

PHP

Prithviraj Mohanty
Asst.Prof. Dept of CSIT





Introduction

- The PHP **Hypertext Preprocessor** (PHP) is a programming language that allows web developers to create dynamic content that interacts with databases.
- PHP is basically used for developing web based software applications.
- 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 started out as a small open source project that evolved as more and more people found out how useful it was. [Rasmus Lerdorf](#) unleashed the first version of PHP way back in 1994.



What is 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 free to download and use

PHP is an amazing and popular language!

- It is powerful enough to be at the core of the biggest blogging system on the web (WordPress)!
- It is deep enough to run the largest social network (Facebook)!
- It is also easy enough to be a beginner's first server side language!



What is a PHP File?

- 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"
- **Note:** With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

What Can PHP Do?

- 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



Why PHP?

- 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
- PHP is free. Download it from the official PHP resource:
www.php.net
- PHP is easy to learn and runs efficiently on the server side

Characteristics of PHP

Five important characteristics make PHP's practical nature possible –

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity



How to work with PHP?

To start using PHP, you can:

- Find a web host with PHP and MySQL support
- Install a web server on your own PC, and then install PHP and MySQL

Use a Web Host With PHP Support

- If your server has activated support for PHP you do not need to do anything.
- Just create some .php files, place them in your web directory, and the server will automatically parse them for you.
- You do not need to compile anything or install any extra tools.
- Because PHP is free, most web hosts offer PHP support.

Set Up PHP on Your Own PC

However, if your server does not support PHP, you must:

- install a web server
- install PHP
- install a database, such as MySQL
- The official PHP website (PHP.net) has installation instructions for PHP:
<http://php.net/manual/en/install.php>



PHP Syntax

- A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

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.



First PHP Program

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h1>My first PHP page</h1>
```

```
<?php
```

```
echo "Hello World!";
```

```
?>
```

```
</body>
```

```
</html>
```

My first PHP page
Hello World!

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



Comments in PHP

- A comment in PHP code is a line that is not read/executed as part of the program.
- Its only purpose is to be read by someone who is looking at the code.

Comments can be used to:

- Let others understand what you are doing
- Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code

Example:

```
<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>
```



PHP Case Sensitivity

- In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive.
- In the example below, all three echo statements below are **legal** (and equal):

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

```
<?php
```

```
ECHO "Hello World!<br>";
```

```
echo "Hello World!<br>";
```

```
EcHo "Hello World!<br>";
```

```
?>
```

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

However; all variable names are **case-sensitive**.

```
<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>
```



PHP Variables

- Variables are "containers" for storing information.

Creating (Declaring) PHP Variables

- In PHP, a variable starts with the \$ sign, followed by the name of the variable:

Example

```
<?php  
$txt = "Hello world!";  
$x = 5;  
$y = 10.5;  
?>
```

Note: When you assign a text value to a variable, put quotes around the value.

Note: Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

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)



Output Variables

- The PHP echo statement is often used to output data to the screen.

Example-1

```
<?php  
$txt = "my India";  
echo "I love $txt!";  
?>
```

Example-2

```
<?php  
$x = 5;  
$y = 4;  
echo $x + $y;  
?>
```

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

Example-1

```
<?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>";
?>
```

Example-2

```
<?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>";
?>
```

Note: 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):

Example

```
<?php
$x = 5;
$y = 10;
function myTest() {
    global $x, $y;
    $y = $x + $y;
}
myTest();
echo $y; // outputs 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:

Example

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



PHP echo and print Statements

- echo and print are more or less the same. They are both used to output data to the screen.

The differences are small:

- echo has no return value while print has a return value of 1, so it can be used in expressions.
- echo can take multiple parameters (although such usage is rare) while print can take one argument.
- echo is marginally faster than print.

Example-1(echo)

```
<?php  
echo "<h2>PHP is Fun!</h2>";  
echo "Hello world!<br>";  
echo "I'm about to learn PHP!<br>";  
?>
```

Example-2(print)

```
<?php  
print "<h2>PHP is Fun!</h2>";  
print "Hello world!<br>";  
print "I'm about to learn PHP!";  
?>
```



PHP 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:

Example

```
<?php
$x = "Hello world!";
$y = 'Hello world!';
echo $x;
echo "<br>";
echo $y;
?>
```




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)

Example

```
<?php
$x = 5985;
var_dump($x);
?>
```

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:

Example

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



PHP Boolean

- A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
```

```
$y = false;
```

Booleans are often used in conditional testing.

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.

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:

Example

```
<?php
```

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

```
var_dump($cars);
```

```
?>
```

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:

Example

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

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:

Example

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



PHP String Functions

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!":

Example

```
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

Count The Number of Words in a String

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

Example

```
<?php
echo str_word_count("Hello world!"); //
outputs 2
?>
```

Reverse a String

- The PHP `strrev()` function reverses a string:

Example

```
<?php
echo strrev("Hello world!"); // outputs !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!":

Example

```
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```



PHP Constants

- A constant is an identifier (name) for a simple value. The value cannot be changed during the script.
- A valid constant name starts with a letter or underscore (no \$ sign before the constant name).

Create a PHP Constant

To create a constant, use the `define()` function.

Syntax

`define(name, value, case-insensitive)`

Parameters:

- **name:** Specifies the name of the constant
- **value:** Specifies the value of the constant
- **case-insensitive:** Specifies whether the constant name should be case-insensitive. Default is false

Example-1

```
<?php  
define("GREETING", "Welcome to  
    iter.ac.in!");  
echo GREETING;  
?>
```

Example-2

```
<?php  
define("GREETING", "Welcome to  
    iter.ac.in!", true);  
echo greeting;  
?>
```

Note: Unlike variables, constants are automatically global and can be used across the entire script.



PHP Operators

- Operators 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 String Operators

- PHP has two operators that are specially designed for strings.
 - . Concatenation \$txt1 . \$txt2
Concatenation of \$txt1 and \$txt2
 - .= Concatenation assignment
\$txt1 .= \$txt2 Appends \$txt2 to \$txt1



PHP Arithmetic Operators

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 Array Operators

- The PHP array operators are used to compare arrays.

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



PHP Conditional Statements

- Conditional statements are used to perform different actions based on different conditions.
- 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 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;  
}
```

Example:

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



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);
```

Example:

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



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/decrement 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/decrement counter*: Increases/decreases the loop counter value

Example:

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



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.

Example:

```
<html>  
<body>  
<?php  
$colors = array("red", "green",  
    "blue", "yellow");  
foreach ($colors as $value) {  
    echo "$value <br>";  
}  
?>  
</body>  
</html>
```



PHP Functions

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

```

Example:

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

- **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.



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.

```
<?php
function familyName($fname) {
    echo "$fname Refsnes.<br>";
}
familyName("Rahul");
familyName("Raj");
familyName("Anand");
familyName("Ayush");
familyName("Arya");
?>
```

Example:2

```
<html>
<body>
<?php
function familyName($fname, $year) {
    echo "$fname Refsnes. Born in $year
    <br>";
}
familyName("Rahul","1975");
familyName("Ayush","1978");
familyName("Raj","1983");
?>
</body>
</html>
```




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:

Example

```
<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
}
setHeight(350);
setHeight(); // will use the default value of
50
setHeight(135);
setHeight(80);
?>
```

PHP Functions - Returning values

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

Example

```
<?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);
?>
```



PHP Arrays

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");
```

- The index can be assigned manually:

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

Example:

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



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:

Example

```
<?php
$scars = array("Volvo", "BMW",
    "Toyota");
echo count($scars);
?>
```

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:

Example

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



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";
```

Example:

```
<html>
```

```
<body>
```

```
<?php
```

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

```
echo "Peter is " . $age['Peter'] .  
      " years old.";
```

```
?>
```

```
</body>
```

```
</html>
```



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:

Example

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



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

Example:

```
<?php
$cars = array (
    array("Volvo",22,18),
    array("BMW",15,13),
    array("Saab",5,2),
    array("Land Rover",17,15)
);
for ($row = 0; $row < 4; $row++) {
    echo "<p><b>Row number $row</b></p>";
    echo "<ul>";
    for ($col = 0; $col < 3; $col++) {
        echo "<li>".$cars[$row][$col]."</li>";
    }
    echo "</ul>";
}
?>
```



PHP: Sorting Arrays

- We will go through the following PHP array sort functions:

sort() - sort arrays in ascending order

rsort() - sort arrays in descending order

asort() - sort associative arrays in ascending order, according to the value

ksort() - sort associative arrays in ascending order, according to the key

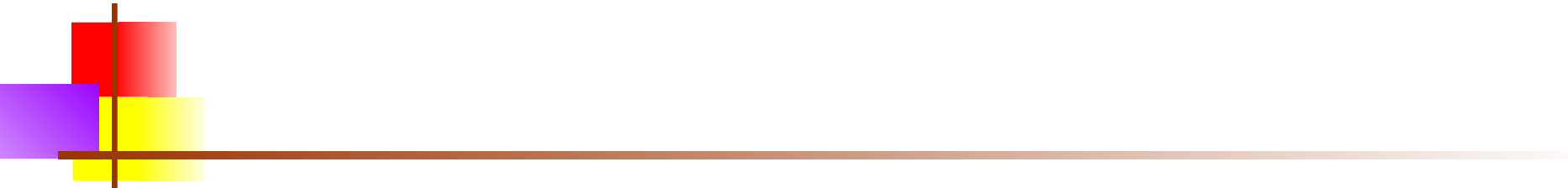
arsort() - sort associative arrays in descending order, according to the value

krsort() - sort associative arrays in descending order, according to the key

Example:

```
<?php
$scars = array("Volvo", "BMW",
               "Toyota");
sort($scars);
$clength = count($scars);
for($x = 0; $x < $clength; $x++) {
    echo $scars[$x];
    echo "<br>";
}
?>
```

The above example sorts the elements of the \$scars array in ascending alphabetical order:

- 
- The following example sorts the elements of the \$numbers array in ascending numerical order:

Example:

```
<?php
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);
$arlength = count($numbers);
for($x = 0; $x < $arlength; $x++) {
    echo $numbers[$x];
    echo "<br>";
}
?>
```

Sort Array in Descending Order – rsort()

- The following example sorts the elements of the \$numbers array in descending numerical order:

Example

```
<?php
$numbers = array(4, 6, 2, 22, 11);
rsort($numbers);
?>
```



Sort Array (Ascending Order), According to Value – asort()

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

Sort Array (Ascending Order), According to Key – ksort()

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



Sort Array (Descending Order), According to Value - arsort()

```
<html> <body>

<?php

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

</body> </html>
```

Sort Array (Descending Order), According to Key – krsort()

```
<html> <body>

<?php

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

</body> </html>
```



PHP Global Variables - Superglobals

- Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- The PHP superglobal variables are:
 - `$GLOBALS`
 - `$_SERVER`
 - `$_REQUEST`
 - `$_POST`
 - `$_GET`
 - `$_FILES`
 - `$_ENV`
 - `$_COOKIE`
 - `$_SESSION`

Example:

```
<html><body>
<?php
$x = 75;
$y = 25;
function addition() {
    $GLOBALS['z'] = $GLOBALS['x'] +
    $GLOBALS['y'];
}
addition();
echo $z;
?>
</body></html>
```

- **Note:** PHP stores all global variables in an array called `$GLOBALS[index]`. The index holds the name of the variable.



PHP: Form Handling

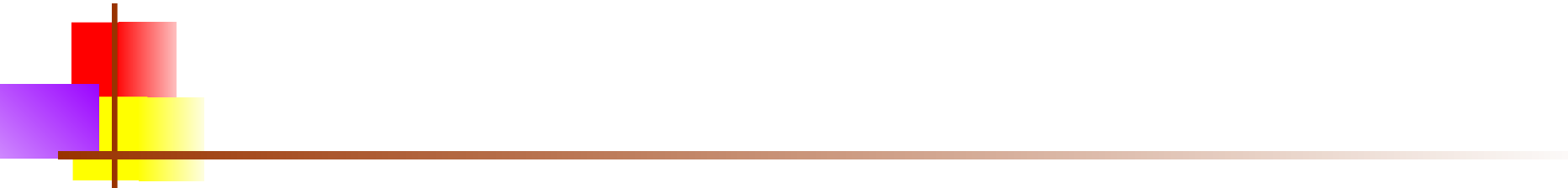
- The PHP superglobals `$_GET` and `$_POST` are used to collect form-data.
- The example below displays a simple HTML form with two input fields and a submit button:

Example:

```
<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>
```

- After clicking 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 same result could also be achieved using the HTTP GET method:

Example

```
<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>
```

- “**welcome_get.php**” looks like this:

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

Note: This page does not contain any form validation, it just shows how you can send and retrieve form data.



GET vs. POST

- 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.
- **Note:** Developers prefer POST for sending form data.

PHP Form Validation

- Think SECURITY when processing PHP forms!
- Now we will see how to process PHP forms with security in mind. Proper validation of form data is important to protect your form from hackers and spammers!



PHP Form Validation Example

PHP Form Validation Example

* required field.

Name: *

E-mail: *

Website:

Comment:

Gender: ☐ Female ☐ Male *

Your Input:

The validation rules for the form above are as follows:

- **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



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.



Validate Form Data With PHP

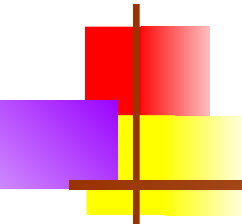
- The first thing we will do is to pass all variables through PHP's **htmlspecialchars()** function.
 - When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:
`<script>location.href('http://www.hacked.com')</script>`
 - this would not be executed, because 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:
 - Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP **trim()** function)
 - 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()**.



Example:

```
<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 echo
    htmlspecialchars($_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

Your Input:

prithviraj
prithviraj_cs@rediffmail.com
WWW.iter.ac.in
Wel come to ITER
male



PHP Forms - Required Fields

- From the validation rules table on the previous page, we see that the "Name", "E-mail", and "Gender" fields are required. These fields cannot be empty and must be filled out in the HTML form.
- In the following code we have added some new variables: **\$nameErr, \$emailErr, \$genderErr, and \$websiteErr.**
- These error variables will hold error messages for the required fields. We have also added an if else statement for each \$_POST variable.
- This checks if the \$_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test_input() function:



PHP - Display The Error Messages

- When in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields)

```
<?php
```

```
// define variables and set to empty values
```

```
$nameErr = $emailErr = $genderErr = $websiteErr =  
"";
```

```
$name = $email = $gender = $comment = $website =  
"";
```

```
if ($_SERVER["REQUEST_METHOD"] == "POST")  
{
```

```
    if (empty($_POST["name"])) {
```

```
        $nameErr = "Name is required"; } else {
```

```
    $name = test_input($_POST["name"]); }
```

```
    if (empty($_POST["email"])) {
```

```
        $emailErr = "Email is required";
```

```
    } else {
```

```
        $email = test_input($_POST["email"]); }
```

```
    if (empty($_POST["website"])) {
```

```
        $website = "";
```

```
    } else {
```

```
        $website = test_input($_POST["website"]);
```

```
    }
```

```
    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"]);
```

```
    }}
```

```
?>
```



PHP Forms - Validate E-mail and URL

PHP - Validate Name

- The code below shows a simple way to check if the name field only contains letters and whitespace. If the value of the name field is not valid, then store an error message:

```
$name = test_input($_POST["name"]);  
if (!preg_match("/^[a-zA-Z ]*$/",$name))  
{  
    $nameErr = "Only letters and white  
    space allowed";  
}
```

- The **preg_match()** function searches a string for pattern, returning true if the pattern exists, and false otherwise.

PHP - Validate E-mail

- The easiest and safest way to check whether an email address is well-formed is to use PHP's **filter_var()** function.
- In the code below, if the e-mail address is not well-formed, then store an error message:

```
$email =  
    test_input($_POST["email"]);  
if (!filter_var($email,  
    FILTER_VALIDATE_EMAIL)) {  
    $emailErr = "Invalid email format";  
}
```




PHP - Validate URL

- The code below shows a way to check if a URL address syntax is valid (this regular expression also allows dashes in the URL). If the URL address syntax is not valid, then store an error message:

```
$website = test_input($_POST["website"]);  
if (!preg_match("/^b(?:(:https?|ftp):\\V\\V|www\\.)([-a-z0-9+&@#\\/  
    %?~=~_|!:,.;]*[-a-z0-9+&@#\\V/%=~_|]/i",$website)) {  
    $websiteErr = "Invalid URL";  
}
```



PHP - Keep The Values in The Form

- To show the values in the input fields after the user hits the submit button, a little PHP script can be added inside the value attribute of the following input fields: name, email, and website.
- In the comment textarea field, we put the script between the `<textarea>` and `</textarea>` tags.
- The little script outputs the value of the `$name`, `$email`, `$website`, and `$comment` variables.
- To show which radio button was checked, you must manipulate the checked attribute (not the value attribute for radio buttons):

Name: `<input type="text" name="name" value="<?php echo $name;?>">`

E-mail: `<input type="text" name="email" value="<?php echo $email;?>">`

Website: `<input type="text" name="website" value="<?php echo $website;?>">`

Comment: `<textarea name="comment" rows="5" cols="40"><?php echo $comment;?></textarea>`

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`