Unit – I

Introduction to PHP: Declaring variables, data types, arrays, strings, operators, expressions, control structures, functions, Reading data from web form controls like text boxes, radio buttons, lists etc,. Handling File Uploads, Connecting to databases (MySQL as reference), executing simple queries, handling results, Handling sessions and cookies.

File Handling in PHP: File operations like opening, closing, reading, writing, appending, deleting etc. on text and binary files, listing directories.

Introduction to PHP :

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages.
- PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.
- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"
- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data
- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
echo "My first PHP script!";
?>
</body>
</html>
```

OUTPUT:
My first PHP script!

Basic PHP Syntax

A PHP script can be placed anywhere in the document.
A PHP script starts with **<?php** and ends with **?>**:

```
<?php
// PHP code goes here
?>
```

The default file extension for PHP files is ".php".
A PHP file normally contains HTML tags, and some PHP scripting code.
Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>
</body>
</html>
```

Note: PHP statements end with a semicolon (;).

Comments in PHP

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
// This is a single-line comment
# This is also a single-line comment
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/
// You can also use comments to leave out parts of a code line
$x = 5 /* + 15 */ + 5;
echo $x;
?>
</body>
</html>
```

In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive.

```
<!DOCTYPE html>
<html>
<body>
```

```
<?php
$color = "red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>
</body>
</html>
```

OUTPUT:

My car is red
My house is
My boat is

Declaring variables:

Variables are "containers" for storing information.
A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive ($age and $AGE are two different variables)
- Remember that PHP variable names are case-sensitive!

Example:

```
<!DOCTYPE html>
<html>
<body>
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
echo $txt;
echo "<br>";
echo $x;
echo "<br>";
echo $y;
?>
</body>
</html>
```

OUTPUT:

Hello world!
5
10.5

Output Variables

The PHP echo statement is often used to output data to the screen.
The following examples will show how to output text and a variable:
Example1:

```
<!DOCTYPE html>
<html>
<body>
<?php
$txt = "W3Schools.com";
echo "I love $txt!";
?>
</body>
</html>
```

OUTPUT:
I love W3Schools.com!

Example2:

```
<!DOCTYPE html>
<html>
<body>
<?php
$txt = "W3Schools.com";
echo "I love " . $txt . "!";
?>
</body>
</html>
```

OUTPUT:
I love W3Schools.com!

Example3:

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 5;
$y = 4;
echo $x + $y;
?>
</body>
</html>
```

OUTPUT:
9

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is.

PHP automatically converts the variable to the correct data type, depending on its value.

In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

PHP Variables Scope

In PHP, variables can be declared anywhere in the script.
The scope of a variable is the part of the script where the variable can be referenced/used.
PHP has three different variable scopes:

- local
- global
- static

Global and Local Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 5; // global scope
 function myTest() {
   // using x inside this function will generate an error
   echo "<p>Variable x inside function is: $x</p>";
}
myTest();
echo "<p>Variable x outside function is: $x</p>";
?>
</body>
</html>
```

OUTPUT:
Variable x inside function is:
Variable x outside function is: 5

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

```
<!DOCTYPE html>
<html>
<body>
```

```
<?php
function myTest() {
   $x = 5; // local scope
   echo "<p>Variable x inside function is: $x</p>";
}
myTest();
// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
</body>
</html>
```

OUTPUT: Variable x inside function is: 5
Variable x outside function is:

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

PHP The global Keyword

The global keyword is used to access a global variable from within a function.
To do this, use the global keyword before the variables (inside the function):

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 5;
$y = 10;
function myTest() {
   global $x, $y;
   $y = $x + $y;
}
myTest();  // run function
echo $y; // output the new value for variable $y
?>
</body>
</html>
```

OUTPUT:
15

PHP also stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 5;
$y = 10;
function myTest() {
  $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}
myTest();
echo $y;
?>
</body>
</html>
```

OUTPUT:
15

PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job.

To do this, use the **static** keyword when you first declare the variable:

```
<!DOCTYPE html>
<html>
<body>

<?php
function myTest() {
  static $x = 0;
  echo $x;
  $x++;
}

myTest();
echo "<br>";
myTest();
echo "<br>";
myTest();
?>

</body>
</html>
OUTPUT:
0
1
2
```

Data Types:

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource

PHP String

A string is a sequence of characters, like "Hello world!".

A string can be any text inside quotes. You can use single or double quotes:

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = "Hello world!";
$y = 'Hello world!';

echo $x;
echo "<br>";
echo $y;
?>

</body>
</html>
OUTPUT:
Hello world!
Hello world!
```

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

In the following example $x is an integer. The PHP var_dump() function returns the data type and value:

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 5985;
var_dump($x);
?>
</body>
</html>
OUTPUT:
int(5985)
```

PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.

In the following example $x is a float. The PHP var_dump() function returns the data type and value:

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 10.365;
var_dump($x);
?>

</body>
</html>
OUTPUT:
```

float(10.365)

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
$y = false;
```

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

PHP Array

An array stores multiple values in one single variable.

In the following example $cars is an array. The PHP var_dump() function returns the data type and value:

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
?>

</body>
</html>
```

OUTPUT:

```
array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }
```

PHP Object

An object is a data type which stores data and information on how to process that data.

In PHP, an object must be explicitly declared.

First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

```
<!DOCTYPE html>
<html>
<body>
```

```
<?php
class Car {
   function Car() {
      $this->model = "VW";
   }
}
// create an object
$herbie = new Car();

// show object properties
echo $herbie->model;
?>

</body>
</html>
```

OUTPUT:
VW

PHP NULL Value

Null is a special data type which can have only one value: NULL.

A variable of data type NULL is a variable that has no value assigned to it.

Tip: If a variable is created without a value, it is automatically assigned a value of NULL.

Variables can also be emptied by setting the value to NULL:

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>
</body>
</html>
```

OUTPUT:
NULL

PHP Resource

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP.A common example of using the resource data type is a database call.We will not talk about the resource type here, since it is an advanced topic.

PHP Arrays

An array stores multiple values in one single variable:

```
<!DOCTYPE html>
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
</body>
</html>
```

OUTPUT:

I like Volvo, BMW and Toyota.

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

In PHP, the array() function is used to create an array:

```
array();
```

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

```
<!DOCTYPE html>
<html>
```

```
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
</body>
</html>
```

OUTPUT:

I like Volvo, BMW and Toyota.

Get The Length of an Array - The count() Function: The count() function is used to return the length (the number of elements) of an array:

```
<!DOCTYPE html>
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo count($cars);
?>
</body>
</html>
```

OUTPUT: 3

Loop Through an Indexed Array: To loop through and print all the values of an indexed array, you could use a for loop, like this:

```
<!DOCTYPE html>
<html>
<body>
<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);
for($x = 0; $x < $arrlength; $x++) {
   echo $cars[$x];
   echo "<br>";
}
?>
</body>
</html>
```

OUTPUT:

Volvo
BMW
Toyota

PHP Associative Arrays: Associative arrays are arrays that use named keys that you assign to them.

There are two ways to create an associative array:

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script:

```
<!DOCTYPE html>
<html>
<body>
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
</body>
</html>
```

OUTPUT:
Peter is 35 years old.

Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

```
<!DOCTYPE html>
<html>
<body>
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
foreach($age as $x => $x_value) {
   echo "Key=" . $x . ", Value=" . $x_value;
   echo "<br>";
}
?>
```

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

OUTPUT:

Key=Peter, Value=35
Key=Ben, Value=37
Key=Joe, Value=43

PHP - Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.PHP understands multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

The dimension of an array indicates the number of indices you need to select an element.

- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

First, take a look at the following table:

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array
  (
  array("Volvo",22,18),
  array("BMW",15,13),
```

```
 array("Saab",5,2),
 array("Land Rover",17,15)
 );
```

Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.

To get access to the elements of the $cars array we must point to the two indices (row and column):

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array
 (
 array("Volvo",22,18),
 array("BMW",15,13),
 array("Saab",5,2),
 array("Land Rover",17,15)
 );

echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>

</body>
</html>
```

OUTPUT:
Volvo: In stock: 22, sold: 18.
BMW: In stock: 15, sold: 13.
Saab: In stock: 5, sold: 2.
Land Rover: In stock: 17, sold: 15.
We can also put a For loop inside another For loop to get the elements of the $cars array (we still have to point to the two indices):
OUTPUT:
Row number 0

- Volvo
- 22
- 18

Row number 1

- BMW
- 15
- 13

Row number 2

- Saab
- 5
- 2

Row number 3

- Land Rover
- 17
- 15

PHP Strings

A string is a sequence of characters, like "Hello world!".

PHP String Functions

In this chapter we will look at some commonly used functions to manipulate strings.

Get The Length of a String

The PHP strlen() function returns the length of a string.

The example below returns the length of the string "Hello world!":

```
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```

OUTPUT: 12

Count The Number of Words in a String

The PHP str_word_count() function counts the number of words in a string:

```
<!DOCTYPE html>
<html>
```

```
<body>

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

</body>
</html>
```

OUTPUT:2

Reverse a String

The PHP strrev() function reverses a string:

```
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```

OUTPUT: !dlrow olleH

Search For a Specific Text Within a String

The PHP strpos() function searches for a specific text within a string.

If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE.

The example below searches for the text "world" in the string "Hello world!":

```
<!DOCTYPE html>
<html>
<body>

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

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

OUTPUT: 6

Replace Text Within a String

The PHP str_replace() function replaces some characters with some other characters in a string.

The example below replaces the text "world" with "Dolly":

```
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```

OUTPUT: Hello Dolly!

Operators :

perators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators

PHP Arithmetic Operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result	
+	Addition	$x + $y	Sum of $x and $y	
-	Subtraction	$x - $y	Difference of $x and $y	
*	Multiplication	$x * $y	Product of $x and $y	
/	Division	$x / $y	Quotient of $x and $y	
%	Modulus	$x % $y	Remainder of $x divided by $y	
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power (Introduced in PHP 5.6)	

PHP Assignment Operators

The PHP assignment operators are used with numeric values to write a value to a variable.

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment	Same as...	Description	Show it
x = y	x = y	The left operand gets set to the value of the expression on the right	
x += y	x = x + y	Addition	
x -= y	x = x - y	Subtraction	
x *= y	x = x * y	Multiplication	
x /= y	x = x / y	Division	
x %= y	x = x % y	Modulus	

PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example	Result	Show it
==	Equal	$x == $y	Returns true if $x is equal to $y	
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type	
!=	Not equal	$x != $y	Returns true if $x is not equal to $y	
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y	
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type	
>	Greater than	$x > $y	Returns true if $x is greater than $y	
<	Less than	$x < $y	Returns true if $x is less than $y	
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y	
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y	

PHP Increment / Decrement Operators

The PHP increment operators are used to increment a variable's value.

The PHP decrement operators are used to decrement a variable's value.

Operator	Name	Description	Show it
++$x	Pre-increment	Increments $x by one, then returns $x	
$x++	Post-increment	Returns $x, then increments $x by one	

--$x Pre-decrement Decrements $x by one, then returns $x

$x-- Post-decrement Returns $x, then decrements $x by one

PHP Logical Operators

The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result	Show it
and	And	$x and $y	True if both $x and $y are true	
or	Or	$x or $y	True if either $x or $y is true	
xor	Xor	$x xor $y	True if either $x or $y is true, but not both	
&&	And	$x && $y	True if both $x and $y are true	
||	Or	$x || $y	True if either $x or $y is true	
!	Not	!$x	True if $x is not true	

PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result	Show it
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2	
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1	

PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Name	Example	Result	Show it
+	Union	$x + $y	Union of $x and $y	
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs	
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types	
!=	Inequality	$x != $y	Returns true if $x is not equal to $y	
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y	
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y	

Expressions :

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 a 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

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.

Lets give explanation for few concepts being used in POSIX regular expression. After that we will introduce you with 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.

Sr.No	Expression & Description
1	**[0-9]** It matches any decimal digit from 0 through 9.
2	**[a-z]** It matches any character from lower-case a through lowercase z.
3	**[A-Z]** It matches any character from uppercase A through uppercase Z.
4	**[a-Z]** 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.

Sr.No	Expression & Description
1	**p+** It matches any string containing at least one p.
2	**p***

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

Examples

Following examples will clear your concepts about matching characters.

Sr.No	Expression & Description
1	**[^a-zA-Z]** It matches any string not containing any of the characters ranging from a through z and A through Z.
2	**p.p** It matches any string containing p, followed by any character, in turn followed by another p.
3	**^.{2}$** It matches any string containing exactly two characters.
4	**(.*)** It matches any string enclosed within and .

5	**p(hp)*** It matches any string containing a p followed by zero or more instances of the sequence php.

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 −

Sr.No	Expression & Description
1	**[[:alpha:]]** It matches any string containing alphabetic characters aA through zZ.
2	**[[:digit:]]** It matches any string containing numerical digits 0 through 9.
3	**[[:alnum:]]** It matches any string containing alphanumeric characters aA through zZ and 0 through 9.
4	**[[:space:]]** 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 −

1	ereg() 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.
2	ereg_replace() The ereg_replace() function searches for string specified by pattern and replaces pattern with replacement if found.

3	eregi() The eregi() function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive.
4	eregi_replace() The eregi_replace() function operates exactly like ereg_replace(), except that the search for pattern in string is not case sensitive.
5	split() The split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.
6	spliti() The spliti() function operates exactly in the same manner as its sibling split(), except that it is not case sensitive.
7	sql_regcase() 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.

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.

Lets give explanation for few concepts being used in PERL regular expressions. After that we will introduce you wih regular expression related functions.

Meta characters

A meta character 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' meta character: **/([\d]+)000/**, Here **\d** will search for any string of numerical character.

Following is the list of meta characters 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

PHP's Regexp PERL Compatible Functions

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

Sr.No	Function & Description
1	preg_match() The preg_match() function searches string for pattern, returning true if pattern exists, and false otherwise.
2	preg_match_all() The preg_match_all() function matches all occurrences of pattern in string.
3	preg_replace() The preg_replace() function operates just like ereg_replace(), except that regular expressions can be used in the pattern and replacement input parameters.
4	preg_split() The preg_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern.
5	preg_grep() The preg_grep() function searches all elements of input_array, returning all elements matching the regexp pattern.
6	preg_ quote()

	Quote regular expression characters

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

- **if statement** - executes some code if one condition is true
- **if...else statement** - executes some code if a condition is true and another code if that condition is false
- **if...elseif....else statement** - executes different codes for more than two conditions
- **switch statement** - selects one of many blocks of code to be executed

PHP - The if Statement

The if statement executes some code if one condition is true.

Syntax

```
if (condition) {
   code to be executed if condition is true;
}
```

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

```
<!DOCTYPE html>
<html>
<body>
<?php
$t = date("H");

if ($t < "20") {
   echo "Have a good day!";
}
?>
 </body>
</html>
```

OUTPUT:
Have a good day!

PHP - The if…else Statement

The if....else statement executes some code if a condition is true and another code if that condition is false.

Syntax

```
if (condition) {
   code to be executed if condition is true;
} else {
   code to be executed if condition is false;
}
```

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

```
<!DOCTYPE html>
<html>
<body>
<?php
$t = date("H");
if ($t < "20") {
   echo "Have a good day!";
} else {
   echo "Have a good night!";
}
?>

</body>
</html>
```

OUTPUT: Have a good day!

PHP - The if…elseif….else Statement

The if....elseif...else statement executes different codes for more than two conditions.

Syntax

```
if (condition) {
   code to be executed if this condition is true;
} elseif (condition) {
```

```
   code to be executed if this condition is true;
} else {
   code to be executed if all conditions are false;
}
```

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

```
<!DOCTYPE html>
<html>
<body>
<?php
$t = date("H");
echo "<p>The hour (of the server) is " . $t;
echo ", and will give the following message:</p>";

if ($t < "10") {
   echo "Have a good morning!";
} elseif ($t < "20") {
   echo "Have a good day!";
} else {
   echo "Have a good night!";
}
?>

</body>
</html>
OUTPUT:
```

The hour (of the server) is 03, and will give the following message:

Have a good morning!

switch Statement

The switch statement is used to perform different actions based on different conditions.

The PHP switch Statement

Use the switch statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n) {
  case label1:
    code to be executed if n=label1;
    break;
  case label2:
    code to be executed if n=label2;
    break;
  case label3:
    code to be executed if n=label3;
    break;
  ...
  default:
    code to be executed if n is different from all labels;
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically. The **default** statement is used if no match is found.

```
<!DOCTYPE html>
<html>
<body>
<?php
$favcolor = "red";

switch ($favcolor) {
  case "red":
    echo "Your favorite color is red!";
    break;
  case "blue":
    echo "Your favorite color is blue!";
    break;
  case "green":
    echo "Your favorite color is green!";
    break;
  default:
    echo "Your favorite color is neither red, blue, nor green!";
}
?>
```

```
</body>
</html>
```
OUTPUT: Your favorite color is red!

PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- **while** - loops through a block of code as long as the specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

---

The PHP while Loop

The while loop executes a block of code as long as the specified condition is true.

Syntax

```
while (condition is true) {
   code to be executed;
}
```

The example below first sets a variable $x to 1 ($x = 1). Then, the while loop will continue to run as long as $x is less than, or equal to 5 ($x <= 5). $x will increase by 1 each time the loop runs ($x++):

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 1;
 while($x <= 5) {
  echo "The number is: $x <br>";
  $x++;
}
?>
```

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

OUTPUT:

```
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
```

The PHP do...while Loop

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax

```
do {
   code to be executed;
} while (condition is true);
```

The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 1;
do {
   echo "The number is: $x <br>";
   $x++;
} while ($x <= 5);
?>

</body>
</html>
```

OUTPUT:

```
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
```

Notice that in a do while loop the condition is tested AFTER executing the statements within the loop. This means that the do while loop would execute its statements at least once, even if the condition is false the first time.

The example below sets the $x variable to 6, then it runs the loop, **and then the condition is checked**:

```
<!DOCTYPE html>
<html>
<body>
<?php
$x = 6;
do {
  echo "The number is: $x <br>";
  $x++;
} while ($x <= 5);
?>

</body>
</html>
OUTPUT: The number is: 6
```

The PHP for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (init counter; test counter; increment counter) {
  code to be executed;
}
```

Parameters:

- *init counter*: Initialize the loop counter value
- *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment counter*: Increases the loop counter value

The example below displays the numbers from 0 to 10:

```
<!DOCTYPE html>
<html>
<body>
<?php
```

```
for ($x = 0; $x <= 10; $x++) {
  echo "The number is: $x <br>";
}
?>
</body>
</html>
```

OUTPUT:

```
The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
The number is: 6
The number is: 7
The number is: 8
The number is: 9
The number is: 10
```

The PHP foreach Loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

```
foreach ($array as $value) {
   code to be executed;
}
```

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

The following example demonstrates a loop that will output the values of the given array ($colors):

```
<!DOCTYPE html>
<html>
<body>
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
  echo "$value <br>";
```

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

OUTPUT:

red
green
blue
yellow

PHP Functions:

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

PHP User Defined Functions

Besides the built-in PHP functions, we can create our own functions.

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

A function will not execute immediately when a page loads.

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

Create a User Defined Function in PHP

A user defined function declaration starts with the word "function":

Syntax

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

Note: A function name can start with a letter or underscore (not a number).

Tip: Give the function a name that reflects what the function does!

Function names are NOT case-sensitive.

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

```
<!DOCTYPE html>
<html>
<body>
<?php
function writeMsg() {
  echo "Hello world!";
}

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

OUTPUT:

Hello world!

PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

```
<!DOCTYPE html>
<html>
<body>
<?php
function familyName($fname) {
  echo "$fname Refsnes.<br>";
}
familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
```

```
familyName("Borge");
?>
</body>
</html>
```

OUTPUT:

Jani Refsnes.
Hege Refsnes.
Stale Refsnes.
Kai Jim Refsnes.
Borge Refsnes.

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

```
<!DOCTYPE html>
<html>
<body>
<?php
function familyName($fname, $year) {
  echo "$fname Refsnes. Born in $year <br>";
}
familyName("Hege","1975");
familyName("Stale","1978");
familyName("Kai Jim","1983");
?>
</body>
</html>
```

OUTPUT:

Hege Refsnes. Born in 1975
Stale Refsnes. Born in 1978
Kai Jim Refsnes. Born in 1983

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

```
<!DOCTYPE html>
<html>
<body>
<?php
function setHeight($minheight = 50) {
  echo "The height is : $minheight <br>";
}
setHeight(350);
setHeight();
setHeight(135);
setHeight(80);
```

```
?>
</body>
</html>
```
OUTPUT:
The height is : 350
The height is : 50
The height is : 135
The height is : 80

PHP Functions - Returning values

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

```
<!DOCTYPE html>
<html>
<body>
<?php
function sum($x, $y) {
   $z = $x + $y;
   return $z;
}
echo "5 + 10 = " . sum(5,10) . "<br>";
echo "7 + 13 = " . sum(7,13) . "<br>";
echo "2 + 4 = " . sum(2,4);
?>
</body>
</html>
```
OUTPUT:
5 + 10 = 15
7 + 13 = 20
2 + 4 = 6

Reading data from web form controls like text boxes, radio buttons, lists etc,

PHP Form Handling

The PHP super globals $_GET and $_POST are used to collect form-data.

PHP - A Simple HTML Form

The example below displays a simple HTML form with two input fields and a submit button:

```
<!DOCTYPE HTML>
<html>
<body>
```

```
<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

OUTPUT:

Name:

E-mail:

Submit

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
</body>
</html>
```

The output could be something like this:

Welcome John

Your email address is john.doe@example.com

The same result could also be achieved using the HTTP GET method:

```
<!DOCTYPE HTML>
<html>
<body>
```

```
<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

Name:

E-mail:

Submit

and "welcome_get.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>

</body>
</html>
```

The code above is quite simple. However, the most important thing is missing. You need to validate form data to protect your script from malicious code.

GET vs. POST

Both GET and POST create an array (e.g. array(key => value, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as $_GET and $_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

$_GET is an array of variables passed to the current script via the URL parameters.

$_POST is an array of variables passed to the current script via the HTTP POST method.

When to use GET?

Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!

When to use POST?

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.

Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

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

Developers prefer POST for sending form data.

PHP Form Validation

Think SECURITY when processing PHP forms!

These pages will show how to process PHP forms with security in mind. Proper validation of form data is important to protect your form from hackers and spammers!

The HTML form we will be working at in these chapters, contains various input fields: required and optional text fields, radio buttons, and a submit button:

The validation rules for the form above are as follows:

Field	**Validation Rules**
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and.)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)

Gender	Required. Must select one

First we will look at the plain HTML code for the form:

Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>

Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male

The Form Element

The HTML code of the form looks like this:

<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">

When the form is submitted, the form data is sent with method="post".

What is the $_SERVER["PHP_SELF"] variable?

The $_SERVER["PHP_SELF"] is a super global variable that returns the filename of the currently executing script.

So, the $_SERVER["PHP_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

What is the htmlspecialchars() function?

The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with < and >. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Big Note on PHP Form Security

The $_SERVER["PHP_SELF"] variable can be used by hackers!

If PHP_SELF is used in your page then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute.

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.

Assume we have the following form in a page named "test_form.php":

<form method="post" action="<?php echo $_SERVER["PHP_SELF"];?>">

Now, if a user enters the normal URL in the address bar like "http://www.example.com/test_form.php", the above code will be translated to:

<form method="post" action="test_form.php">

So far, so good.

However, consider that a user enters the following URL in the address bar:

http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E

In this case, the above code will be translated to:

<form method="post" action="test_form.php/"><script>alert('hacked')</script>

This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box). This is just a simple and harmless example how the PHP_SELF variable can be exploited.

Be aware of that **any JavaScript code can be added inside the <script> tag!** A hacker can redirect the user to a file on another server, and that file can hold malicious code that can alter the global variables or submit the form to another address to save the user data, for example.

How To Avoid $_SERVER["PHP_SELF"] Exploits?

$_SERVER["PHP_SELF"] exploits can be avoided by using the htmlspecialchars() function.

The form code should look like this:

<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">

The htmlspecialchars() function converts special characters to HTML entities. Now if the user tries to exploit the PHP_SELF variable, it will result in the following output:

<form method="post"
action="test_form.php/"><script>alert('hacked')</script>">

The exploit attempt fails, and no harm is done!

Validate Form Data With PHP

The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.

When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:

<script>location.href('http://www.hacked.com')</script>

- this would not be executed, because it would be saved as HTML escaped code, like this:

<script>location.href('http://www.hacked.com')</script>

The code is now safe to be displayed on a page or inside an e-mail.

We will also do two more things when the user submits the form:

1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
2. Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).

We will name the function test_input().

Now, we can check each $_POST variable with the test_input() function, and the script looks like this:

```
<!DOCTYPE HTML>
<html>
<head>
</head>
<body>

<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
<h2>PHP Form Validation Example</h2>
<form method="post" action="<?php echohtmlspecialchars($_SERVER["PHP_SELF"]);?>">
```

```
 Name: <input type="text" name="name">
 <br><br>
 E-mail: <input type="text" name="email">
 <br><br>
 Website: <input type="text" name="website">
 <br><br>
 Comment: <textarea name="comment" rows="5"cols="40"></textarea>
 <br><br>
 Gender:
 <input type="radio" name="gender" value="female">Female
 <input type="radio" name="gender" value="male">Male
 <br><br>
 <input type="submit" name="submit" value="Submit">
</form>
<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>

</body>
</html>
```

OUTPUT:

PHP Form Validation Example

Name:

E-mail:

Website:

Comment: 

Gender: Female Male

Submit

Your Input:

ramesh
check@gmail.com
www.cmrec.ac.in
welcome
male

Notice that at the start of the script, we check whether the form has been submitted using $_SERVER["REQUEST_METHOD"]. If the REQUEST_METHOD is POST, then the form has been submitted - and it should be validated. If it has not been submitted, skip the validation and display a blank form.

However, in the example above, all input fields are optional. The script works fine even if the user does not enter any data.

The next step is to make input fields required and create error messages if needed.

PHP Complete FORM example

```
<!DOCTYPE HTML>
<html>
<head>
<style>
.error {color: #FF0000;}
</style>
</head>
<body>

<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";
```

```php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
    // check if name only contains letters and whitespace
    if (!preg_match("/^[a-zA-Z ]*$/",$name)) {
      $nameErr = "Only letters and white space allowed";
    }
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
    // check if e-mail address is well-formed
    if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
      $emailErr = "Invalid email format";
    }
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
    // check if URL address syntax is valid (this regular expression also
allows dashes in the URL)
    if (!preg_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-
9+&@#\/%?=~_|!:,.;]*[-a-z0-9+&@#\/%=~_|]/i",$website)) {
      $websiteErr = "Invalid URL";
    }
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
```

```
function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>

<h2>PHP Form Validation Example</h2>
<p><span class="error">* required field.</span></p>
<form method="post" action="<?php echohtmlspecialchars($_SERVER["PHP_SELF"
]);?>">
  Name: <input type="text" name="name" value="<?php echo$name;?>">
  <span class="error">* <?php echo $nameErr;?></span>
  <br><br>
  E-mail: <input type="text" name="email" value="<?phpecho $email;?>">
  <span class="error">* <?php echo $emailErr;?></span>
  <br><br>
  Website: <input type="text" name="website" value="<?phpecho $website;?>"
>
  <span class="error"><?php echo $websiteErr;?></span>
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"><?php echo $comment
;?></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" <?php if(isset($gender) &&
$gender=="female") echo "checked";?>value="female">Female
  <input type="radio" name="gender" <?php if(isset($gender) &&
$gender=="male") echo "checked";?>value="male">Male
  <span class="error">* <?php echo $genderErr;?></span>
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>
```

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

OUTPUT:

PHP Form Validation Example

* required field.

Name: ramesh *

E-mail: check@g *

Website: www.cr

Comment: 

Gender: Female Male *

Submit

Your Input:

ramesh
check@gmail.com
www.cmrec.ac.in
ss
male

Handling sessions and cookies:

An **HTTP cookie** (also called **web cookie**, **Internet cookie**, **browser cookie** or simply **cookie**) is a small piece of data sent from a website and stored on the user's computer by the user's web browser while the user is browsing. Cookies were designed to be a reliable mechanism for websites to remember stateful

information (such as items added in the shopping cart in an online store) or to record the user's browsing activity (including clicking particular buttons, logging in, or recording which pages were visited in the past). They can also be used to remember arbitrary pieces of information that the user previously entered into form fields such as names, addresses, passwords, and credit card numbers.

A cookie is often used to identify a user.

What is a Cookie?

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the setcookie() function.

Syntax

setcookie(*name, value, expire, path, domain, secure, httponly*);

Only the *name* parameter is required. All other parameters are optional.

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

EXAMPLE:

```
<!DOCTYPE html>
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>
<?php
if(!isset($_COOKIE[$cookie_name])) {
```

```
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
<p><strong>Note:</strong> You might have to reload the page to see the value of the cookie.</p>
</body>
</html>
```

OUTPUT:

Cookie 'user' is set!
Value is: John Doe

Note: You might have to reload the page to see the value of the cookie.

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

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the setcookie() function:

Example

```
<!DOCTYPE html>
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
<html>
<body>
<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

<p><strong>Note:</strong> You might have to reload the page to see the new value of the cookie.</p>
</body>
</html>
```

OUTPUT:
Cookie 'user' is set!
Value is: John Doe

Note: You might have to reload the page to see the new value of the cookie.

Delete a Cookie

To delete a cookie, use the setcookie() function with an expiration date in the past:

Example

```
<!DOCTYPE html>
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>
<?php
echo "Cookie 'user' is deleted.";
?>

</body>
</html>
```

OUTPUT: Cookie 'user' is deleted.

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable:

Example

```
<!DOCTYPE html>
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>
<?php
if(count($_COOKIE) > 0) {
   echo "Cookies are enabled.";
} else {
   echo "Cookies are disabled.";
```

```
}
?>
</body>
</html>
```
OUTPUT: Cookies are enabled.

PHP Sessions

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.

What is a PHP Session?

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

Tip: If you need a permanent storage, you may want to store the data in a database.

Start a PHP Session

A session is started with the session_start() function.

Session variables are set with the PHP global variable: $_SESSION.

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

Example

```php
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

OUTPUT: Session variables are set.

Note: The session_start() function must be the very first thing in your document. Before any HTML tags.

Get PHP Session Variable Values

Next, we create another page called "demo_session2.php". From this page, we will access the session information we set on the first page ("demo_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session_start()).

Also notice that all session variable values are stored in the global $_SESSION variable:

Example

```php
<?php
session_start();
?>
<!DOCTYPE html>
<html>
```

```
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . ".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>
```

OUTPUT:

Favorite color is green.
Favorite animal is cat.

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

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
print_r($_SESSION);
?>

</body>
</html>
```

OUTPUT: Array ([favcolor] => green [favanimal] => cat)

How does it work? How does it know it's me?

Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12. Then, when a session is opened on another page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

Modify a PHP Session Variable

To change a session variable, just overwrite it:

```php
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

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

</body>
</html>
```

OUTPUT: Array ([favcolor] => yellow [favanimal] => cat)

Destroy a PHP Session

To remove all global session variables and destroy the session, use session_unset() and session_destroy():

```php
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();

echo "All session variables are now removed, and the session is destroyed."
?>

</body>
</html>
```

OUTPUT:
All session variables are now removed, and the session is destroyed.

File Handling in PHP:

File handling is an important part of any web application. You often need to open and process a file for different tasks.

PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

Be careful when manipulating files!

When you are manipulating files you must be very careful.

You can do a lot of damage if you do something wrong. Common errors are: editing the wrong file, filling a hard-drive with garbage data, and deleting the content of a file by accident.

PHP readfile() Function

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called "webdictionary.txt", stored on the server, that looks like this:

AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language

The PHP code to read the file and write it to the output buffer is as follows (the readfile() function returns the number of bytes read on success):

Example

```
<?php
echo readfile("webdictionary.txt");
?>
```

Output:

AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper Text Markup Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG = Scalable Vector Graphics XML = EXtensible Markup Language236

The readfile() function is useful if all you want to do is open up a file and read its contents.

PHP File Open/Read/Close

PHP Open File - fopen()

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.

We will use the text file, "webdictionary.txt", during the lessons:

AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language

The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

Example

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

```
echo fread($myfile,filesize("webdictionary.txt"));
fclose($myfile);
?>
```

Tip: The fread() and the fclose() functions will be explained below.

The file may be opened in one of the following modes:

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

	Returns FALSE and an error if file already exists

PHP Read File - fread()

The fread() function reads from an open file.

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

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

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

PHP Close File - fclose()

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

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

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

PHP Read Single Line - fgets()

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

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

Example

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

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

PHP Read Single Character - fgetc()

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

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

Example

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