PHP (PHP: Hypertext Preprocessor)

+ 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 case sensitive language.
+ PHP is free to download and use
+ 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.

Basic PHP Syntax:

- A PHP script can be placed anywhere in the document.
- PHP script starts with **<?php** and ends with **?>**:
- PHP statements are terminated by semicolon (;).

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

      (OR)

<?php
  // PHP code goes here
?>

      (OR)

<script language=”PHP”>
    //PHP code goes here
</script>
```

- 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:

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

       <?php
          echo "Hello World!";
       ?>
   </body>
</html>
```

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

```
// 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
*/
```

Variables:

- Variables are "containers" for storing information.
- PHP dynamically typed language. That is PHP has no-type declaration.
- PHP automatically converts the variable to the correct data type, depending on its value.
- In PHP, a variable starts with the $ sign, followed by the name of the variable

- The value can be assigned to the variables in the following manner.

  $var_name = value;

- When you assign a text value to a variable, put quotes around the value.
- If the value is not assigned to the variable then by default the value is NULL.
- PHP variable names are case-sensitive.

Following are some rules that must be followed while using the 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 _).

PHP Constants:

- Constants are like variables except that once they are defined they cannot be changed or undefined.
- A valid constant name starts with a letter or underscore (no $ sign before the constant name).

Note: Unlike variables, constants are automatically global across the entire script.

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

The example below creates a constant with a **case-sensitive** name:

Example:

```
<?php
   define("GREETING", "  Welcome  to  W3Schools.com!  ");
   echo GREETING;
?>
```

Output:

```
Welcome to W3Schools.com!
```

The example below creates a constant with a **case-insensitive** name:

Example:

```
<?php
   define("GREETING", "Welcome  to  W3Schools.com!",  true);
   echo greeting;
?>
```

Output:

```
Welcome to W3Schools.com!
```

Outputs:

In PHP there are two basic ways to get output:

1. echo and
2. print.

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.

The PHP echo Statement:

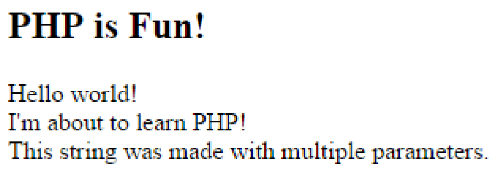
The echo statement can be used with or without parentheses: echo or echo ( ).

Display Text

The following example shows how to output text with the echo command (notice that the text can contain HTML markup):

Example:

```
<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple parameters.";
?>
```

Output: 

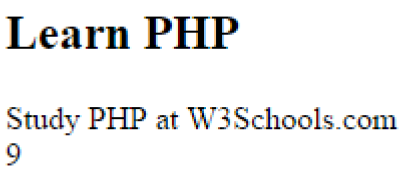
```
PHP is Fun!

Hello world!
I'm about to learn PHP!
This string was made with multiple parameters.
```

Display Variables

The following example shows how to output text and variables with the echo statement:

```
<?php
   $txt1 = "Learn PHP";
   $txt2 = "W3Schools.com";
   $x = 5;
   $y = 4;

   echo "<h2>$txt1</h2>";
   echo "Study PHP at $txt2<br>";
   echo $x + $y;
?>
```

Output:



```
Learn PHP

Study PHP at W3Schools.com
9
```

<u>The PHP print Statement</u>:

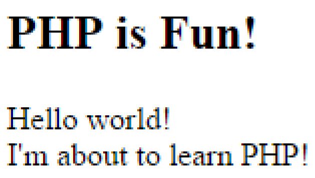
The print statement can be used with or without parentheses: print or print( ).

Display Text: The following example shows how to output text with the print command (notice that the text can contain HTML markup):

Example:

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

Output:



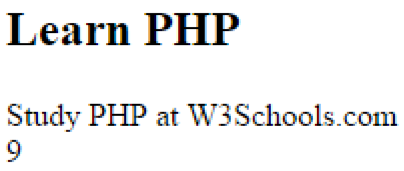
```
PHP is Fun!

Hello world!
I'm about to learn PHP!
```

Display Variables: The following example shows how to output text and variables with the print statement:

Example:

```
<?php
     $txt1 = "Learn PHP";
     $txt2 = "W3Schools.com";
     $x = 5;
     $y = 4;

     print "<h2>$txt1</h2>";
     print "Study PHP at $txt2<br>";
     print $x + $y;
?>
```

Output:



```
Learn PHP

Study PHP at W3Schools.com
9
```

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
- NULL

Scalar / Simple Data types

- **Array**
- **Object**

Compound Data Types

- 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;
?>
```

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.

In the following example $x is an integer.

Example:

```
<?php
     $x = 5985;
     echo $x;
?>
```

Output:

```
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.

Example:

```
<?php
     $x = 10.365;
     echo $x;
?>
```

Output:

```
10.365
```

PHP Boolean:

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

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

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

PHP Array

- An array stores multiple values in one single variable. In the following example $cars is an array.

Example:

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

Output:

```
I like Volvo, BMW and 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:

Output:

```
VW
```

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

```
     echo $herbie->model;
?>
```

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 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 Arithmetic Operators

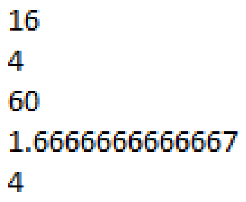
The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

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

Example:

```
     <html>
         <body>
              <?php
                $x = 10;
                $y = 6;
                echo $x + $y;
                echo $x - $y;
                echo $x * $y;
```

Output:



```
16
4
60
1.6666666666667
4
```

```
                echo $x / $y;
                echo $x % $y;

              ?>

   </body>
   </html>
```

PHP Assignment Operators:

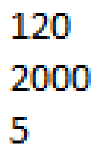
- The PHP assignment operators are used with numeric values to write a value to a variable.
- The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Operator	Example	Same as..	Description
=	$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

Example:

```
<html>
     <body>
          <?php
               $x = 20;
               $x += 100;
                  echo $x;
               $y = 20;
                $y *= 100;
                  echo $y;

               $z = 100;
               $z /= 20;
               echo $z;

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

Output:



```
120
2000
5
```

PHP Comparison Operators:

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

Operator	Name	Example	Result
==	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:

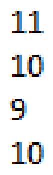
- The PHP increment operators are used to increment a variable's value.
- The PHP decrement operators are used to decrement a variable's value.

Operator	Name	Description
++$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

Example:

```
<html>
     <body>
          <?php
               $x = 10;
               echo ++$x;
               $y = 10;
               echo $y++;
               $z = 10;
               echo --$z;
               $w = 10;
               echo $w--;
          ?>
     </body>
</html>
```

Output:



```
11
10
9
10
```

PHP Logical Operators:

- The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result
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

Example:

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

               $x = 100;
               $y = 50;
               if ($x == 100 and $y == 50) {
                    echo "Hello world!";
               }
          ?>
     </body>
</html>
```

Output:

```
Hello world!
```

PHP String Operators:

- PHP has two operators that are specially designed for strings

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

Example:

```
<html>
   <body>
       <?php
       $txt1 = "Hello";
             $txt2 = " world!";
```

Output:

```
Hello world!
```

```
                    $txt1 .= $txt2;
                    echo $txt1;
              ?>
           </body>
        </html>
```

PHP Array Operators:

- The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	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

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.
- Function names are NOT case-sensitive.

Create a User Defined Function in PHP:

- A user defined function declaration starts with the word "function".

Syntax:

```
function functionName()

{
  code to be executed;
}
```

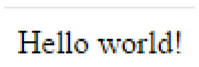
Note: A function name can start with a letter or underscore (not a number).

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.

Example:

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

Output:



```
Hello world!
```

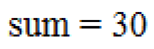
PHP Function Arguments

- 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.

Example:

```
<?php
      function add($a, $b) {
           echo "sum=",$a+$b;
      }
      add(10,20); // call the function
?>
```

Output:



```
sum = 30
```

The above example has a function with two argument ($a, $b). When the **add()** function is called, we also pass along $a and $b values (e.g. 10 and 20), the $a and $b is used inside the function, which outputs sum of $a and $b.

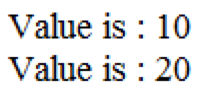
PHP Default Argument Value:

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

Example:

```
<?php
      function displayVal($a=20) {
         echo "Value is : ".$a."<br>";
      }
displayVal(10); // call the function
```

Output:



```
Value is : 10
Value is : 20
```

```
          displayVal(); // call the function
      ?>
```

PHP Functions returning value:

- A function can return a value using the **return** statement in conjunction with a value or object. Return stops the execution of the function and sends the value back to the calling code.
- You can return more than one value from a function using **return array(1,2,3,4)**.
- Following example takes two integer parameters and adds them together and then returns their sum to the calling program. Note that **return** keyword is used to return a value from a function.

Example:

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

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

This will display following result –

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

PHP – Arrays:

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

- An array is a data structure that stores one or more similar type of values in a single value. For example if you want to store 100 numbers then instead of defining 100 variables its easy to define an array of 100 length.
- In PHP, the array() function is used to create an array:

```
array();
```

There are three different kind of arrays.

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

Indexed arrays:

- An array with a numeric index. Values are stored and accessed in linear fashion.
- These arrays can store numbers, strings and any object but their index will be represented by numbers.
- By default array index starts from zero.

Creation of indexed arrays :

There are two ways to create indexed arrays:

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

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

(OR)

2) the index can be assigned manually:

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

Array elements Access : The numerical index are used in a script to access the array elements.

Examples:
```
$age[0] ;
$age[1] ;
```

Example:

Following is the example showing how to create and access numeric arrays.Here we have used **array()** function to create array.

```
<html>
   <body>
     <?php
         /* First method to create array. */
         $numbers = array( 1, 2, 3, 4, 5);

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

This will produce the following result –

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

Get The Length of an Array :

- The `count()` function is used to return the length (the number of elements) of an array.

Example: **Output:**

```
3
```

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

Associative arrays:

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

Creation of Associative arrays:

There are two ways to create an associative array:

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

(OR)

2.
```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

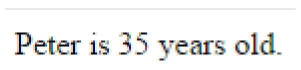
Array elements Access : The named keys are used in a script to access the array elements.

Examples:
```
$age['Peter'] ;
$age['Ben'] ;
```

Example:

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

Output:



```
Peter is 35 years old.
```

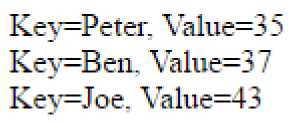
Loop Through an Associative Array:

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

Example:

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

foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
```

Output:



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

Multidimensional arrays :

- A multidimensional array is an array containing one or more arrays.
- A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.
- Values in the multi-dimensional array are accessed using multiple index.

Two-dimensional Arrays: A two-dimensional array is an array of arrays.

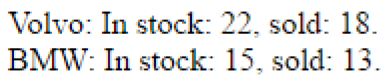
Example : `$cars = array ( array("Volvo",22,18), array("BMW",15,13));`

Now the two-dimensional **$cars** array contains two 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).

Example:

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

Output:



```
Volvo: In stock: 22, sold: 18.
BMW: In stock: 15, sold: 13.
```

Three-dimensional Arrays: A three-dimensional array is an array of arrays of arrays.

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

Sorting Arrays:

The elements in an array can be sorted in alphabetical or numerical order, descending or ascending.

Sort Functions For Arrays:

1. **sort()** - sort arrays in ascending order
2. **rsort()** - sort arrays in descending order
3. **asort()** - sort associative arrays in ascending order, according to the value
4. **ksort()** - sort associative arrays in ascending order, according to the key
5. **arsort()** - sort associative arrays in descending order, according to the value
6. **krsort()** - sort associative arrays in descending order, according to the key

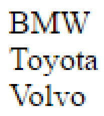
Sort Array in Ascending Order - sort() :

The following example sorts the elements of the $cars array in ascending alphabetical order.

Example:

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

$clength = count($cars);
for($x = 0; $x < $clength; $x++) {
    echo $cars[$x];
    echo "<br>";
```

Output:



```
BMW
Toyota
Volvo
```

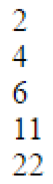
```
}
?>
```

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);
$arrlength = count($numbers);
for($x = 0; $x < $arrlength; $x++) {
 echo $numbers[$x];
    echo "<br>";
}
?>
```

Output:



```
2
4
6
11
22
```

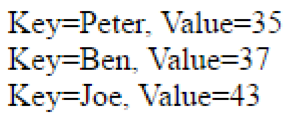
Sort Array (Ascending Order), According to Value - asort() :

The following example sorts an associative array in ascending order, according to the value

Example:

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

Output:



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

Sort Array (Ascending Order), According to Key - ksort() :

The following example sorts an associative array in ascending order, according to the key

Example:

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

Output:

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

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 Superglobals variables are:

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_COOKIE
- $_SESSION

PHP $GLOBALS:

- $GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script.
- PHP stores all global variables in an array called $GLOBALS [*index*]. The *index* holds the name of the variable.

The example below shows how to use the super global variable $GLOBALS.

Example:

```
<html>
<body>
<?php
$x = 75;
$y = 25;

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

addition();
echo $z;
```

Output:

```
100
```

```
?>

</body>
</html>
```

In the example above, since z is a variable present within the $GLOBALS array, it is also accessible from outside the function.

PHP $_SERVER:

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

The following table lists the most important elements that can go inside $_SERVER:

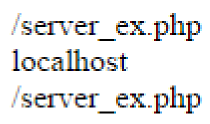
Element	Description
`$_SERVER['PHP_SELF']`	Returns the filename of the currently executing script
`$_SERVER['SERVER_ADDR']`	Returns the IP address of the host server
`$_SERVER['SERVER_NAME']`	Returns the name of the host server
`$_SERVER['REQUEST_METHOD']`	Returns the request method used to access the page (such as GET or POST)
`$_SERVER['SERVER_PORT']`	Returns the port on the server machine being used by the web server for communication (such as 80)
`$_SERVER['SCRIPT_NAME']`	Returns the path of the current script
`$_SERVER['SCRIPT_URI']`	Returns the URI of the current page

The example below shows how to use some of the elements in $_SERVER.

Example:

```
<html>
<body>
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

Output:



```
/server_ex.php
localhost
/server_ex.php
```

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

PHP $_REQUEST:

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

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. Then, we can use the super global variable $_REQUEST to collect the value of the input field

Example:

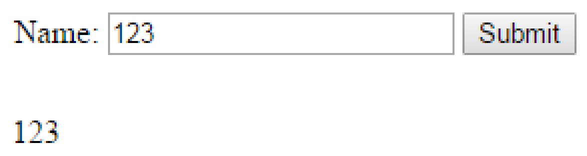
Form.html

```
<html>
<body>

<form method="post" action="submitform.php">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

</body>
</html>
```

submitform.php

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_REQUEST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>
```

Output:



```
123
```

```
123
```

PHP $_POST:

PHP $_POST is widely used to collect form data after submitting an HTML form with method="post".

PHP $_GET:

PHP $_GET can also be used to collect form data after submitting an HTML form with method="get".

Form Handling

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

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.

- 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.

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

- 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.

PHP - A Simple HTML Form:

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

```
<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>
```

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.

welcome.php

```
<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
```

Database Connectivity In PHP

PHP provides built-in database connectivity for a wide range of databases

- ❖ MySQL, PostgreSQL, Oracle, Berkeley DB, Informix, mSQL, Lotus Notes, etc
- ❖ Starting support for a specific database may involve PHP configuration steps

• Another advantage of using a programming language that has been designed for the creation of web apps.

Steps to create a database application in PHP:

1. Create a database connection
2. Select database you wish to use
3. Perform a SQL query
4. Do some processing on query results
5. Close database connection

1. Creating database connection

Use either mysql_connect or mysql_pconnect to create database connection

- ✓ mysql_connect: connection is closed at end of script (end of page)
- ✓ mysql_pconnect: creates persistent connection

• Connection remains even after end of the page
• Parameters

1. $Server – hostname of server
2. $Username – username on the database
3. $Password – password on the database
   mysql_connect($Server,$Username,$Password)

2. Select database you wish to use

mysql_select_db() – Pass it the database name

3. Perform a SQL query

- ❖ Create query string
  - $query = ‘*SQL formatted string*’
  - $query = ‘SELECT * FROM *table*’
- ❖ Submit query to database for processing
  - $result = **mysql_query**($query);
  - For UPDATE, DELETE, DROP, etc, returns TRUE or FALSE
  - For SELECT, SHOW, DESCRIBE or EXPLAIN, $result is an identifier for the results, and does not contain the results themselves
- ❖ $result is called a “resource” in this case
- ❖ A result of FALSE indicates an error
- ❖ If there is an error – **mysql_error()** returns error string from last MySQL call

4. Process Results:

Many functions exist to work with database results

• **mysql_num_rows()**

- Number of rows in the result set
- Useful for iterating over result set

• **mysql_fetch_array()**

- Returns a result row as an array
- Can be associative or numeric or both (default)
- $row = mysql_fetch_array($result);
- $row[‘*column name*’] :: value comes from database row with specified column name
- $row[0] :: value comes from first field in result set.

Process Results Loop

• Easy loop for processing results:

```
$result = mysql_query($qstring);
$num_rows = mysql_num_rows($result);
for ($i=0; $i<$num_rows; $i++) {
$row = mysql_fetch_array($result);
// take action on database results here
}
```

5. Closing Database Connection

mysql_close()

❖ Closes database connection

❖ Only works for connections opened with **mysql_connect()**

❖ Connections opened with **mysql_pconnect()** ignore this call