PHP

PHP Introduction

> PHP is a recursive acronym for “PHP: Hypertext Preprocessor”

> It is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML.

PHP Introduction

> PHP is a server-side scripting language

> PHP scripts are executed on the server

> PHP is an interpreted language, i.e. there is no need for compilation.

> PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)

> PHP is open source software

> PHP is an object-oriented language

PHP Features

> PHP runs on different platforms (Windows, Linux, Unix, etc.)

> PHP is compatible with almost all servers used today (Apache, IIS, etc.)

> PHP is easy to learn and runs efficiently on the server side

PHP Features

> Script written in PHP executes much faster than those scripts written in other languages such as JSP & ASP.

> PHP pages contain HTML with embedded code that does "something" (like in the next slide, it outputs "Hi, I'm a PHP script!").

PHP Coding

The PHP code is enclosed in special start and end processing instructions **<?php** and **?>** that allow you to jump into and out of "PHP mode."

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>Example</title>
    </head>
    <body>

        <?php
            echo "Hi, I'm a PHP script!";
        ?>

    </body>
</html>
```

PHP code

> PHP code is executed on the server, generating HTML which is then sent to the client.

> The client would receive the results of running that script, but would not know what the underlying code was.

PHP Running Style

On Server

```
<html>
<head> <title>Welcome</title> </head>
<body>
<?
  echo "Hello";
  print "<br />";
  echo "<b>I'm here..</b>";
?>
</body>
</html>
```

```
<html>
<head> <title>Welcome</title> </head>
<body>
Hello<br /><b>I'm here..</b></body>

</html>
```

PHP Getting Started

> To install PHP, suggest to install AMP (Apache, MySQL, PHP) software stack. It is available for all operating systems.

WAMP for Windows

LAMP for Linux

MAMP for Mac

SAMP for Solaris

FAMP for FreeBSD

XAMPP (Cross, Apache, MySQL, PHP, Perl) for Cross Platform: It includes some other components too such as FileZilla, OpenSSL, Webalizer, Mercury Mail etc.

PHP Example- Hello World

```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <?php echo '<p>Hello World</p>'; ?>
 </body>
</html>
```

Above is the PHP source code.

PHP Hello World

It renders as HTML that looks like this:

```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <p>Hello World</p>
 </body>
</html>
```

PHP- Print Statement

- 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. Multiline print

```
<?php
        echo "Hello by PHP echo
        this is multi line
        text printed by
        PHP echo statement
        ";
?>
```

2. Escape character print

```
<?php
        echo "Hello escape \"sequence\" characters";
?>
```

3. Print Variable Value

```
<?php
        $msg="Hello PHP";
        echo "Message is: $msg";
?>
```

PHP print

1. Multiline print

```
<?php
        print "Hello by PHP echo
        this is multi line
        text printed by
        PHP Print statement
        ";
?>
```

2. Escape character print

```
<?php
        print "Hello escape \"sequence\" characters";
?>
```

3. Print Variable Value

```
<?php
        $msg="Hello PHP";
        print "Message is: $msg";
?>
```

PHP Comments

// to make a single-line comment

or

*/** and ***/* to make a large comment block.

Unix Shell style single line comment

```
<html>
<body>

<?php
//This is a comment

/*
This is
a comment
block
*/
?>

</body>
</html>
```

PHP Variables

> Variables are used for storing values, like text strings, numbers or arrays.

> When a variable is declared, it can be used over and over again in your script.

> All variables in PHP start with a $ sign symbol.

> The correct way of declaring a variable in PHP:

```
$var_name = value;
```

PHP Variables

```php
<?php
$txt="Hello World!";
$x=16;
?>
```

> In PHP, a variable does not need to be declared before adding a value to it.

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

PHP Variable Rules

> A variable name must start with a letter or an underscore "_“ not a number

> A variable name can only contain alpha-numeric characters, underscores (a-z, A-Z, 0-9, and _)

> A variable name should not contain spaces.

> If a variable name is more than one word, it should be separated with an underscore ($my_string) or with capitalization ($myString)

> Case Sensitive

>Loosely Type language: PHP automatically converts the variable to the correct data type, depending on its value.

Examples

1. <?php $str="hello string“; $x=200; $y=44.6;

echo "string is: $str **
**"; echo "integer is: $x **
**";

echo "float is: $y **
**";

?>

2. **<?php** $x=5; $y=6; $z=$x+$y; echo $z; **?>**

3. <?php $color="red"; echo "My car is " . $color . "**
**";

echo "My house is " . $COLOR . "**
**"; // error

echo "My boat is " . $coLOR . "**
"; // error **?>

4. <?php

$a="hello"; //letter (valid)

$_b="hello"; //underscore (valid)

echo "$a **
** $_b"; **?>**

PHP Concatenation

> The concatenation operator (.) is used to put two string values together.

> To concatenate two string variables together, use the concatenation operator:

```
<?php
$txt1="Hello World!";
$txt2="What a nice day!";
echo $txt1 . " " . $txt2;
?>
```

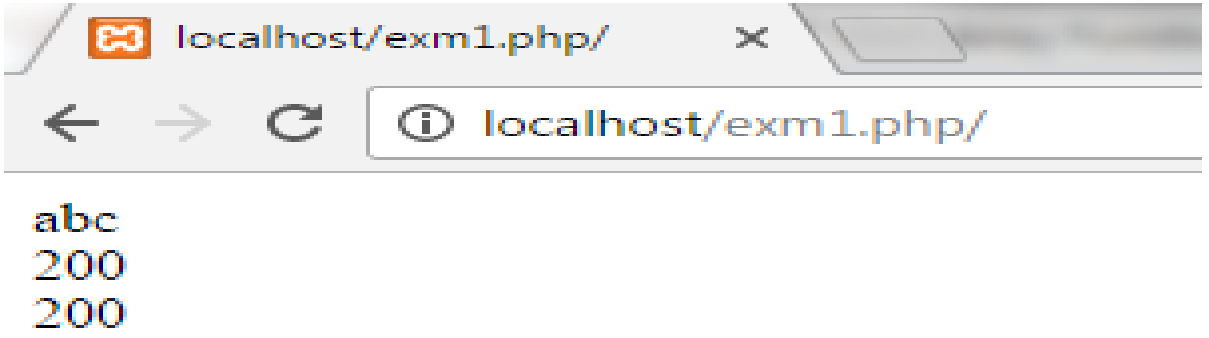
PHP $ and $$ Variables

>The $var (single dollar) is a normal variable with the name var that stores any value like string, integer, float, etc.

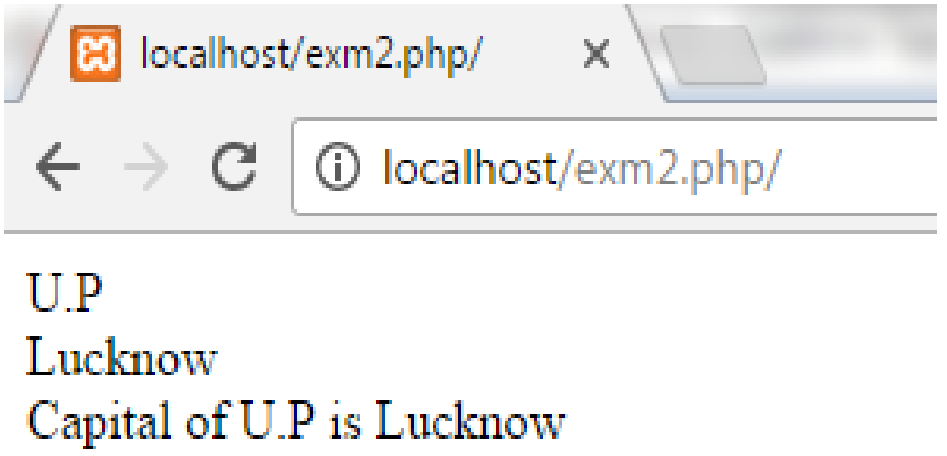
>The $$var (double dollar) is a reference variable that stores the value of the $variable inside it.

Example

```php
<?php
$x = "abc";
$$x = 200;
echo $x."<br/>";
echo $$x."<br/>";
echo $abc;
?>
```




```php
<?php
 $x="U.P";
$$x="Lucknow";
echo $x. "<br>";
echo $$x. "<br>";
echo "Capital of $x is " . $$x;
?>
```



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 Data Types

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

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

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

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

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

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

```php
<?php
class Car {
    function Car() {
        $this->model = "VW";
    }
}

// create an object
$herbie = new Car();

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

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

PHP Constants

> A constant is an identifier for a value that cannot change during the course of a script

> Constants can be user defined, or you can use some of the predefined constants that PHP always has available

```
<?
define("MYCONSTANT", "This is a test of defining constants.");
echo MYCONSTANT;
?>
```

PHP Operators

> Operators are used to operate on values. There are four classifications of operators:

- > Arithmetic
- > Assignment
- > Comparison
- > Logical

PHP Operators

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

PHP Operators

Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
.=	x.=y	x=x.y
%=	x%=y	x=x%y

PHP Operators

Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
<>	is not equal	5<>8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

PHP Operators

Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

PHP Conditional Statements

> To perform different actions for different decisions.

> Use conditional statements in code to do this.

> In PHP have the following conditional statements.

PHP Conditional Statements

> **if** statement - use this statement to execute some code only if a specified condition is true

> **if...else** statement - use this statement to execute some code if a condition is true and another code if the condition is false

> **if...elseif....else** statement - use this statement to select one of several blocks of code to be executed

> **switch** statement - use this statement to select one of many blocks of code to be executed

PHP Conditional Statements

> The following example will output "Have a nice weekend!" if the current day is Friday:

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri") echo "Have a nice weekend!";
?>

</body>
</html>
```

PHP Conditional Statements

> Use the **if....else** statement to execute some code if a condition is true and another code if a condition is false.

```
<html>
<body>

<?php
$d=date("D");
if  ($d=="Fri")
   echo "Have a nice weekend!";
else
   echo "Have a nice day!";
?>

</body>
</html>
```

PHP Conditional Statements

> If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces **{ }**

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Hello!<br />";
  echo "Have a nice weekend!";
  echo "See you on Monday!";
  }
?>

</body>
</html>
```

PHP Conditional Statements

> The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?>

</body>
</html>
```

PHP Conditional Statements

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

```
switch (n)
{
case label1:
  code to be executed if n=label1;
  break;
case label2:
  code to be executed if n=label2;
  break;
default:
  code to be executed if n is different from both label1 and label2;
}
```

PHP Conditional Statements

> For switch, 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.

PHP Conditional Statements

```
<html>
<body>

<?php
switch ($x)
{
case 1:
   echo "Number 1";
   break;
case 2:
   echo "Number 2";
   break;
case 3:
   echo "Number 3";
   break;
default:
   echo "No number between 1 and 3";
}
?>

</body>
</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>
```

$_GET method Example

```
<html>     <body>
<form method="get" >
  Name: <input type="number" name="number">
  <input type="submit" value="odd or even">
</form>

<?php
if ($_GET)
{
    // collect value of input field
    $no = $_GET['number'];
    if (empty($no)) {                echo "Empty, Try once again!";     }

        else {            if (($no%2)==0) {         echo $no." is Even!";     }
                                    else {         echo $no." is Odd";          }
                }

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

$_POST method Example

```
<html>
<body>
<form method="post" >
  Name: <input type="number" name="number">
  <input type="submit" value="odd or even">
</form>
<?php
if ($_POST)
{
    // collect value of input field
    $no = $_POST['number'];
    if (empty($no)) {              echo "Empty, Try once again!";     }
       else {
                    if (($no%2)==0) {              echo $no." is Even!";    }
                   else {      echo $no." is Odd";                  }
                }
}
?>
</body> </html>
```

PHP Loops

> **while** - loops through a block of code while a specified condition is true

> **do...while** - loops through a block of code once, and then repeats the loop as long as a 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

PHP Loops - while

> The while loop executes a block of code while a condition is true. The example below defines a loop that starts with i=1.

> The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<html>
<body>

<?php
$i=1;
while($i<=5)
  {
  echo "The number is " . $i . "<br />";
  $i++;
  }
?>

</body>
</html>
```

Output:

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

PHP Loops – do ... while

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

> The next example defines a loop that starts with i=1. It will then increment i with 1, and write some output. Then the condition is checked, and the loop will continue to run as long as i is less than, or equal to 5:

PHP Loops – do ... while

```
<html>
<body>

<?php
$i=1;
do
  {
  $i++;
  echo "The number is " . $i . "<br />";
  }
while ($i<=5);
?>

</body>
</html>
```

Output:

```
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
```

PHP Loops - for

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

Syntax

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

PHP Loops - for

Parameters:

- **init**: Mostly used to set a counter (but can be any code to be executed once at the beginning of the loop)
- **condition**: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- **increment**: Mostly used to increment a counter (but can be any code to be executed at the end of the loop)

PHP Loops - for

> The example below defines a loop that starts with i=1.

> The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<html>
<body>

<?php
for (i=1; i<=5; i++)
  {
  echo "The number is " . i . "<br />";
  }
?>

</body>
</html>
```

Output:

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

Example

```
<html>
<body>

<?php
$j=2;
for ($i = 1; $i <= 10; $i++) {
  echo  i. “X”.$y. “=” .$i*$j .<br>";
}
?>

</body>
</html>
```

PHP Loops - foreach

```
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) - so on the next loop iteration, you'll be looking at the next array value.

PHP Loops - foreach

> The following example demonstrates a loop that will print the values of the given array:

```
<html>
<body>

<?php
$x=array("one","two","three");
foreach ($x as $value)
   {
   echo $value . "<br />";
   }
?>

</body>
</html>
```

Output:

```
one
two
three
```

PHP Arrays

> An array variable is a storage area holding a number or text.

> An array is a special variable, which can store multiple values in one single variable.

PHP Arrays

>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="Saab";
$cars2="Volvo";
$cars3="BMW";
```

PHP Arrays

> An array can hold all variable values under a single name.

> And can access the values by referring to the array name.

> Each element in the array has its own index so that it can be easily accessed.

PHP Arrays

In PHP, there are three kind of arrays:

- **Numeric array** - An array with a numeric index
- **Associative array** - An array where each ID key is associated with a value
- **Multidimensional array** - An array containing one or more arrays

PHP Numeric Arrays

> A numeric array stores each array element with a numeric index.

> There are two methods to create a numeric array.

PHP Numeric Arrays

> In the following example the index is automatically assigned (the index starts at 0):

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

> In the following example, assign the index manually:

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

PHP Numeric Arrays

> In the following example, access the variable values by referring to the array name and index:

```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
?>
```

The code above will output:

```
Saab and Volvo are Swedish cars.
```

PHP Associative Arrays

> With an associative array, each ID key is associated with a value.

> When storing data about specific named values, a numerical array is not always the best way to do it.

> With associative arrays we can use the values as keys and assign values to them.

PHP Associative Arrays

> In this example, Use an array to assign ages to the different persons:

```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

> This example is the same as the one above, but shows a different way of creating the array:

```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

PHP Associative Arrays

The ID keys can be used in a script:

```
<?php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";

echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

The code above will output:

```
Peter is 32 years old.
```

PHP Multidimensional Arrays

> In a multidimensional array, each element in the main array can also be an array.

> And each element in the sub-array can be an array, and so on.

PHP Multidimensional Arrays

In this example we create a multidimensional array, with automatically assigned ID keys:

```
$families = array
  (
  "Griffin"=>array
  (
  "Peter",
  "Lois",
  "Megan"
  ),
  "Quagmire"=>array
  (
  "Glenn"
  ),
  "Brown"=>array
  (
  "Cleveland",
  "Loretta",
  "Junior"
  )
  );
```

PHP Multidimensional Arrays

The array above would look like this if written to the output:

```
Array
(
[Griffin] => Array
   (
   [0] => Peter
   [1] => Lois
   [2] => Megan
   )
[Quagmire] => Array
   (
   [0] => Glenn
   )
[Brown] => Array
   (
   [0] => Cleveland
   [1] => Loretta
   [2] => Junior
   )
)
```

PHP Multidimensional Arrays

Lets try displaying a single value from the array above:

```
echo "Is " . $families['Griffin'][2] .
" a part of the Griffin family?";
```

The code above will output:

```
Is Megan a part of the Griffin family?
```

Array Example

```
<html>
<body>

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

</body>
</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>
```

```
<html>
<body>

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

</body>
</html>
```

PHP Functions

> To keep the script from being executed when the page loads, put it into a function.

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

> Call a function from anywhere within a page.

PHP Functions

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

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

> Give the function a name that reflects what the function does

> The function name can start with a letter or underscore (not a number)

PHP Functions

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

```
<html>
<body>

<?php
function writeName()
{
echo "Kai Jim Refsnes";
}

echo "My name is ";
writeName();
?>

</body>
</html>
```

PHP Functions - Parameters

Adding parameters...

> To add more functionality to a function, add parameters. A parameter is just like a variable.

> Parameters are specified after the function name, inside the parentheses.

PHP Functions - Parameters

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

```
<html>
<body>

<?php
function writeName($fname)
{
echo $fname . " Refsnes.<br />";
}

echo "My name is ";
writeName("Kai Jim");
echo "My sister's name is ";
writeName("Hege");
echo "My brother's name is ";
writeName("Stale");
?>

</body>
</html>
```

PHP Functions - Parameters

Output:

```
My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes.
My brother's name is Stale Refsnes.
```

PHP Functions - Parameters

```
<html>
<body>

<?php
function writeName($fname,$punctuation)
{
echo $fname . " Refsnes" . $punctuation . "<br />";
}

echo "My name is ";
writeName("Kai Jim",".");
echo "My sister's name is ";
writeName("Hege","!");
echo "My brother's name is ";
writeName("Ståle","?");
?>

</body>
</html>
```

This example adds different punctuation.

PHP Functions - Parameters

Output:

```
My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes!
My brother's name is Ståle Refsnes?
```

Superglobals

> Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope – and 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

$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).

Example:

```
<?php
$x = 75;
$y = 25;

function addition() {
    $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}
addition();
echo $z;
?>
```

$_SERVER is a PHP superglobal variable which holds information about headers, paths, and script locations.

Example:

```
<?php
    echo $_SERVER['PHP_SELF'];
    echo "<br>";
    echo $_SERVER['SERVER_NAME'];
    echo "<br>";
    echo $_SERVER['HTTP_HOST'];
    echo "<br>";
    echo $_SERVER['HTTP_REFERER'];
    echo "<br>";
    echo $_SERVER['HTTP_USER_AGENT'];
    echo "<br>";
    echo $_SERVER['SCRIPT_NAME'];
    ?>
```

$_SERVER['PHP_SELF']

Returns the filename of the currently executing script

$_SERVER['GATEWAY_INTERFACE']

Returns the version of the Common Gateway Interface (CGI) the server is using

$_SERVER['SERVER_ADDR']

Returns the IP address of the host server

$_SERVER['SERVER_NAME']

Returns the name of the host server

$_SERVER['SERVER_SOFTWARE']

Returns the server identification string (such as Apache/2.2.24)

$_SERVER['SERVER_PROTOCOL']

Returns the name and revision of the information protocol (such as HTTP/1.1)

$_SERVER['REQUEST_METHOD']

Returns the request method used to access the page (such as POST)

$_SERVER['REQUEST_TIME']

Returns the timestamp of the start of the request (such as 1377687496)

$_SERVER['QUERY_STRING']

Returns the query string if the page is accessed via a query string

$_SERVER['HTTP_ACCEPT']

Returns the Accept header from the current request

$_SERVER['HTTP_ACCEPT_CHARSET']

Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)

$_SERVER['HTTP_HOST']

Returns the Host header from the current request

$_SERVER['HTTP_REFERER']

Returns the complete URL of the current page (not reliable because not all user-agents support it)

$_SERVER['HTTPS']

Is the script queried through a secure HTTP protocol

$_SERVER['REMOTE_ADDR']

Returns the IP address from where the user is viewing the current page

$_SERVER['REMOTE_HOST']

Returns the Host name from where the user is viewing the current page

$_SERVER['REMOTE_PORT']

Returns the port being used on the user's machine to communicate with the web server

$_SERVER['SCRIPT_FILENAME']

Returns the absolute pathname of the currently executing script

$_SERVER['SERVER_ADMIN']

Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host)

$_SERVER['SERVER_PORT']

Returns the port on the server machine being used by the web server for communication (such as 80)

$_SERVER['SERVER_SIGNATURE']

Returns the server version and virtual host name which are added to server-generated pages

$_SERVER['PATH_TRANSLATED']

Returns the file system based path to the current script

$_SERVER['SCRIPT_NAME']

Returns the path of the current script

$_SERVER['SCRIPT_URI']

Returns the URI of the current page

$_REQUEST is used to collect data after submitting an HTML form.

```
<html> <body>
    <form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
      Name: <input type="text" name="fname">
      <input type="submit">
    </form>
    <?php
    if ($_SERVER["REQUEST_METHOD"] == "POST") {
        // collect value of input field
        $name = $_REQUEST['fname'];
        if (empty($name)) {
            echo "Name is empty";
        } else {
            echo $name;
        }
    }
    ?>
    </body></html>
```

PHP Forms - $_GET Function

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

> Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send (max. 100 characters).

PHP Forms - $_GET Function

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

```
http://www.w3schools.com/welcome.php?fname=Peter&age=37
```

Notice how the URL carries the information after the file name.

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

PHP Forms - $_GET Function

The "welcome.php" file can now use the $_GET function to collect form data (the names of the form fields will automatically be the keys in the $_GET array)

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

PHP Forms - $_GET Function

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

> This method should not be used when sending passwords or other sensitive information!

> However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

> The get method is not suitable for large variable values; the value cannot exceed 100 chars.

PHP Forms - $_POST Function

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

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

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

PHP Forms - $_POST Function

```
<form action="action.php" method="post">
 <p>Your name: <input type="text" name="name" /></p>
 <p>Your age: <input type="text" name="age" /></p>
 <p><input type="submit" /></p>
</form>
```

And here is what the code of action.php might look like:

```
Hi <?php echo htmlspecialchars($_POST['name']); ?>.
You are <?php echo (int)$_POST['age']; ?> years old.
```

PHP Forms - $_POST Function

Apart from **htmlspecialchars()** and **(int)**, it should be obvious what this does. **htmlspecialchars()** makes sure any characters that are special in html are properly encoded so people can't inject HTML tags or Javascript into your page.

For the age field, since we know it is a number, we can just convert it to an integer which will automatically get rid of any stray characters. The **$_POST['name']** and **$_POST['age']** variables are automatically set for you by PHP.

PHP Forms - $_POST Function

When to use **method="post"**?

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

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

Check which button was pressed

```
if (isset($_POST['button1']))
 echo "Button 1 was pressed";
elseif (isset($_POST['button2']))
 echo "Button 2 was pressed";
else
 echo "no button pressed";
```

Email validation:

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

URL validation

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

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

Form.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>
```

Welcome.php

```
<html>
<body>

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

</body>
</html>
```

COMPLETE FORM VALIDATION

```
<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 = "";

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"]);   } }

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 echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">

  Name: <input type="text" name="name“>
     <span class="error">* <?php echo $nameErr;?></span>  <br><br>

  E-mail: <input type="text" name="email">
     <span class="error">* <?php echo $emailErr;?></span>  <br><br>

 Website: <input type="text" name="website">
       <span class="error"><?php echo $websiteErr;?></span>   <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
            <input type="radio" name="gender" value="other">Other
            <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>
```

References

- ***https://www.tutorialspoint.com/php/***
- ***https://www.javatpoint.com/php-tutorial***
- ***https://www.w3schools.com/php/***
- ***https://www.guru99.com/php-tutorials.html***