil()

It returns the **next highest integer** by rounding the value upwards.

Short for ceiling—takes a single argument (typically a double) and returns the smallest integer that is greater than or equal to that argument.

The function ceil() takes a float and returns the integer greater than or equal to that float.

Syntax:

float **ceil** (float $value)

Parameters:

$value - The numeric value to round

Example:

```
<html>
<head>
```

```
        <title>USE OF CEIL</title>
</head>
<body>
        <?php
                $my_float = 10.10;
                $my_int = ceil($my_float); // $my_int is equal to 11
                echo($my_int . "<br />");
                echo(ceil(-10.10) . "<br />");// it is equal to 10
        ?>
</body>
</html>
```

Output:

11
-10

It means (ceil(1.4)) and 2 , both are equals. And ceil(7 / 2) = 4

floor()

Return the **next lowest integer** by rounding the value downwards. The floor() function is the opposite of ceil().

Syntax:

float **floor** (float $value)

Parameters:

$value - The numeric value to round

Takes a single argument (typically a double) and returns the largest integer that is less than or equal to that argument, Here value is the numeric value to round.

```
$my_int = floor(4.7); // $my_int is equal to 4
$my_int = floor(-4.7); // $my_int is equal to –5
```

Example:

```
<head>
        <title>USE OF FLOOR</title>
</head>
<body>
        <?php
```

```
        echo(floor(10.3) . "<br />"); // 10
        echo(floor(9.999). "<br />");  //9
        echo(floor(-2.15)."<br />");  // -3
        echo(floor(-4.8)."<br />"); //-5
        echo(floor(4.8)."<br />"); //4
    ?>
</body>
</html>
```

Output:

```
10
9
-3
-5
4
```

round()

The round() function takes a float and **returns the nearest integer**. If the fractional part of the float is exactly one half, the rounding is to the highest absolute number.

Syntax:

float **round** (float $val [, int $precision = 0 [, int $mode = PHP_ROUND_HALF_UP]])

Returns the rounded value of val to specified precision (number of digits after the decimal point). precision can also be negative or zero (default).

Parameters:

val - The value to round

precision - The optional number of decimal digits to round to.

mode - One of **PHP_ROUND_HALF_UP**, **PHP_ROUND_HALF_DOWN**, **PHP_ROUND_HALF_EVEN** or **PHP_ROUND_HALF_ODD**.

Example:

```
<html>
<head>
```

```
        <title>USE OF ROUND</title>
</head>
<body>
        <?php
                echo(round(5.7)."<br/>");  // 6
                echo(round(-5.7)."<br/>");  //-6
                echo(round(-5.5)."<br/>"); //-6
                echo (round(10.5, 0, PHP_ROUND_HALF_DOWN)."<br/>");   // 10
                echo (round(10.5, 0, PHP_ROUND_HALF_EVEN)."<br/>");    // 10
                echo (round(10.5, 0, PHP_ROUND_HALF_ODD)."<br/>");     // 11
                echo (round(9.5, 0, PHP_ROUND_HALF_UP)."<br/>");       // 10
                echo (round(9.5, 0, PHP_ROUND_HALF_DOWN)."<br/>");     // 9
                echo (round(9.5, 0, PHP_ROUND_HALF_EVEN)."<br/>");     // 10
                echo (round(9.5, 0, PHP_ROUND_HALF_ODD)."<br/>");      // 9
        ?>
</body>
</html>
```

fmod()

Return the floating - point **remainder of the division** of two numbers.

Syntax:

float **fmod** (float $x , float $y)

Returns the floating point remainder of dividing the dividend (x) by the divisor (y). The reminder (*r*) is defined as: x = i * y + r, for some integer *i*. If y is non-zero, *r* has the same sign as x and a magnitude less than the magnitude of y.

Parameters:

x - The dividend
y - The divisor

It will return the floating point remainder of x/y

Example:

```
<head>
        <title>USE OF FMODE</title>
```

```
</head>
<body>
    <?php
        $x = 5.7;
        $y = 1.3;
        $r = fmod($x, $y);    // $r equals 0.5, because 4 * 1.3 + 0.5 = 5.7
        echo $r;
    ?>
</body>
</html>
```

Output:

0.5

min()

Takes any number of numerical arguments (but at least one) and **returns the smallest of the arguments**, found in the set of supplied arguments.

Syntax:

mixed **min** (array $values)
mixed **min** (mixed $value1 , mixed $value2 [, mixed $value3...])

If the first and only parameter is an array, **min()** returns the lowest value in that array. If at least two parameters are provided, **min()** returns the smallest of these values.

Note: PHP will evaluate a non-numeric string as 0 if compared to integer, but still return the string if it's seen as the lowest value. If multiple arguments evaluate to 0, **min()** will return the lowest alphanumerical string value if any strings are given, else a numeric 0 is returned.

Parameters:

values - An array containing the values.
min() - returns the numerically lowest of the parameter values.

Example:

```
<head>
    <title>USE OF MIN</title>
</head>
<body>
```

```
<?php
        echo (min(10, 20, 50, 40, 7)."<br />"); // 7
        echo (min(array(20, 4, 50))."<br />");  // 4
        echo (min(0, 'hello')."<br />");        // 0
        echo (min('hello', 0)."<br />");        // hello
        echo (min('hello', -1)."<br />");       // -1

        // With multiple arrays, min compares from left to right
        // so in our example: 2 == 2, but 4 < 5

        echo (min(array(2, 4, 8), array(2, 5, 1))."<br />"); // array(2, 4, 8)

        // If both an array and non-array are given, the array
        // is never returned as it's considered the largest

        echo (min('string', array(2, 5, 7), 42)."<br />");   // string

    ?>
</body>
</html>
```

max()

Takes any number of numerical arguments (but at least one) and **returns the largest of the arguments**, found in the set of supplied arguments.

Syntax:

mixed **max** (array $values)

mixed **max** (mixed $value1 , mixed $value2 [, mixed $value3...])

If the first and only parameter is an array, **max()** returns the highest value in that array. If at least two parameters are provided, **max()** returns the biggest of these values.

Note: PHP will evaluate a non-numeric string as 0 if compared to integer, but still return the string if it's seen as the numerically highest value. If multiple arguments evaluate to *0*, **max()** will return a numeric *0* if given, else the alphabetical highest string value will be returned.

Parameters:

values - An array containing the values.

max() - returns the numerically highest of the parameter values. If multiple values can be considered of the same size, the one that is listed first will be returned.

When **max()** is given multiple arrays, the longest array is returned. If all the arrays have the same length, **max()** will use lexicographic ordering to find the return value. When given a string it will be cast as an integer when comparing.

Example:

```
<head>
    <title>USE OF MAX</title>
</head>
<body>
    <?php
        echo (max(10, 20, 50, 40, 7)."<br />");  // 50
        echo (max(array(20, 4, 50))."<br />"); // 50
        echo (max(0, 'hello')."<br />");    // 0
        echo (max('hello', 0)."<br />");    // hello
        echo (max(-1, 'hello')."<br />");   // hello
    ?>
```

pow()

The pow() function **raises the first argument to the power of the second argument**, and returns the result.

Syntax:

number **pow** (number $base , number $exp)

Parameters:

base - The base to use
exp - The exponent

Example:

```
<html>
<head>
    <title>USE OF POW()</title>
</head>
```

```
<body>
        <?php
                echo pow(5,2) . "<br />";
                echo pow(10,2) . "<br />";
                echo pow(-5,2) . "<br />";
                echo pow(-5,-2) . "<br />";
                echo pow(-5,2.5);
        ?>
</body>
</html>
```

Output:

25
100
25
0.04
NAN

sqrt()

The sqrt() function returns the square root of a number.

Syntax:

float **sqrt** (float $arg)

Parameters:

arg - The argument to process

Example:

```
<html>
<head>
        <title>USE OF SQRT()</title>
</head>
<body>
        <?php
```

```
        echo(sqrt(0) . "<br />");
        echo(sqrt(1) . "<br />");
        echo(sqrt(9) . "<br />");
        echo(sqrt(0.64) . "<br />");
        echo(sqrt(25))
    ?>
</body>
</html>
```

Output:

0
1
3
0.8
5

rand()

The rand() function **generates a random integer**. If this function is called without parameters, it returns a **random integer between 0 and RAND_MAX**.

If you want a random number between 1000 and 2000 , use rand (1000,2000). It will return any random number between 1000 and 2000.

Syntax:

```
int rand ( void )
int rand ( int $min , int $max )
```

Parameters:

min - The lowest value to return (default: 0)
max - The highest value to return

Note: On some platforms (such as Windows) RAND_MAX is only 32768. So, if you require a range larger than 32768, you can specify min and max, or use the mt_rand() function instead.

Example:

```
<html>
<head>
```

```
        <title>USE OF RAND()</title>
</head>
<body>
        <?php
                echo(rand() . "<br />");
                echo(rand() . "<br />");
                echo(rand(1000,2000))
        ?>
</body>
</html>
```

Output:

25816
28721
1590

Array Functions

The array functions allow you to manipulate arrays.

count()

The count() function counts the **elements of an array** or the **properties of an object**.

Syntax:

int **count** (mixed $var [, int $mode = COUNT_NORMAL])

PARAMETER	DESCRIPTION
array	Required. Specifies the array or object to count.
mode	Optional. Specifies the mode of the function. Possible values: 0 - Default. Does not detect multidimensional arrays (arrays within arrays) 1 - Detects multidimensional arrays

Note: This function may return 0 if a variable isn't set, but it may also return 0 if a variable contains an empty array. The isset() function can be used to test if a variable is set.

Example:

```
<?php
        $friend = array("Hitesh", "Jitendra", "Manish", "Ganesh");
        $result = count($friend);
        echo $result;
?>
```

Output:

4

list()

The list() function is used to **assign values to a list of variables** in one operation.

Syntax:

array **list** (mixed $var1 [, mixed $var2...])

PARAMETER	DESCRIPTION
var1	Required. The first variable to assign a value to
var2	Optional. More variables to assign values to

Note: This function only works on numerical arrays.

Example:

```
<?php
        $my_mob = array("Nokia","Samsung","IPhone");
        list($a, $b, $c) = $my_mob;
        echo "I have several mobiles, a $a, a $b and a $c.";
```

?>

Output:

I have several mobiles, a Nokia, a Samsung and a IPhone.

Example:

```
<?php
        $my_mob = array("Nokia","Samsung","IPhone");
        list($b, , $c) = $my_mob;
        echo "But I only use the $a and $c mobiles.";
?>
```

Output:

But I only use the Samsung and IPhone mobiles.

in_array()

The in_array() function **searches an array for a specific value**. This function returns TRUE if the value is found in the array, or FALSE otherwise.

Syntax:

bool **in_array** (mixed **search** , array **array_name** [, bool **type**])

PARAMETER	DESCRIPTION
Search	Required. Specifies the what to search for
array_name	Required. Specifies the array to search
Type	Optional. If this parameter is set, the in_array() function searches for the search-string and specific type in the array

Note: If the search parameter is a string and the type parameter is set to TRUE, the search is case-sensitive.

Example:

```
<?php
        $test = array("Mango", "Apple", 99);
        if (in_array("Apple" ,$test))
        {
                echo "Match found";
        }
        else
        {
                echo "Match not found";
        }
        if (in_array("99" ,$test))
        {
                echo "Match found";
        }
        else
        {
                echo "Match not found";
        }
        if (in_array(99 ,$test))
        {
                echo "Match found";
        }
        else
        {
                echo "Match not found";
        }
?>
```

Output:

Match found Match not found Match found

current()

The current() function returns the **value of the current element in an array**.

Syntax:

mixed **current** (array &$array_name)

PARAMETER	DESCRIPTION
array_name	Required. Specifies the array to use

Note: This function returns FALSE on empty elements or elements with no value. This function does not move the arrays internal pointer. To do this, use the next() and prev() functions.

Example:

```
<?php
    $friend = array("Hitesh", "Jitendra", "Manish", "Ganesh");
    echo current($friend) . "<br />";
?>
```

Output:

Hitesh

next()

The next() function moves the internal pointer to, and outputs, the **next element in the array**. This function returns the value of the next element in the array on success, or FALSE if there are no more elements.

Syntax:

mixed **next**(array $array_name)

PARAMETER	DESCRIPTION
array_name	Required. Specifies the array to use

Note: This function returns FALSE on empty elements or elements with no value.

Example:

```
<?php
        $friend = array("Hitesh", "Jitendra", "Manish", "Ganesh");
        echo current($friend) . "<br />";
        echo next($friend);
?>
```

Output:

Hitesh
Jitendra

prev()

The prev() function moves the internal pointer to, and outputs, the **previous element in the array**. This function returns the value of the previous element in the array on success, or FALSE if there are no more elements.

Syntax:

mixed **prev**(array $array_name)

PARAMETER	DESCRIPTION
array_name	Required. Specifies the array to use

Note: This function returns FALSE on empty elements or elements with no value.

Example:

```
<?php
        $friend = array("Hitesh", "Jitendra", "Manish", "Ganesh");
        echo current($friend) . "<br />";
        echo next($friend) . "<br />";
        echo prev($friend);
?>
```

Output:

Hitesh
Jitendra
Hitesh

end()

The end() function moves the internal pointer to, and outputs, the **last element in the array**. This function returns the value of the last element in the array on success.

Syntax:

mixed **end**(array $array_name)

PARAMETER	DESCRIPTION
array_name	Required. Specifies the array to use

Example:

```
<?php
    $friend = array("Hitesh", "Jitendra", "Manish", "Ganesh");
    echo current($friend) . "<br />";
    echo end($friend) . "<br />";
?>
```

Output:

Hitesh
Ganesh

each()

each() function return **the current key and value pair from an array** and advance the array cursor.

Note: After **each()** has executed, the array cursor will be left on the next element of the array, or past the last element if it hits the end of the array. You have to use reset() if you want to traverse the array again using each.

Syntax:

array **each** (array &$array_name)

PARAMETER	DESCRIPTION

array_name	Required. Specifies the array to use

Example:

```
<?php
        $friend = array("Hitesh", "Jitendra", "Manish", "Ganesh");
        $f = each($friend);
        print_r ($f);
?>
```

Output:

```
Array
(
   [1] => Hitesh
   [value] => Hitesh
   [0] => 0
   [key] => 0
)
```

Example:

```
<?php
        $friend = array("frnd1"=>"Hitesh", "frnd2"=>"Jitendra");
        $f = each($friend);
        print_r ($f);
?>
```

Output:

```
Array
(
   [1] => Hitesh
   [value] => Hitesh
   [0] => frnd1
   [key] => frnd1
)
```


each() is typically used in conjunction with list() to traverse an array.

Example:

```
<?php
```

```
        $friend = array("frnd1"=>"Hitesh", "frnd2"=>"Jitendra","frnd3"=>"Ganesh");
        reset($friend);
        while (list($key, $val) = each($friend))
        {
                echo "$key => $val."<br>";
        }
?>
```

Output:

```
frnd1 => Hitesh
frnd2 => Jitendra
frnd3 => Ganesh
```

sort()

The sort() function **sorts an array by the values**. This function assigns new keys for the elements in the array. Existing keys will be removed. This function returns TRUE on success, or FALSE on failure.

Syntax:

```
bool sort ( array &$array_name [, int $sort_type = SORT_REGULAR ] )
```

PARAMETER	DESCRIPTION
array_sort	Required. Specifies the array to sort
sort_type	Optional. Specifies how to sort the array values. Possible values: SORT_REGULAR - Default. Treat values as they are (don't change types) SORT_NUMERIC - Treat values numerically SORT_STRING - Treat values as strings SORT_LOCALE_STRING - Treat values as strings, based on local settings

Example:


```
<?php
        $friend = array("Hitesh", "Jitendra", "Manish", "Ganesh");
        sort($friend);
        print_r($friend);
```

```
?>
```

Output:

```
Array
(
[0] => Ganesh
[1] => Hitesh
[2] => Jitendra
[3] => Manish
)
```

rsort()

The rsort() function **sorts an array by the values in reverse order**. This function assigns new keys for the elements in the array. Existing keys will be removed. This function returns TRUE on success, or FALSE on failure.

Syntax:

bool **rsort** (array &$array_name [, int $sort_type = SORT_REGULAR])

PARAMETER	DESCRIPTION
array_name	Required. Specifies the array to sort
Sort_type	Optional. Specifies how to sort the array values. Possible values: SORT_REGULAR - Default. Treat values as they are (don't change types) SORT_NUMERIC - Treat values numerically SORT_STRING - Treat values as strings SORT_LOCALE_STRING - Treat values as strings, based on local settings

Example:

```
<?php
        $friend = array("Hitesh", "Jitendra", "Manish", "Ganesh");
        rsort($friend);
        print_r($friend);
?>
```


Output:

```
Array
(
[0] => Manish
[1] => Jitendra
[2] => Hitesh
[3] => Ganesh
)
```

asort()

The asort() function **sorts an array by the values**. The values keep their original keys. This function returns TRUE on success, or FALSE on failure.

Syntax:

bool **asort** (array &$array [, int $sort_type = SORT_REGULAR])

PARAMETER	DESCRIPTION
array_name	Required. Specifying an array
sort_type	Optional. Specifies how to sort the array values. Possible values: SORT_REGULAR - Default. Treat values as they are (don't change types) SORT_NUMERIC - Treat values numerically SORT_STRING - Treat values as strings SORT_LOCALE_STRING - Treat values as strings, based on local settings

Example:

```
<?php
    $friend = array("Hitesh", "Jitendra", "Manish", "Ganesh");
    asort($friend);
    print_r($friend);
?>
```

Output:

```
Array
(
[3] => Ganesh
[0] => Hitesh
[1] => Jitendra
[2] => Manish
```

)

arsort()

The arsort() function **sorts an array by the values in reverse order**. The values keep their original keys. This function returns TRUE on success, or FALSE on failure.

Syntax:

bool **arsort** (array &$array_name [, int $sort_type = SORT_REGULAR])

PARAMETER	DESCRIPTION
array_name	Required. Specifying an array
sort_type	Optional. Specifies how to sort the array values. Possible values: SORT_REGULAR - Default. Treat values as they are (don't change types) SORT_NUMERIC - Treat values numerically SORT_STRING - Treat values as strings SORT_LOCALE_STRING - Treat values as strings, based on local settings

Example:

```
<?php
        $friend = array("Hitesh", "Jitendra", "Manish", "Ganesh");
        asort($friend);
        print_r($friend);
?>
```

Output:

```
Array
(
[2] => Manish
[1] => Jitendra
[0] => Hitesh
[3] => Ganesh
)
```

array_merge()

The array_merge() function **merges one or more arrays into one array**.

Syntax:

array **array_merge** (array $array1 [, array $array2 [, array $array3...]])

PARAMETER	DESCRIPTION
array1	Required. Specifies an array
array2	Optional. Specifies an array
array3	Optional. Specifies an array

You can assign one array to the function, or as many as you like. If two or more array elements have the same key, the last one overrides the others. If you assign only one array to the array_merge() function, and the keys are integers, the function returns a new array with integer keys starting at 0 and increases by 1 for each value. (See example 2)

Example 1:

```
<?php
        $a1=array("a"=>"Hitesh","b"=>"Jitendra");
        $a2=array("c"=>"Ganesh","b"=>"Manish");
        print_r(array_merge($a1,$a2));
?>
```

Output:

```
Array
(
[a] => Hitesh
[b] => Manish
[c] => Ganesh
)
```

Example 2:

```
<?php
        $a=array(3=>"Hitesh",4=>"Ganesh");
        print_r(array_merge($a));
?>
```

Output:

```
Array
(
[0] => Hitesh
[1] => Ganesh
```

)

array_reverse()

The array_reverse() function **returns an array in the reverse order**.

Syntax:

array **array_reverse** (array $array_name [, bool $preserve_keys = false])

PARAMETER	DESCRIPTION
array_name	Required. Specifies an array
preserve_keys	Optional. Possible values: true false Specifies if the function should preserve the array's keys or not.

Example:

```
<?php
    $a=array("a"=>"Hitesh","b"=>"Ganesh","c"=>"Jitendra");
    print_r(array_reverse($a));
?>
```

Output:

```
Array
(
[c] => Jitendra
[b] => Ganesh
[a] => Hitesh
)
```


ksort()

The ksort() function **sorts an array by the keys**. This function returns TRUE on success, or FALSE on failure.

Syntax:

bool **ksort** (array &$array_name [, int $sort_type = SORT_REGULAR])

PARAMETER	DESCRIPTION

array_name	Required. Specifies the array to sort
sort_type	Optional. Specifies how to sort the array values. Possible values: SORT_REGULAR - Default. Treat values as they are (don't change types) SORT_NUMERIC - Treat values numerically SORT_STRING - Treat values as strings SORT_LOCALE_STRING - Treat values as strings, based on local settings

Example:

```
<?php
        $friend = array("d"=>"Hitesh","c"=> "Jitendra","a"=> "Manish", "b"=>"Ganesh");
        sort($friend);
        print_r($friend);
?>
```

Output:

```
Array
(
[a] => Manish
[b] => Ganesh
[c] => Jitendra
[d] => Hitesh
)
```

Date Functions

Date functions are use to manipulate the date and time. It also help to display time/date in different format

date()

The date() function **formats a local time/date**.

Syntax:

string **date** (string $format [, int $timestamp])

PARAMETER	DESCRIPTION
Format	Required. Specifies how to return the result: d - The day of the month (from 01 to 31) D - A textual representation of a day (three letters) j - The day of the month without leading zeros (1 to 31) l (lowercase 'L') - A full textual representation of a day N - The ISO-8601 numeric representation of a day (1 for Monday through 7 for Sunday) S - The English ordinal suffix for the day of the month (2 characters st, nd, rd or th. Works well with j) w - A numeric representation of the day (0 for Sunday through 6 for Saturday) z - The day of the year (from 0 through 365) W - The ISO-8601 week number of year (weeks starting on Monday) F - A full textual representation of a month (January through December) m - A numeric representation of a month (from 01 to 12) M - A short textual representation of a month (three letters) n - A numeric representation of a month, without leading zeros (1 to 12) t - The number of days in the given month L - Whether it's a leap year (1 if it is a leap year, 0 otherwise) o - The ISO-8601 year number Y - A four digit representation of a year y - A two digit representation of a year a - Lowercase am or pm A - Uppercase AM or PM B - Swatch Internet time (000 to 999) g - 12-hour format of an hour (1 to 12) G - 24-hour format of an hour (0 to 23) h - 12-hour format of an hour (01 to 12) H - 24-hour format of an hour (00 to 23) i - Minutes with leading zeros (00 to 59) s - Seconds, with leading zeros (00 to 59) e - The timezone identifier (Examples: UTC, Atlantic/Azores) I (capital i) - Whether the date is in daylights savings time (1 if Daylight Savings Time, 0 otherwise)

	O - Difference to Greenwich time (GMT) in hours (Example: +0100) T - Timezone setting of the PHP machine (Examples: EST, MDT) Z - Timezone offset in seconds. The offset west of UTC is negative, and the offset east of UTC is positive (-43200 to 43200) c - The ISO-8601 date (e.g. 2004-02-12T15:19:21+00:00) r - The RFC 2822 formatted date (e.g. Thu, 21 Dec 2000 16:01:07 +0200) U - The seconds since the Unix Epoch (January 1 1970 00:00:00 GMT)
Timestamp	Optional.

Example:

```
<html>
<head>
        <title>USE OF DATE()</title>
</head>
<body>
        <?php
                echo("Result with date():<br />");
                echo (date("Y-m-d")."<br/>"); // Today's Date
                echo (date("Y-m-d H:i:s",time())."<br/>"); //Current date and time
                echo(date("l") . "<br />");
                echo(date("l dS \of F Y h:i:s A") . "<br />");
        ?>
</body>
</html>
```

Output:

Result with date():
2011-04-10
2011-04-10 23:53:32
Sunday
Sunday 10th of July 2011 11:53:32 PM

Example:

```
<html>
<head>
```

```
        <title>USE OF DATE()</title>
</head>
<body>
        <?php
                echo "Today is ".date("F d, Y")."<br />";
                echo "Today is ".date("l")."<br />";
        ?>
</body>
</html>
```

Output:

Today is April 11, 2011
Today is Monday

getdate()

The getdate() function returns an **array that contains date and time information** for a Unix timestamp. The returning array contains ten elements with relevant information needed when formatting a date string:

- [seconds] – seconds
- [minutes] – minutes
- [hours] – hours
- [mday] - day of the month
- [wday] - day of the week
- [year] – year
- [yday] - day of the year
- [weekday] - name of the weekday
- [month] - name of the month
- [0]- Number of seconds since the Unix epoch (timestamp). (System Dependent, typically -2147483648 through 2147483647.)

Syntax:

array **getdate** ([int $timestamp = time()])

PARAMETER	DESCRIPTION
timestamp	Optional. Specifies the time in Unix time format

Example:

```
<html>
<head>
        <title>USE OF GETDATE()</title>
</head>
<body>
        <?php
                $today = getdate();
                print_r($today);  //print_r(getdate());
        ?>
</body>
</html>
```

Output:

```
Array
(
   [seconds] => 40
   [minutes] => 58
   [hours]   => 21
   [mday]    => 17
   [wday]    => 2
   [mon]     => 6
   [year]    => 2011
   [yday]    => 191
   [weekday] => Tuesday
   [month]   => April
   [0]       => 1055901520
)
```

Example:

```
<html>
<head>
        <title>USE OF GETDATE()</title>
</head>
<body>
        <?php
                $my_t=getdate(date("U"));
                print("$my_t[weekday], $my_t[month] $my_t[mday], $my_t[year]");
```

```
    ?>
</body>
</html>
```

Output:

Monday, April 11, 2011

DateTime::setDate()

DateTime::setDate — Sets the date.

Description

Object oriented style

public DateTime **DateTime::**setDate (int $year , int $month , int $day)

Procedural style

DateTime date_date_set (DateTime $object , int $year , int $month , int $day)

Resets the current date of the DateTime object to a different date.

Parameters

object - Procedural style only: A DateTime object returned by date_create(). The function modifies this object.

year - Year of the date.

month - Month of the date.

day - Day of the date.

Returns the DateTime object for method chaining or **FALSE** on failure.

checkdate()

The checkdate() function returns **true if the specified date is valid, and false otherwise**.

A date is valid if:

- month is between 1 and 12 inclusive.
- day is within the allowed number of days for the particular month.

- year is between 1 and 32767 inclusive.

Syntax:

bool **checkdate** (int $month , int $day , int $year)

Parameters

month - The month is between 1 and 12 inclusive.
day - The day is within the allowed number of days for the given month. Leap years are taken into consideration.
year - The year is between 1 and 32767 inclusive.

Example:

```
<html>
<head>
    <title>USE OF CHECKDATE()</title>
</head>
<body>
    <?php
        var_dump(checkdate(12,31,2000)); //true
        var_dump(checkdate(2,29,2003));  //false
        var_dump(checkdate(2,29,2004));  // true
    ?>
</body>
</html>
```

time()

The time() function returns the **current time as a Unix timestamp** (the number of seconds since January 1 1970 00:00:00 GMT).

Syntax:

int **time** (void)

Parameter

void - Optional

Example:

```
<html>
<head>
        <title>USE OF TIME()</title>
</head>
<body>
        <?php
                $t=time();
                echo($t . "<br />");
                echo(date("D F d Y",$t));
        ?>
</body>
</html>
```

Output:

1310325145
Mon April 11 2011

mktime()

The mktime() function returns the **Unix timestamp for a date**. This timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

The mktime() function is useful for doing date arithmetic and validation. It will automatically calculate the correct value for out-of-range input:

Syntax:

int **mktime** (int hour, int minute, int second, int month, int day, int year, int is_dst)

PARAMETER	DESCRIPTION
Hour	Optional. Specifies the hour
Minute	Optional. Specifies the minute
Second	Optional. Specifies the second
Month	Optional. Specifies the numerical month
Day	Optional. Specifies the day

Year	Optional. Specifies the year. The valid range for year is on some systems between 1901 and 2038. However this limitation is overcome in PHP 5
is_dst	Optional. Set this parameter to 1 if the time is during daylight savings time (DST), 0 if it is not, or -1 (the default) if it is unknown. If it's unknown, PHP tries to find out itself (which may cause unexpected results). **Note:** This parameter became deprecated in PHP 5. The new timezone handling features should be used instead.

Note: If the arguments are invalid, the function returns false.

Example:

```
<html>
<head>
        <title>USE OF MKTIME()</title>
</head>
<body>
        <?php
                echo(date("M-d-Y",mktime(0,0,0,12,36,2001))."<br />");
                echo(date("M-d-Y",mktime(0,0,0,14,1,2001))."<br />");
                echo(date("M-d-Y",mktime(0,0,0,1,1,2001))."<br />");
                echo(date("M-d-Y",mktime(0,0,0,1,1,99))."<br />");
        ?>
</body>
</html>
```

Output:

Jan-05-2002
Feb-01-2002
Jan-01-2001
Jan-01-1999

File System Functions

The filesystem functions allow you to access and manipulate the filesystem.

fopen()

The fopen() function **opens a file or URL**. If fopen() fails, it returns FALSE and an error on failure. You can hide the error output by adding an '@' in front of the function name.

Syntax:

resource **fopen** (string $file , string $mode [, bool $use_include_path = false [,resource $context]])

PARAMETER	DESCRIPTION
File	Required. Specifies the file or URL to open
Mode	Required. Specifies the type of access you require to the file/stream. Possible values: "r" (Read only. Starts at the beginning of the file) "r+" (Read/Write. Starts at the beginning of the file) "w" (Write only. Opens and clears the contents of file; or creates a new file if it doesn't exist) "w+" (Read/Write. Opens and clears the contents of file; or creates a new file if it doesn't exist) "a" (Write only. Opens and writes to the end of the file or creates a new file if it doesn't exist) "a+" (Read/Write. Preserves file content by writing to the end of the file) "x" (Write only. Creates a new file. Returns FALSE and an error if file already exists) "x+" (Read/Write. Creates a new file. Returns FALSE and an error if file already exists)
include_path	Optional. Set this parameter to '1' if you want to search for the file in the include_path (in php.ini) as well
context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream

Example:

```
<?php
        $file = fopen("test.txt","r");
        $file = fopen("/home/test/test.txt","r");
        $file = fopen("/home/test/test.gif","wb");
        $file = fopen("http://www.example.com/","r");
        $file = fopen("ftp://user:password@example.com/test.txt","w");
?>
```

fread()

The fread() **reads from an open file**. The function will stop at the end of the file or when it reaches the specified length, whichever comes first. This function returns the read string, or FALSE on failure.

Syntax:

string **fread** (resource $file , int $length)

PARAMETER	DESCRIPTION
File	Required. Specifies the open file to read from
length	Required. Specifies the maximum number of bytes to read

Note: This function is binary-safe (meaning that both binary data, like images, and character data can be written with this function).

Example:
Read 10 bytes from file:

```
<?php
        $file = fopen("test.txt","r");
        fread($file,"10");
        fclose($file);
?>
```

Example:
Read entire file:

```
<?php
        $file = fopen("test.txt","r");
        fread($file,filesize("test.txt"));
        fclose($file);
?>
```


fwrite()

The fwrite() **writes to an open file**. The function will stop at the end of the file or when it reaches the specified length, whichever comes first. This function returns the number of bytes written, or FALSE on failure.

Syntax:

int **fwrite** (resource $file , string $string [, int $length])

PARAMETER	DESCRIPTION
File	Required. Specifies the open file to write to
String	Required. Specifies the string to write to the open file
Length	Optional. Specifies the maximum number of bytes to write

Note: This function is binary-safe (meaning that both binary data, like images, and character data can be written with this function).

Example:

```
<?php
        $file = fopen("test.txt","w");
        echo fwrite($file,"Hello World. Testing!");
        fclose($file);
?>
```

Output:

21

fclose()

The fclose() function **closes an open file**. This function returns TRUE on success or FALSE on failure.

Syntax:

bool **fclose** (resource $file)

PARAMETER	DESCRIPTION
File	Required. Specifies the file to close

Example:

```
<?php
        $file = fopen("test.txt","r");
        //some code to be executed
```

```
    fclose($file);
?>
```

file_exists()

The file_exists() function checks whether or not a file or directory exists. This function returns TRUE if the file or directory exists, otherwise it returns FALSE.

Syntax:

```
    bool file_exists ( string $path )
```

PARAMETER	DESCRIPTION
Path	Required. Specifies the path to check

Example:

```
<?php
    echo file_exists("test.txt");
?>
```

Output:

1

is_readable()

The is_readable() function **checks whether the specified file is readable**. This function returns TRUE if the file is readable.

Syntax:

```
    bool is_readable ( string $file )
```

PARAMETER	DESCRIPTION
File	Required. Specifies the file to check

Note: The result of this function are cached. Use clearstatcache() to clear the cache.

Example:

```
<?php
        $file = "test.txt";
        if(is_readable($file))
        {
                echo ("$file is readable");
        }
        else
        {
                echo ("$file is not readable");
        }
?>
```

Output:

test.txt is readable

is_writable()

The is_writable() function **checks whether the specified file is writeable**. This function returns TRUE if the file is writeable.

Syntax:

bool **is_writable** (string $file)

PARAMETER	DESCRIPTION
file	Required. Specifies the file to check

Note: The result of this function are cached. Use clearstatcache() to clear the cache.

Example:

```
<?php
        $file = "test.txt";
        if(is_writable($file))
        {
                echo ("$file is writeable");
        }
        else
        {
                echo ("$file is not writeable");
```

```
    }
?>
```

Output:

test.txt is writeable

fgets()

The fgets() function returns a **line from an open file**. The fgets() function stops returning on a new line, at the specified length, or at EOF, whichever comes first. This function returns FALSE on failure.

Syntax:

string **fgets** (resource $file [, int $length])

PARAMETER	DESCRIPTION
file	Required. Specifies the file to read from
length	Optional. Specifies the number of bytes to read. Default is 1024 bytes.

Example:

```
<?php
    $file = fopen("test.txt","r");
    echo fgets($file);
    fclose($file);
?>
```


Output:

Hello, this is a test file.

Example:
Read file line by line:

```
<?php
    $file = fopen("test.txt","r");
    while(! feof($file))
```

```
{
        echo fgets($file). "<br />";
}
fclose($file);
?>
```

Output:

```
Hello, this is a test file.
There are three lines here.
This is the last line.
```

fgetc()

The fgetc() function returns a **single character from an open file**.

Syntax:

string **fgetc** (resource $file)

PARAMETER	DESCRIPTION
File	Required. Specifies the file to check

Note: This function is slow and should not be used on large files. If you need to read one character at a time from a large file, use fgets() to read data one line at a time and then process the line one character at a time with fgetc().

Example:

```
<?php
        $file = fopen("test2.txt","r");
        echo fgetc($file);
        fclose($file);
?>
```

Output:

```
H
```

Example:

Read file character by character:

```
<?php
        $file = fopen("test2.txt","r");
        while (! feof ($file))
        {
                echo fgetc($file);
        }
        fclose($file);
?>
```

Output:

Hello, this is a test file.

file()

The file() **reads a file into an array**. Each array element contains a line from the file, with newline still attached.

Syntax:

array **file** (string $file [, int $flags = 0 [, resource $context]])

PARAMETER	DESCRIPTION
File	Required. Specifies the file to read
Flags	Optional. Set this parameter to '1' if you want to search for the file in the include_path (in php.ini) as well
context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream. Can be skipped by using NULL.

Note: This function became binary-safe in PHP 4.3. (meaning that both binary data, like images, and character data can be written with this function).

Example:

```
<?php
        print_r(file("test.txt"));
?>
```

Output:

```
Array
(
[0] => Hello World. Testing testing!
[1] => Another day, another line.
[2] => If the array picks up this line,
[3] => then is it a pickup line?
)
```

file_get_contents()

The file_get_contents() **reads a file into a string.** This function is the preferred way to read the contents of a file into a string. Because it will use memory mapping techniques, if this is supported by the server, to enhance performance.

Syntax:

string **file_get_contents** (string $file [, bool $use_include_path = false [, resource $context [, int $offset = -1 [, int $maxlen]]]])

PARAMETER	DESCRIPTION
file	Required. Specifies the file to read
use_include_path	Optional. Set this parameter to '1' if you want to search for the file in the include_path (in php.ini) as well
context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream. Can be skipped by using NULL.
offset	Optional. Specifies where in the file to start reading. This parameter was added in PHP 5.1
maxlen	Optional. Specifies how many bytes to read. This parameter was added in PHP 5.1

Note: This function is binary-safe (meaning that both binary data, like images, and character data can be written with this function).

Example:

```
<?php
        echo file_get_contents("test.txt");
?>
```

Output:

This is a test file with test text.

file_put_contents()

The file_put_contents() **writes a string to a file**. This function follows these rules when accessing a file:

- If FILE_USE_INCLUDE_PATH is set, check the include path for a copy of *filename*.
- Create the file if it does not exist.
- Open the file.
- Lock the file if LOCK_EX is set.
- If FILE_APPEND is set, move to the end of the file. Otherwise, clear the file content.
- Write the data into the file.
- Close the file and release any locks

This function returns the number of character written into the file on success, or FALSE on failure.

Syntax:

int **file_put_contents** (string $file , mixed $data [, int $mode = 0 [, resource $context]])

PARAMETER	DESCRIPTION
File	Required. Specifies the file to write to. If the file does not exist, this function will create one
Data	Required. The data to write to the file. Can be a string, an array or a data stream
Mode	Optional. Specifies how to open/write to the file. Possible values: FILE_USE_INCLUDE_PATH FILE_APPEND LOCK_EX
Context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream.

Note: Use FILE_APPEND to avoid deleting the existing content of the file.

Example:

```
<?php
        echo file_put_contents("test.txt","Hello World. Testing!");
?>
```

Output:

21

fseek()

The fseek() function **seeks in an open file**. This function moves the file pointer from its current position to a new position, forward or backward, specified by the number of bytes. This function returns 0 on success, or -1 on failure. Seeking past EOF will not generate an error.

Syntax:

int **fseek** (resource $file , int $offset [, int $whence = SEEK_SET])

PARAMETER	DESCRIPTION
File	Required. Specifies the open file to seek in
offset	Required. Specifies the new position (measured in bytes from the beginning of the file)
whence	Optional. (added in PHP 4). Possible values: SEEK_SET - Set position equal to offset. Default SEEK_CUR - Set position to current location plus offset SEEK_END - Set position to EOF plus offset (to move to a position before EOF, the offset must be a negative value)

Note: Find the current position by using ftell()!

Example:

```
<?php
    $file = fopen("test.txt","r");
    // read first line
    fgets($file);
    // move back to beginning of file
    fseek($file,0);
?>
```


ftell()

The ftell() function returns the **current position in an open file**. Returns the current file pointer position, or FALSE on failure.

Syntax:

int **ftell** (resource $handle)

PARAMETER	DESCRIPTION
File	Required. Specifies the open file to check

Example:

```
<?php
        $file = fopen("test.txt","r");
        // print current position
        echo ftell($file);
        // change current position
        fseek($file,"15");
        // print current position again
        echo "<br />" . ftell($file);
        fclose($file);
?>
```

Output:

```
0
15
```

rewind()

The rewind() function **"rewinds" the position of the file pointer to the beginning of the file**. This function returns TRUE on success, or FALSE on failure.

Syntax:

bool **rewind** (resource $file)

PARAMETER	DESCRIPTION
file	Required. Specifies the open file

Example:

```
<?php
        $file = fopen("test.txt","r");
        //Change position of file pointer
        fseek($file,"15");
        //Set file pointer to 0
```

```
    rewind($file);
    fclose($file);
?>
```

rename()

The rename() function **renames a file or directory**. This function returns TRUE on success, or FALSE on failure.

Syntax:

bool **rename** (string $oldname , string $newname [, resource $context])

PARAMETER	DESCRIPTION
oldname	Required. Specifies the file or directory to be renamed
newname	Required. Specifies the new name of the file or directory
context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream

Example:

```
<?php
    rename("images","pictures");
?>
```

unlink()

The unlink() function **deletes a file**. This function returns TRUE on success, or FALSE on failure.

Syntax:

bool **unlink** (string $file [, resource $context])

PARAMETER	DESCRIPTION
file	Required. Specifies the file to delete
context	Optional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream

Example:

```
<?php
        $file = "test.txt";
        if (!unlink($file))
        {
                echo ("Error deleting $file");
        }
        else
        {
                echo ("Deleted $file");
        }
?>
```

copy()

The copy() function **copies a file**. It returns TRUE on success and FALSE on failure.

Syntax:

bool **copy** (string $file , string $to_file)

PARAMETER	DESCRIPTION
File	Required. Specifies the file to copy
to_file	Required. Specifies the file to copy to

Note: If the destination file already exists, it will be overwritten.

Example:


```
<?php
        echo copy("source.txt","target.txt");
?>
```

Output:

1

Miscellaneous Functions

The miscellaneous functions were only placed here because none of the other categories seemed to fit.

define()

The define() function **defines a constant**. Constants are much like variables, except for the following differences:

- A constant's value cannot be changed after it is set.
- Constant names do not need a leading dollar sign ($).
- Constants can be accessed regardless of scope.
- Constant values can only be strings and numbers.

Syntax:

bool **define** (string $name , mixed $value [, bool $case_insensitive = false])

PARAMETER	DESCRIPTION
name	Required. Specifies the name of the constant
value	Required. Specifies the value of the constant
case_insensitive	Optional. Specifies whether the constant name should be case-insensitive. If set to TRUE, the constant will be case-insensitive. Default is FALSE (case-sensitive)

Example:
Define a case-sensitive constant:

```
<?php
    define("NAME","Hello, I am Chetan Rathod");
    echo constant("NAME");
?>
```

Output:

Hello, I am Chetan Rathod

Example:
Define a case-insensitive constant:

```
<?php
    define("NAME"," Hello, I am Chetan Rathod ",TRUE);
    echo constant("name");
```

?>

Output:

Hello, I am Chetan Rathod

constant()

The constant( ) function **returns the value of a constant**.

Syntax:

mixed **constant** (string $constant)

PARAMETER	DESCRIPTION
constant	Required. Specifies the name of the constant to check

Note: This function also works with class constants.

Example:

```
<?php
        //define a constant
        define("NAME","Hello, I am Bhumika");
        echo constant("NAME");
?>
```

Output:

Hello, I am Bhumika

defined()

The defined() function **checks whether a constant exists**. Returns TRUE if the constant exists, or FALSE otherwise.

Syntax:

bool **defined** (string $name)

PARAMETER	DESCRIPTION
Name	Required. Specifies the name of the constant to check

Example:

```
<?php
    define("NAME","Hello, I am Chetan");
    echo defined("NAME");
?>
```

Output:

1

die()

The die() function **prints a message and exits the current script**. This function is an alias of the exit() function.

Syntax:

die(String $message)

PARAMETER	DESCRIPTION
message	Required. Specifies the message or status number to write before exiting the script. The status number will not be written to the output.

Example:

```
<?php
    $site = "https://www.jump2learn.com/";
    fopen($site,"r")
    or die("Unable to connect to $site");
?>
```

exit()

The exit() function **prints a message and exits the current script**. This function is an alias of the die() function.

Syntax:

void **exit**(String $message)
void **exit**(Int $status)

PARAMETER	DESCRIPTION
message,status	Required. Specifies the message or status number to write before exiting the script. The status number will not be written to the output.

Example:

```
<?php
        $site = "http://www.vivekanandcollege.ac.in/";
        fopen($site,"r")
        or exit("Unable to connect to $site");
?>
```

sleep()

The sleep() function **delays execution of the current script for a specified number of seconds**.

Syntax:

int **sleep** (int $seconds)

PARAMETER	DESCRIPTION
seconds	Required. Specifies the number of seconds to delay the script

Example:

```
<?php
        echo date('h:i:s') . "<br />";
        //sleep for 10 seconds
        sleep(10);
        //start again
        echo date('h:i:s');
?>
```

Output:

06:10:06
06:10:16

usleep()

The usleep() function delays execution of the current script for a specified number of microseconds (a microsecond equals one millionth of a second).

Syntax:

void **usleep** (int $micro_seconds)

PARAMETER	DESCRIPTION
microseconds	Required. Specifies the number of microseconds to delay the script

Note: This function did not work on Windows platforms until PHP 5.

Example:

```
<?php
        echo date('h:i:s') . "<br />";
        //sleep for 10 seconds
        usleep(10000000);
        //start again
        echo date('h:i:s');
?>
```

Output:

```
06:12:24
06:12:34
```

include()

The include() statement **includes and evaluates the specified file**. Files are included based on the given file path. If the file isn't found in the include_path, **include()** will finally check in the calling script's own directory and the current working directory before failing.

The include() will produce a warning if it cannot find a file.(this is different behavior from require(), which will produce a fatal error.)

If a path is absolute (starting with a drive letter or \ on Windows, or / on Unix/Linux systems) or relative to the current directory (starting with . or ..)

Syntax:

include 'path of file'

Example:

data.php

```
<?php
        $color = 'green';
        $fruit = 'apple';
?>
```

test.php

```
<?php
        echo "A $color $fruit"; // A
        include 'data.php';
        echo "A $color $fruit"; // A green apple
?>
```

require()

require() is same as include() except upon failure it will also **produce a fatal E_COMPILE_ERROR level error**.

In other words, it will stop the script whereas include() only produce a warning (**E_WARNING**) which allows the script to continue.

header()

header() is used to **send a raw** HTTP **header**.

Note: header() must be called before any actual output is sent, either by normal HTML tags, blank lines in a file, or from PHP.

Syntax:

void **header** (string $string [, bool $replace = true [, int $http_response_code]])

Parameters:

string - The header string.

There are two special-case header calls. The first is a header that starts with the string "*HTTP/*" (case is not significant), which will be used to figure out the HTTP status code to send.

```
<?php
    header("HTTP/1.0 404 Not Found");
?>
```

The second special case is the "Location:" header. Not only does it send this header back to the browser, but it also returns a status code to the browser.

```
<?php
    header("Location: http://www.example.com/"); /* Redirect browser */
    /* Make sure that code below does not get executed when we redirect. */
    exit;
?>
```

replace - The optional replace parameter indicates whether the header should replace a previous similar header, or add a second header of the same type. By default it will replace, but if you pass in **FALSE** as the second argument you can force multiple headers of the same type. For example:

```
<?php
    header('WWW-Authenticate: Negotiate');
    header('WWW-Authenticate: NTLM', false);
?>
```

http_response_code - Forces the HTTP response code to the specified value. Note that this parameter only has an effect if the string is not empty.

Example 1:

If you want the user to be prompted to save the data you are sending, such as a generated PDF file, you can use the » Content-Disposition header to supply a recommended filename and force the browser to display the save dialog.

```
<?php
    // We'll be outputting a PDF
    header('Content-type: application/pdf');
```

```
        // It will be called downloaded.pdf
        header('Content-Disposition: attachment; filename="downloaded.pdf"');
        // The PDF source is in original.pdf
        readfile('original.pdf');
?>
```

Example 2:

PHP scripts often generate dynamic content that must not be cached by the client browser or any proxy caches between the server and the client browser. Many proxies and clients can be forced to disable caching with:

```
<?php
        header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
        header("Expires: Sat, 26 Jul 1997 05:00:00 GMT"); // Date in the past
?>
```

get_browser()

The get_browser() function **looks up the user's browscap.ini file and returns the capabilities of the user's browser**. This function returns an object or an array with information about the user's browser on success, or FALSE on failure.

Syntax:

```
mixed get_browser ([ string $user_agent [, bool $return_array = false ]] )
```

PARAMETER	DESCRIPTION
user_agent	Optional. Specifies the name of an HTTP user agent. Default is the value of $HTTP_USER_AGENT. You can bypass this parameter with NULL
return_array	Optional. If this parameter is set to TRUE, the function will return an array instead of an object

Example:

```
<?php
        echo $_SERVER['HTTP_USER_AGENT'] . "<br /><br />";
        $browser = get_browser(null,true);
        print_r($browser);
?>
```

This code display the capabilities of the user's browser in the form of array.

show_source()

The show_source() function **outputs a file with the PHP syntax highlighted**. The syntax is highlighted by using HTML tags. The colors used for highlighting can be set in the php.ini file or with the ini_set() function. This function returns TRUE on success, or FALSE on failure.

Syntax:

```
bool show_source(String  $filename, bool $return)
```

PARAMETER	DESCRIPTION
Filename	Required. Specifies the file to display
Return	Optional. If this parameter is set to TRUE, this function will return the highlighted code as a string, instead of printing it out. Default is FALSE

Note: When using this function, the entire file will be displayed - including passwords and any other sensitive information!

Example:

"test.php":

```
<html>
<body>
        <?php
                show_source("test.php");
        ?>
</body>
</html>
```

Output:

```
<html>
<body>
        <?php show_source("test.php");?>
</body>
</html>
```

If you select "View source" in the browser window, it will look something like this:

```
<html>
<body>
        <span style="color: #000000">&lt;html&gt;
        <br />
        &lt;body&gt;
        <br />
        <code>
                <span style="color: #0000BB">&lt;?php
                <br />show_source</span>
                <span style="color: #007700">(</span>
                <span style="color: #DD0000">"test.php"</span>
                <span style="color: #007700">);<br /></span>
                <span style="color: #0000BB">?&gt;<br /></span>
                &lt;/body&gt;
                <br />
                &lt;/html&gt;</span>
        </code>
</body>
</html>
```

USER-DEFINED FUNCTIONS

Function is a block of statements. To keep the script from being executed when the page loads, you can put it into a function. A function will be executed by a call to the function. You may call a function from anywhere within a page.

Create a PHP Function

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

Syntax:

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

PHP function guidelines:

- Function name is user defined.
- The function name can start with a letter or underscore (not a number)

Example:

A simple function that writes my name when it is called:

```
<html>
<body>
    <?php
        function writeName()
        {
            echo “Chetan Rathod";
        }
        echo "My name is ";
        writeName();
    ?>
</body>
</html>
```

Output:

My name is Chetan Rathod

Function’s Parameters

To add more functionality to a function, we can add parameters. A parameter is just like a variable. Parameters are specified after the function name, inside the parentheses.

Example 1:

The following example will write different first names, but equal last name:

```
<html>
<body>
    <?php
```

```
        function writeName($fname)
        {
                echo $fname . " Rathod.<br>";
        }
        echo "My name is ";
        writeName("Chetan");
        echo "My brother's name is ";
        writeName("Jignesh");
    ?>
</body>
</html>
```

Output:

My name is Chetan Rathod.
My brother's name is Jignesh Rathod.

You can pass any number of parameters to the function.

Arguments:

In function we supply the value to the Function's parameters which are known as argument. We can pass argument to the function at the calling time of function. Arguments are specified after the function name, inside the parentheses.

There are two methods of passing argument to the Function.

1. Passing Arguments by Value
2. Passing Arguments by Reference

Passing Arguments by Value

Default Method is Pass by Value. Here we pass the value to the function. So changes of the variable is affected only inside the function. Outside the Function the values of variables remain no change.

Example:

```
<html>
<body>
    $a=10;
    $b=20;
```

```
<?php
        function Swap_ab($a,$b)
        {
                $temp=$a;
                $a=$b;
                $b=$temp;
                echo "Value of $a is " .$a<br>";
                echo "Value of $b is " .$b;
        }
        echo "Inside the Function: <br> ";
        Swap_ab($a,$b);
        echo "Outside the Function: <br> ";
        echo "Value of $a is " .$a<br>";
        echo "Value of $b is " .$b;
?>
</body>
</html>
```

Output:

Inside The Function:
Value of $a is 20
Value of $b is 10
Outside The Function:
Value of $a is 10
Value of $b is 20

PASSING ARGUMENTS BY REFERENCE

We can Pass the arguments by References(address) also. You may want any changes made to an argument within a function to be reflected outside of the function's scope. Passing the argument by reference accomplishes this.

Here we pass the reference of the value(original value) to the function. So a change of the variable inside the function is also affected outside from the function. It changes the value of the variables. Use & sign as a prefix to variable name for passing as a reference.

Example:

```
<html>
<body>
        $a=10;
        $b=20;
                <?php
                        function Swap_ab(&$a,&$b)
                        {
                                $temp=$a;
                                $a=$b;
                                $b=$temp;
                                echo "Value of $a is  " .$a<br>";
                                echo "Value of $b is  " .$b;
                        }
                        echo "Inside the Function: <br> ";
                        Swap_ab($a,$b);
                        echo "Outside the Function: <br> ";
                        echo "Value of $a is  " .$a<br>";
                        echo "Value of $b is  " .$b;
                ?>
</body>
</html>
```

Output:

Inside The Function:
Value of $a is 20
Value of $b is 10
Outside The Function:
Value of $a is 20
Value of $b is 10

Here you can see the difference of pass by value and pass by reference.

DEFAULT ARGUMENT VALUES

Default values can be assigned to input arguments, which will be automatically assigned to the argument if no other value is provided.

Example:

```
function  add($a,$b=10)
{
        $total=$a+$b;
        echo $total;
}
```

Calling:

```
add(50);
```

Output is 60 because default value is 10.

If multiple optional arguments are specified, you can selectively choose which ones are passed along.

Example:

```
function  add($a,$b="",$c="")
{
        $total=$a+$b+$c;
        echo $total;
}
```

Calling:

```
add(10,"",40);
```

Here output is 50

RETURN VALUES

To let a function return a value, use the return statement.

Example:

```
<html>
<body>
    <?php
        function add($x,$y)
        {
            $total=$x+$y;
            return $total;
        }
        echo "3 + 6 = " . add(1,16);
    ?>
</body>
</html>
```

Output:

3+ 6 = 9