

# Server Side Web Technology

- ☐ PHP
- ☐ MySQL

- ☐ Internal (40 Marks)
    - ☐ Internal Exam – 10 Marks
    - ☐ Written Exam – 10 Marks
    - ☐ Assignment – 05 Marks
    - ☐ Practical Exam – 10 Marks
    - ☐ viva – 05 Marks
- 

Total                      40 Marks

- ☐ External (60 Marks)

# Scripting Language :

- ❑ A Scripting Language is a programming language that is used to manipulate, customise, and automate the facilities of an existing system.
- ❑ In such systems, useful functionality is already available through a user interface, and the scripting language is a mechanism for exposing that functionality to program control.
- ❑ Scripting languages are easier to learn than compiled languages.
- ❑ Scripts get more done in less code and less coding time than compiled languages.

# Types of Scripting

- ☐ Server Side Scripting
- ☐ Client Side Scripting

## Example of Scripting Language

- ☐ JavaScript
- ☐ VBScript
- ☐ Perl
- ☐ Python
- ☐ PHP
- ☐ TCL (Tool Command Language)

# Server Side Or Client Side

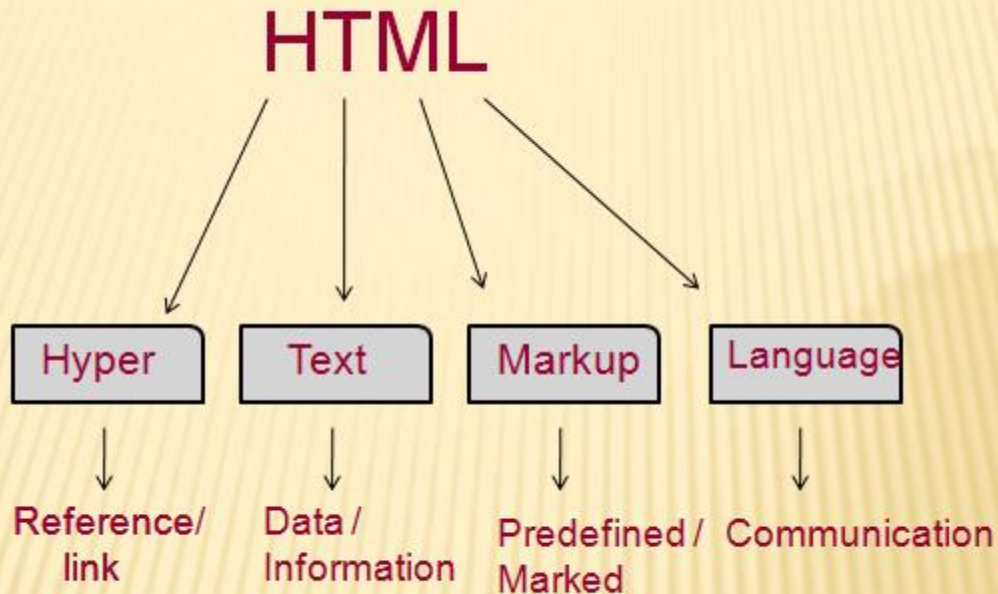
- **Server-side**

- Processed by server
- Does not rely on browser support
- Provide browser with data that does not reside on client
- Script code not visible in page source
- Can
  - Manage sessions (shopping baskets, etc.)
  - Database processing
- Eg: PHP ,Python , Ruby

- **Client-side**

- Processed by browser
- Does not depend on web server requirements
- Does not need to interact with server to create content
  - processed by browser
- Script code is viewable in page source
- Eg: HTML, JavaScript

## Prerequisites before learning PHP ( if used for website creation)



**HTML** Stands for **Hyper text markup language**. HTML is the client side scripting language i.e used to design static web page.

Hyper means reference link, text means data, mark-up means predefined, language means something that needs to communicate.

1. IT IS TAG DRIVEN LANGUAGE. <>
2. INTERPRETED LANGUAGE.

**HTML IS NOT CASE SENSITIVE**

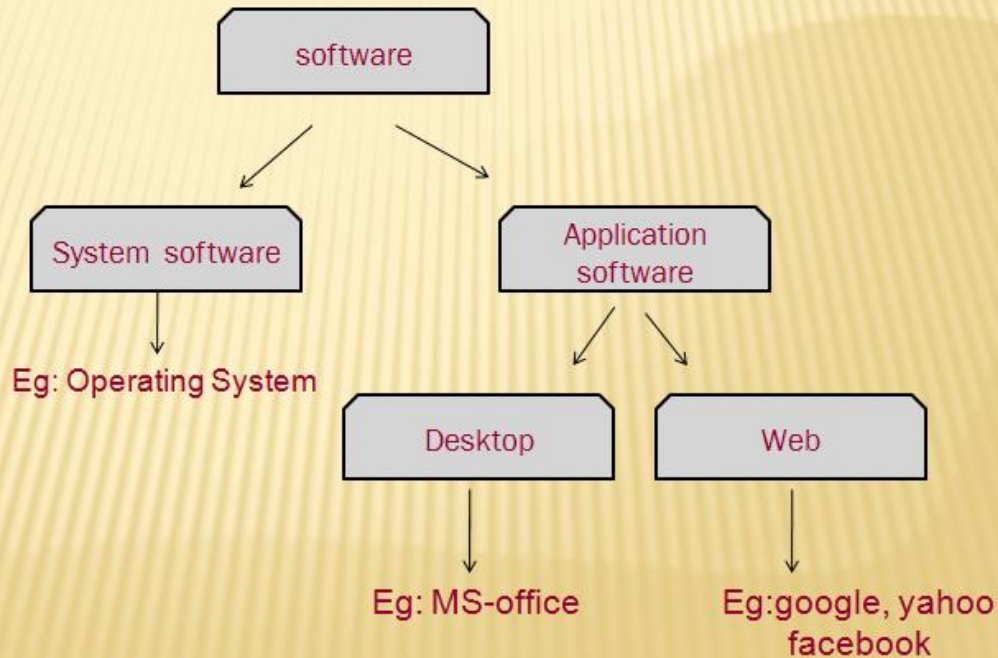
**CSS** stands for **Cascading Styles Sheets**

**CSS**, or **Cascading Styles Sheets**, is a way to style and present HTML. Whereas the HTML is the **meaning** or **content**, the style sheet is the **presentation** of that document.

Styles have a format of '**property: value**' and most properties can be applied to most HTML tags.

# Software Architecture

## Software and Language References



### Software

All the instruction and programs loaded into the computer's memory are known as software.

### System Software

System software is the computer software designed to operate the computer hardware and to provide a platform for running application software.

### Application Software

Application software also known as "application" or "app", is computer software designed to help the user to perform specific tasks.

### Desktop Application/window application

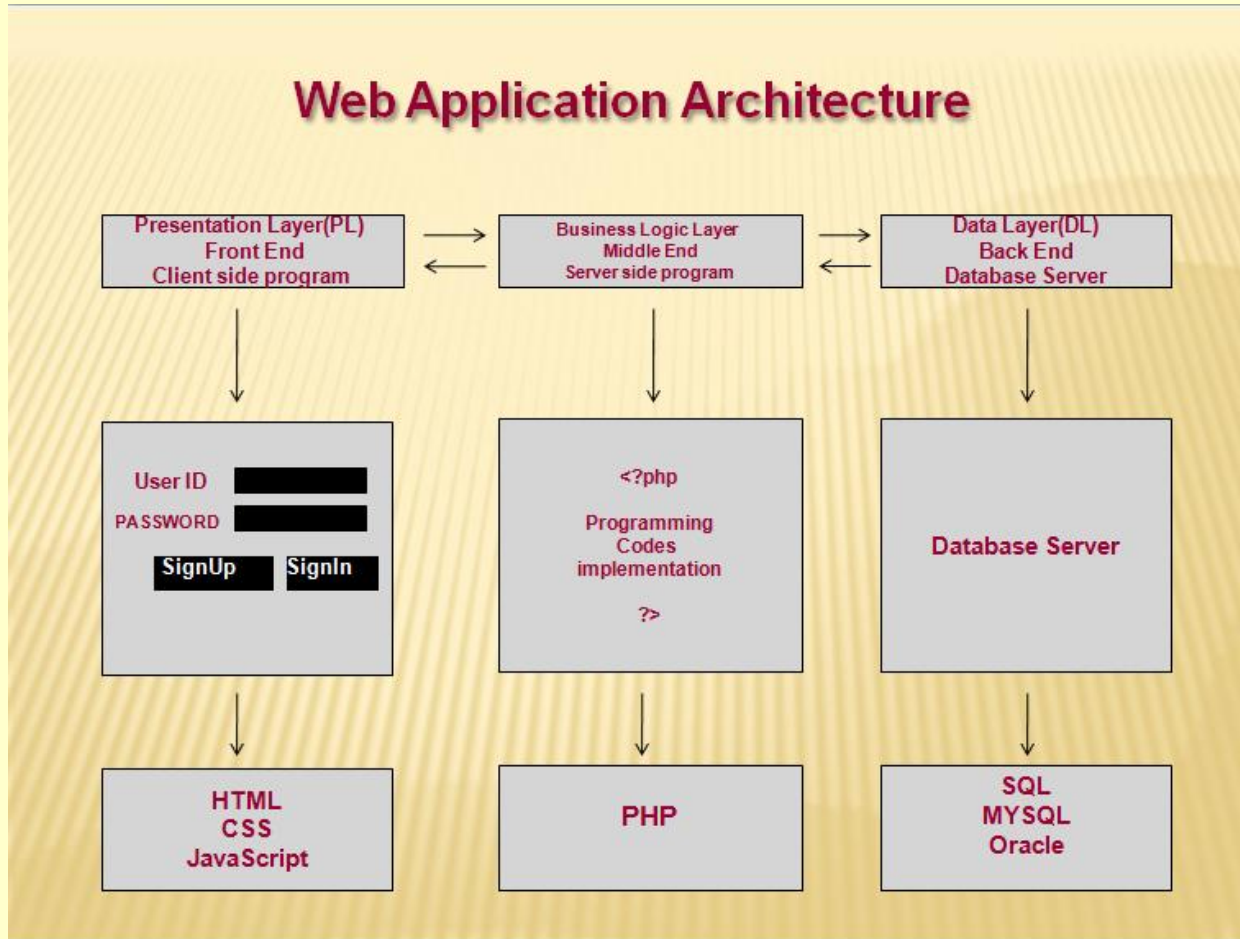
Desktop application can be accessed single user at a time. it is not globally accessible.

eg. MS-office

### Web Application

Web application like google, yahoo, facebook can accessed by multiple users at the same time globally.

# Web Application Architecture



## Presentation layer(PL)

Presentation layer is a user interface.

Client/User interact with any application through presentation layer.

## Business layer(BL)

Business logic is a logic for any particular problem to solve it. it is also called programming.

## Data link layer(DL)

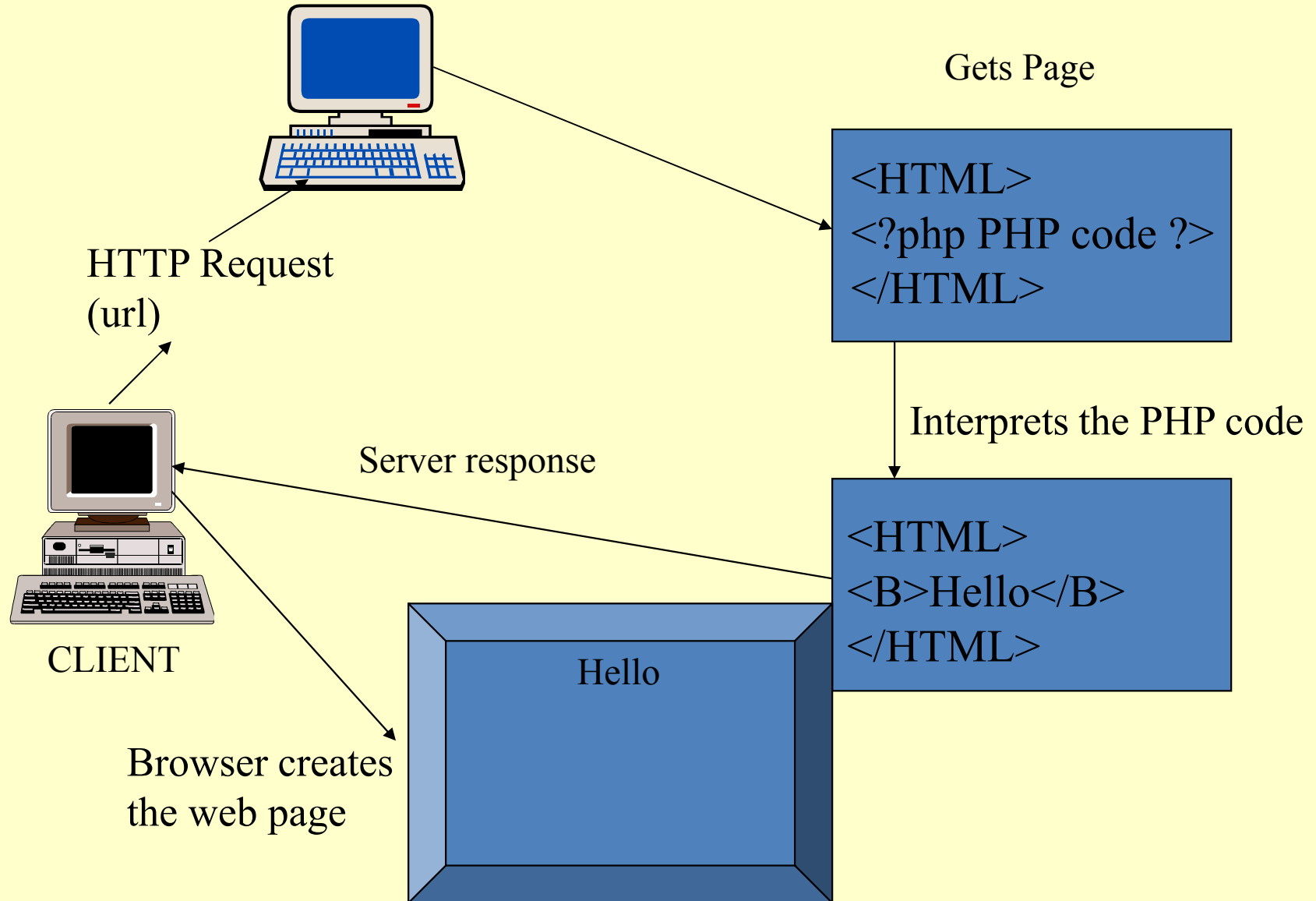
Independent application to transfer data to each other through data link layer.

# Introduction to PHP

- PHP Hypertext Preprocessor.
  - Other Names : Personal Home Page, Professional Home Page
- Is a **server side scripting language**.
  - *Capable of generating the HTML pages*
- HTML generates the web page with the static text and images.
- However the need evolved for dynamic web based application, mostly involving database usage.
- **PHP was originally developed by Rasmus Lardorf in 1994**, and was publicly released in June 1995. This released version is known as PHP 2.



# WEB SERVER



# Why PHP?

- ..there are no. of server side scripting available like ASP, SSJS, JSP.....

## Advantages Of PHP

### 1) Performance

- Scripts written in PHP executes faster than those written in other scripting language, with numerous independent benchmarks, putting the language ahead of competing alternatives like JSP,ASP.NET and PERL.
- Third party accelerators are available to further improve performance and response time.

### 2) Portability

- PHP is available for UNIX, MICROSOFT WINDOWS, MAC OS, and OS/2.
- PHP Programs are portable between platforms.

### 3)Ease Of Use

- Its syntax is clear and consistent. Simple to use.

# Why PHP?-continued

## 4) Open Source

- PHP is an open source.

## 5) Third-Party Application Support

- Its support for a wide range of different databases, including MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. PHP 5.3 Supports more than fifteen different database engines, and it includes a common API for database access.

## 6) Community Support

- A community-supported language like PHP is the access it offers to the creativity and imagination of hundreds of developers across the world.

# PHP Software Requirement

## PHP Server

The PHP Community Provides Some types of Software Server solution under The GNU (General Public License).

**These are the following:**

1. WAMP Server    2. LAMP Server    3. MAMP Server    4. XAMPP Server

All these types of software automatic configure inside operating system after installation it having PHP, MySQL, Apache and operating system base configuration file, it doesn't need to configure manually.

Server	Stands for
WAMP	Microsoft window o/s, Apache Mysql PHP
LAMP	Linux Operating System Apache Mysql PHP
MAMP	Mac os Apache Mysql PHP
XAMPP	x-os(cross operating system) Apache Mysql PHP Perl

# PHP Language features

- *PHP language features such as control structures, operators, variable types, function declaration, class/object declaration are almost similar to any compiled or interpreted language such as C or C++.*

# Syntax used in PHP file

A PHP script starts with `<?php` and ends with `?>`. This script can be placed anywhere in the document. PHP usually has HTML tags, along with some PHP coding.

## **PHP Syntax:**

```
<?php
```

```
PHP Code Here
```

```
?>
```

## **Note:**

The PHP file should be saved using the `*.php` extension, not `*.html`, if saved in `*.html` extension then the PHP code will not be executed.

# For Example

To display an message "Good Morning" following code is executed.

```
<html>
<body>
<?php
echo "Good Morning";
//This is a single line comment
/*
This is
a comment
block
*/
?>
</body>
</html>
```

# Data Types in PHP

A data type is defined as a set of values and the allowable operations on those values. Almost all programming languages explicitly include the notion of data type, though different languages may use different terminology.

In PHP the data type of a variable is not set by the programmer. PHP decides the data type of variables after interpreting the web page. Basic datatype included in php are Variables are nothing but identifiers to the memory location to store data.



# Data Types in PHP(continued..)

- PHP supports various different data types, categorized into two categories.
- **“Compound Data Types”**  
Eg: Array Object
- **“Scalar Data Types”**  
Eg: Boolean Integer Float String
- **“Two special types”**  
resource and NULL

# Data Types in PHP(continued..)

## Boolean:

A Boolean type is typically denoted by "bool" or "boolean". It can hold values either "true" (1) or "false" (0). Any non zero values and non empty string are also considered as TRUE.

When you want to declaring boolean variable, you can declare as given below

```
$variable;
```

where boolean denotes the type of the variable.

You can also declare variable without datatype, in such case PHP will try to determine type of variable based on the value hold by that variable. For example if you assign a string value to variable, this variable becomes a string variable. Below I have declared a variable without its type.

```
$variable= true; //Here $variable is a variable of type boolean.
```

If you want to get a type of a variable, you can use `gettype()` function.

Example:

```
$variable=true;  
print gettype($variable);
```

**Integer :** The integer data type is used to specify a numeric value without a fractional component.

The range of integers in PHP is equivalent to the range of the long data type in C. On 32-bit platforms, integer values can range from -2,147,483,648 to +2,147,483,647. When you are declaring integer variable, you can declare as given below

```
$variable; // where integer denotes the type of the variable.
```

You can also declare variable without datatype, in such case PHP will try to determine type of variable based on the value hold by that variable. Below I have declared a variable without its type.

```
$variable=10; // Here the variable hold the integer value so it is of type integer.
```

**Example:**

```
print gettype($variable);
```

Above code will print type integer because integer(whole number) value is assigned to that variable. If you specify a number beyond the limits of the integer type then PHP automatically converts larger values to floating point numbers.

## Double :

The Double data type is used to specify floating point numbers.

When you are declaring double variable, you can declare as given below

```
$variable;
```

where double denotes the type of the variable.

The range of floating point numbers in PHP is equivalent to the range of the double type in C, Double can range from 1.7E-308 to 1.7E+308. A double may be expressed either as a regular number with a decimal point or in scientific notation.

For example:

```
$var=0.017;
```

```
$var=17.0E-3
```

## String:

A string is a sequence of characters. A string can be delimited by single quotes or double quotes.

For Example:

```
$str1 = "This is a string datatype variable";  
$str2 = 'This is also a string datatype variable';
```

String declared within double-quotes are subject to variable substitution and escape sequence handling, while string declared within single-quotes will not perform variable substitution and escape sequence handling.

For example:

```
$str="Integer";  
echo "String\t$str\n";
```

**RESULT:**  
String Integer

This displays "String" followed by a space and then "Integer" followed by a newline. Here variable substitution is performed on the variable \$str and the escape sequences are converted to their corresponding characters.

## String (continued..)

```
echo 'String\t$str\n';    //where $str="Integer";
```

**RESULT:**

String\t\$str\n

In this case, the output is exactly "String\t\$str\n". There is no variable substitution or handling of escape sequences, the string within the inverted commas are printed as such.

String Concatenation is a process of adding two strings. This is done by attaching one string to the end of another string. This is done by using '.' (period) operator as shown below

```
$str1="Hello";  
$str2="World";  
echo "$str1"."$str2";
```

**RESULT:**

HelloWorld

This displays value of \$str1 as "Hello" followed by value of \$str2 as "World".

## Array:

An array is a compound data type that can contain multiple data values. Each array element can be retrieved by using the array variable name and its key/index value. The index value may be either numeric value or string value.

An array variable can be declared as

```
$val=3;  
$arrayname = array( "first element", 2,$val );  
echo $arrayname[0]; //prints: first element  
echo $arrayname[1]; //prints: 2  
echo $arrayname[2]; //prints: 3
```

Array values can hold values with different data type. As you see in the previous example, elements in an array can be any data type(string, integer, double). Array index always starts at position zero,so the first array element has an index of 0 and the last element has an index one less than the number of elements in the array. You can also use `print_r($arrayname)` function to print the values of an array.

PHP allows you to add an element at the end of an array without specifying an index.

## Array: (continued..)

For example:

```
$arrayname[] = "Test";
```

In this case, the "Test" element is given the index 3 in our \$arrayname array. If the array has non-consecutive elements, PHP selects the index value that is one greater than the current highest index value.

Arrays indexed using strings are called as associative arrays

Example:

```
$sarr["Jan"]=1;
```

```
$sarr["Feb"]=2;
```

we cannot use a simple counter in a for loop to work with this array. We can use the foreach loop or print\_r() function. In the following example we use the foreach loop to iterate through our associative array.

```
foreach($sarr as $sarrval=>$val)
```

```
{
```

```
echo "$val";
```

```
}
```



## NULL:

The special NULL value represents that a variable has no value. NULL is the only possible value of type NULL.

A variable is considered to be NULL if

- it has been assigned the constant NULL.
- it has not been set to any value yet.
- it has been unset()

There is only one value of type NULL, and that is the case-insensitive keyword NULL. **NULL variable can be declared as**

```
$var = NULL;
```

Here the variable hold the Null value so it is of type NULL.

## Example:

```
<?php
$numA = null;
$numB = Null;
$numC = NULL;
$numD;
if($numA==null)
{
echo "<p>numA";
}
if($numB==null)
{
echo "<p>numB";
}
if($numC==null)
{
echo "<p>numC";
}
if($numD==null)
{
echo "<p>numD";
}
?>
```

6/6/2015

## Result/Output:

numA  
numB  
numC  
numD

## Resource :

The Resource is a special PHP Data Type that refers to external resource (e.g. file, image etc.) which is not part of the PHP native language.

## Type Conversion :

Type conversion is nothing but data type change of a certain variable from one type to another. **Such manual overriding of data types is called Type Casting.**

PHP will automatically convert one type to another whenever possible. For example if you assign a string value to variable, this variable becomes a string variable. Sometimes automatic type conversion will lead to unexpected results. For example, calling "print" on an array makes PHP to print "Array" instead of array elements. PHP does not automatically convert the array to a string of all its elements. If you want to permanently change type of variable, you can use **settype() function**.

## Type Conversion (continued..)

### Example:

```
<?php
$var1 = "5bar"; // string
$var2 = true; // boolean
settype($var1, "integer"); // $var1 is now set to 5 (integer)
settype($var2, "string"); // $var2 is now set to "1" (string)
?>
```

In the above example \$var1 is assigned to string value so it is a string variable and \$var2 holds boolean value so it is a boolean variable. Now \$var1 is forced to type cast to integer and \$var2 is forced to type cast to string by using settype function.

If you want to change type temporary, ie, want to use inside an expression, you can use **type casting** .

### Example:

**(type)\$variable**

where type is a type of variable you wish to cast to.

## Type Conversion (continued..)

Examples are:

```
<?php
```

```
$var1 = 12.2; // double
```

```
$var2 = "pre"; // string
```

```
$var3= 10; //integer
```

```
$var4=(string)$var1; // $var1 is now set to 12.2 (string)
```

```
$var5=(boolean)$var2; // $var2 is now set to 1 (boolean)
```

```
$var6=(double)$var3;// $var3 is now set to 10(double)
```

```
print gettype($var4);
```

```
print gettype($var5);
```

```
print gettype($var6); ?>
```

Here \$var1 is converted to type string, \$var2 is converted to type boolean and \$var3 is converted to type double.

It is important to know that most of data type conversions can be done by on its own and some conversions can lead to loss of information. Consider an example of converting a float value to integer will lead to loss of information.

# Variables

- A variables is a temporary containers that can hold a value.
- A variable can hold any type of data (e.g. strings, integers, objects etc.).
- PHP is loosely typed programming language.
- We identify the variables by adding the dollar sign \$ before their name.
- Variables names must include letters (a-z,A-Z), numbers and underscores only.
- A variables name must start either with a letter or an underscore.
- An unassigned (unbound) variable has the value, NULL.
  - The unset function sets a variable to NULL
  - The IsSet function is used to determine whether a variable is NULL
- **PHP Variables names are case sensitive.**

Example:

- \$ \_num1      **OK**
- ~~\$2num~~      **NOT OK**
- \$number12      **OK**

# Dynamic variables

A dynamic variable is named using two dollar signs (\$\$) and is associated with a normal variable that has a similar name.

The value of an usual variable gives the name (without the dollar symbol) to a second variable.

## Example:

```
<?php
$good_people = 12;
$var_name = "good_people";
echo $$var_name;
// assigning a value to $$var_name will change the value to $good_people
$$var_name = 'other value';
echo '<br />'. $good_people;
?>
```

## Result:

```
// 12
// other value
```

## Explanation:

A dynamic variable does not contain its own value. Instead, it contains the location where you can find the value, in other words, the name of another variable.

**Note:** *Dynamic variables, as those by reference can be confusing and it's better to avoid them.*



# Super Global Variables in PHP

PHP super global variable is used to access global variables from anywhere in the PHP script.

PHP Super global variables is accessible inside the same page that defines it, as well as outside the page while local variable's scope is within the page that defines it.

**The PHP super global variables are :**

1) **`$_GET["FormElementName"]`** : It is used to collect value from a form(HTML script) sent with method='get'. information sent from a form with the method='get' is visible to everyone(it display on the browser URL bar).

2) **`$_POST["FormElementName"]`** : It is used to collect value in a form with method="post">. Information sent from a form is invisible to others.(can check on address bar)

3) **`$_REQUEST["FormElementName"]`** : This can be used to collect data with both post and get method.

4) **`$_FILES["FormElementName"]`** : It can be used to upload files from a client computer/system to a server.

OR

4) **`$_FILES["FormElementName"]["ArrayIndex"]`** : Such as File Name, File Type, File Size, File temporary name.

# Super Global Variables in PHP-Continued

5) **`$_SESSION["VariableName"]`** : A session variable is used to store information about a single user, and are available to all pages within one application.

6) **`$_COOKIE["VariableName"]`** : A cookie is used to identify a user. cookie is a small file that the server embedded on user computer.

7) **`$_SERVER["ConstantName"]`** : `$_SERVER` holds information about headers, paths, and script locations.

eg. `$_SERVER["SERVER_PORT"]`  
`$_SERVER["SERVER_NAME"]`,  
`$_SERVER["REQUEST_URI"]`

# Function Name Within Variable

We can assign a function name to be the value of a variable we have. We can later use that variable in order to call the function.

## Example:

```
<?php
function doSomething()
{
    echo 'In Function';
}
$var = 'doSomething';
$var(); //that will result in calling the function
?>
```

## RESULT:

In Function

# PHP Constants

- A constant is an identifier (name) for a simple value. As the name suggests, that value cannot change during the execution of the script (except for magic constants, which aren't actually constants). A constant is case-sensitive by default. By convention, constant identifiers are always uppercase.
- The name of a constant follows the same rules as any label in PHP. A valid constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

## Syntax:

You can define a constant by using the `define()`-function or by using the `const` keyword outside a class definition. **Once a constant is defined, it can never be changed or undefined.**

`define("CONSTANT_NAME", VALUE)`

Example:

```
<?php
define ("USER_NAME","sam");
print "Hi ". USER_NAME;
?>
```

Result: Hi sam

# PHP Constants

- Only scalar data (boolean, integer, float and string) can be contained in constants. It is possible to define constants as a resource, but it should be avoided, as it can cause unexpected results.
- You can get the value of a constant by simply specifying its name. Unlike with variables, you should not prepend a constant with a \$. You can also use the function `constant()` to read a constant's value if you wish to obtain the constant's name dynamically. Use `get_defined_constants()` to get a list of all defined constants.

## Example :

// Valid constant names

```
define("FOO", "something");  
define("FOO2", "something else");  
define("FOO_BAR", "something more");
```

// Invalid constant names

```
define("2FOO", "something");
```

?>

# PHP Magic Constant

There are a number of predefined constants available in PHP

## PHP Magic Constant

__LINE__	The current line number of the file.
__FILE__	The full path and filename of the file.
__FUNCTION__	The function name
__CLASS__	The class name
__METHOD__	The class method name
PHP_VERSION	The PHP version
PHP_INT_MAX	The PHP integer value limit

### Example: \_\_LINE\_\_

```
<?php echo "The Line number : ".  
__LINE__; ?>
```

Output : The Line number : 2

# PHP Operators

- Arithmetic operators
  - `+`, `-`, `*`, `/`, `%`
- String operators
  - `.`
- Assignment operators
  - `=`, `.=`, `+=`, `-=`, `*=`, `/=`
- Comparison operators
  - `==`, `===`, `!=`, `<>`, `!==`, `<`, `>`, `<=`, `>=`
- Increment, decrement operators
  - `++`, `--` (pre and post)
- Logical operators
  - `!`, `&&`, `||`, `and`, `or`, `xor`

# Arithmetic operators

Arithmetic Operators are use for doing arithmetical operations. List of Arithmetic Operators are listed below.

Operator	Name	Example
+	Addition	$10 + 5 = 15$
-	Substraction	$10 - 5 = 5$
*	Multiplication	$10 * 5 = 50$
/	Division	$10 / 5 = 2$
%	Modulus	$10 \% 5 = 0$ (Module Show the remainder value)



### Example:

```
<?php
$fno = 10; // assign first no
$sno = 5; // assign second no
$add = $fno + $sno; //Addition operator
echo "Addition of 10 + 5: ". $add; // Print the result of $add
echo "<br>";
$subs = $fno - $sno; //Substraction operator
echo "Substraction of 10 - 5: ". $subs; // Print the result of $subs
echo "<br>";
$multi = $fno * $sno; //Multiplication operator
echo "Multiplication of 10 * 5: ".$multi; // Print the result of $multi
echo "<br>";
$division = $fno / $sno; //Division operator
echo "Division of 10 / 5: ".$division; // Print the result of $division
echo "<br>";
$modulus = $fno % $sno; //Modulus operator
echo "Modulus of 10 % 5 : ".$modulus; // Print the result of $modulus, Modulus
operator show the remainder value only.
?>
```

# Assignment operators

Assignment operator always use with initialization of a variable. Mostly it consist single charecter eqaul to (=). This assignment operator takes the value of its right-hand operand and assign it to its left-hand operand

---

## Operator Example

+=	\$x += 10; or \$x = \$x + 10;(Equivalent)
-=	\$x -= 10; or \$x = \$x - 10;(Equivalent)
*=	\$x *= 10; or \$x = \$x * 10;(Equivalent)
/=	\$x /= 10; or \$x = \$x / 10;(Equivalent)
%=	\$x %= 10; or \$x = \$x % 10;(Equivalent)
.=	\$x .= test; or \$x = \$x" test ";(Equivalent)

## Example :<?php

```
$x = 15; $y = 15; $z = 15; $a = 15; $b = 15; $c = 15;
```

```
$x+=10; // $x += 10; or $x = $x + 10;(Equivalent)
```

```
echo $x;
```

```
echo "<br>"; // Break Lilne
```

```
$y-=10; // $y -= 10; or $y = $y - 10;(Equivalent)
```

```
echo $y;
```

```
echo "<br>"; // Break Lilne
```

```
$z*=10; // $z *= 10; or $z = $z* 10;(Equivalent)
```

```
echo $z;
```

```
echo "<br>"; // Break Lilne
```

```
$a/=10; // $a /= 10; or $a = $a / 10;(Equivalent)
```

```
echo $a;
```

```
echo "<br>"; // Break Lilne
```

```
$b%=10; // $b %= 10; or $b = $b % 10;(Equivalent)
```

```
echo $b;
```

```
echo "<br>"; // Break Lilne
```

```
$c.= test; // $c .= 10; or $c = $c" test";(Equivalent)
```

```
echo $c;
```

```
echo "<br>"; // Break Lilne
```

```
?>
```

# Comparison operators

Comparison operator perform test on their operands. They return the value true or false.

---

Operator	Name	Example
==	Equivalent	\$x == 5;
!=	Non-Equivalent	\$x != 5;
>	Greater	\$x > 5;
>=	Greater than or Equals to	\$x >= 5;
<	Less Than	\$x < 5;
<=	Less Than or Equals to	\$x <= 5;

**Example :** <?php

```
$x = 5;
if($x == 5){
    print "Result True";
}Else{
    print "Result False";
}
echo "<br>"; // Break line
if($x != 5){
    print "x is not eqaul to 5";
}else{
    print "x is eqaul to 5";
}
echo "<br>"; // Break line
if($x > 6){
    print "x is Greater than 5";
}else{
    print "x is not greater than 5";
}
```

## Example (continued..)

```
echo "<br>"; // Break line
if($x >= 5){
print "x is Greater than or Equal to 5";
}else{
print "x is not greater than 5 ";
}
echo "<br>"; // Break line
if($x < 6){
print "x is Less than 6";
}else{
print "x is not Less than 5";
}
echo "<br>"; // Break line
if($x <= 5){
print "x is Less than or Equal to 5";
}else{
print "x is not Less than 5 ";
}
?>
```

# Logical Operators

The Logical Operator returns true or false value.

Operator	Name	Example
	Or	<code>\$x == 2    \$x &lt; 6;</code> Result is True (here <code>\$x=5;</code> )
or	Or	<code>\$x != 5;</code>
xor	Xor	<code>\$x &gt; 5;</code>
&&	And	<code>\$x &gt;= 5;</code>
and	And	<code>\$x &lt; 5;</code>
!	Not	<code>\$x &lt;= 5;</code>

## Example :

```
<?php
$x = 5;
if($x == 2 || $x < 6){
print "Result True";
}else{
print "Result False";
}
echo "<br>";
if($x == 2 or $x < 6){
print "Result True";
}else{
print "Result False";
}
echo "<br>";
if($x == 5 xor $x < 6){
print "Result Xor True";
}else{
print "Result Xor False";
}
```



## Example (continued..)

```
echo "<br>";
if($x == 5 && $x < 6){
print "Result True";
}else{
print "Result False";
}
echo "<br>";
if($x == 5 and $x < 6){
print "Result True";
}else{
print "Result False";
}
echo "<br>";
if($x != 5){
print "Result True";
}else{
print "Result False";
}
} ?>
```

# Ternary Operator

Ternary operator is another conditional operator.

## Syntax

**(expr1) ? (expr2) : (expr3)**

In the above syntax the expr1 is a condition when satisfied or true it evaluates expr2, otherwise expr3.

## Example:

```
<?php
$mark = 50;
$result = ($mark >= 60)? "Passed Exam": "Failed in Exam";
print "$result";
?>
```

## Result:

Failed in Exam

In the above example, marks of a student is checked in an expression, since its false the value "Failed in Exam" is displayed, if true the first value might be printed.

# Increment and Decrement Operator

Example	Name	Effect
<code>++\$a</code>	Pre-increment	Increments \$a by one, then returns \$a.
<code>\$a++</code>	Post-increment	Returns \$a, then increments \$a by one.
<code>--\$a</code>	Pre-decrement	Decrements \$a by one, then returns \$a.
<code>\$a--</code>	Post-decrement	Returns \$a, then decrements \$a by one.

## Example of Increment/Decrement Operators

```
<?php
$num1 = 12;
$num2 = 24;
$num3 = 32;
$num4 = ++$num2;
$num5 = $num3++;
$num6 = $num1--;
echo "<BR>num1=$num1";
echo "<BR>num2=$num2";
echo "<BR>num3=$num3";
echo "<BR>num4=$num4";
echo "<BR>num5=$num5";
echo "<BR>num6=$num6";
?>
```

num5 is first getting  
the old value of num3  
num3 is incremented  
Afterwards.

### Result:

```
num1=11
num2=25
num3=33
num4=25
num5=32
num6=12
```

# Operators Precedence & Associativity

Operators	Description	Associativity
<code>new</code>	New object—highest precedence	None
<code>[]</code>	Array elements	Right to left
<code>!</code>	Logical Not	Right to left
<code>++</code>	Increment	Right to left
<code>--</code>	Decrement	Right to left
<code>(int), (double), (string), (array), (object)</code>	Cast	Right to left
<code>@</code>	Suppress errors	Right to left
<code>* / %</code>	Multiplication/division/modulus	Left to right
<code>+ - .</code>	Addition/subtraction/string concatenation	Left to right
<code>&lt; &lt;= &gt; &gt;=</code>	Comparison	None
<code>== != &lt;&gt; === !==</code>	Equality	None
<code>&amp;&amp;</code>	Logical And	Left to right
<code>  </code>	Logical Or	Left to right
<code>?:</code>	Conditional	Left to right
<code>= += -= *= /= %=</code>	Assignment	Right to left
<code>and</code>	Logical And	Left to right
<code>or</code>	Logical Or	Left to right
<code>,</code>	List separator—lowest precedence	Left to right

# Control Structures

If **structure** is used for conditional execution of code segment

## Syntax:

```
if (expr)
{
    Statements
}
```

In the above syntax, based on the "expr" the statements are executed

## **Example:**

```
<?php
if ($c > $d)
{
    echo "c is bigger than d";
}
?>
```

In the above example, only if the condition "\$c>\$d" is true, the message "c is bigger than d" is displayed

The conditional statement **"else"** is used as extension of "if" statement. If the condition fails then it executes another statements under the "else" condition.

### Syntax :

```
if (expr)
    {Statements} //true
else
    {Statements} //false
```

Based on the result of expressions, the statements are executed.

### **Example**

```
<?php
$c = 10;
$d = 20;
if ($c > $d)
    {echo "C is bigger than d";}
else
    {echo "D is bigger than c";}    ?>
```

### **Result:**

D is bigger than C

**else if** statement is used as extension of "If" structure if the condition fails then it executes another "If" condition to execute the code segment under the "else if" statement

### Syntax:

```
if (expr 1)
    {Statements} //true
elseif (expr2) //false
    {Statements} //true
else
    {Statements} //false
```

Based on the failure "expr1" condition, "expr2" is checked and then the statements are executed.



## Example

```
<?php
$c = 10;
$d = 10;
if ($c > $d)
{
    echo "c is bigger than d";
}
elseif ($c==$d)
{
    echo "c is equal to d";
}
else
{
    echo "d is smaller than c";
}
?>
```

## Result:

c is equal to d

The **Switch case** statement is used to compare a variable or expression to different values based on which a set of code is executed.

**Syntax :**

Switch (Variable or expression)

```
{  
    Case (value 0):  
        {statements}  
    Case (value 1):  
        {statements}  
    Case (value 2):  
        {statements}  
    Case (value n):  
        {statements}  
    default:  
        {statements}  
}
```

In the above syntax comparing the "variable or expression" in switch statement to the "value" in case, and the statements of that particular case is executed

## Example

```
<?php
$c=3;
switch ($c)
{
case 0:
    echo "value of $c = 0 <br>";
    break;
case 1:    echo "value of $c = 1 <br>";
    break;
case 2:    echo "value of $c = 2 <br>";
    break;
default:    echo "value of $c = Default value <br>";
    break;
}
?>
```

## Result:

value of 3 = Default value

In the above example based on the value of \$c the messages are displayed of that particular case which matches. The default case accepts anything not matched by other cases, so the value "3" displays the default message.

# Loops / Repetition

- ❑ Loops allow a statement to be executed repetitively
- ❑ A stopping condition must be provided, or condition to stop the loop
- ❑ Erroneous stopping condition may cause the loop to loop forever

**While structure** is type of loop statements, where the condition is checked at first, the iteration will not stop even if the value changes while executing statements.

**Syntax:**

```
while(expr)
{
    Statements
} //true
```

If the "expr" is true the statements are executed

## **Example:**

```
<?php
$c=1;
while ($c<=5)
{
    echo $c;
    $c++;
}
?>
```

## **Result:**

12345

**Do While** statement is same as the while statement, the only difference is that it evaluates the expression at the end.

### Syntax

```
do  
{  
    Statements  
}while (expr1);
```

In do while, the statements are executed first, then the expr1 is checked, only drawback of Do While is that it may execute statements once even if the condition is false.

## **Example of do while**

```
<?php
$c =1;
do {
    echo $c;
    $c++;
}
while ($c<=5);
?>
```

## **Result**

12345



**For structure** is a loop structure used in PHP.

## **Syntax**

```
for (initializtion;condition;increment)
{
    Statements
} //true
```

The "initializtion" is the initial value, "condition" is the condition value, "increment" is the increment or decrement value, every time based on the true value of "condition" the loop is executed, and at the end of every iteration it checks the value of "increment"

## **Example**

```
<?php
for($c=0;$c=2;$c++)
{
    echo "The value c is " . $c;
    break;
}
?>
```

## **Result**

The value c is 0

The value c is 1

The value c is 2

If the "expr2" does not have any value then a **break statement** can be used to terminate the loop. See the example below

### **Example**

```
<?php
for ($c = 1; ; $c++) {
    if ($c > 5)
    {
        break;
    }
    echo $c;
}
?>
```

### **Result:**

12345

**"Continue"** is used to skip the current loop iteration and continue with the next iteration of the loop. But "Break" is used to exit from the the whole loop.

### **Syntax:**

**Continue (Optional numeric argument);**

In "Optional numeric argument" we can provide how many enclosing loops to skip.

### **Example:**

```
<?php
for ($c = 0; $c <=10; ++$c) {
if ($c == 5)
{ continue;}
echo $c . "<br>";
}
?>
```

### **Result:**

0 1 2 3 4 6 7 8 9 10

In the above example the iteration of the loop is skipped only when the value is 5, then all other number's are printed .

# Functions:

Some of the useful features of PHP are its built-in functions. But PHP also allows you to write your own functions. The main advantage of custom functions is reuse. If you don't write functions, you may have to write the same piece of code numerous times in your application, whether it's within the same PHP file or in others.

## Syntax:

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

# General Characteristics of Function

- Functions need not be defined before they are called (in PHP 3, they must)
- Function overloading is not supported
- If you try to redefine a function, it is an error
- Functions can have a variable number of parameters
- Default parameter values are supported
- Function definitions can be nested
- Function names are NOT case sensitive
- The return function is used to return a value; If there is no return, there is no returned value.

## **Example of user defined Function:**

```
function write()  
{  
    echo "Minal Abhyankar";  
}  
echo "My name is ";  
write();
```

## **Result:**

My name is Minal Abhyankar

In the above example, the a function is called to print a string.

# Passing arguments to functions

The arguments can be passed to a function, using arguments list separated by commas.

Arguments can be passed in two ways

- 1) "Passing by Reference "
- 2) "Passing by Value"



# Passing by Value:

Arguments are passed by value as default in PHP, and the value is assigned directly in the function definition.

## Example

```
1)<?php
function fruit($type = "cherry")
{
    return "Fruit you chose is $type.";
}
echo fruit();
echo "<br>";
echo fruit(null);
echo "<br>";
echo fruit("Strawberry");
?>
```

## Result:

Fruit you chose is cherry.  
Fruit you chose is .  
Fruit you chose is Strawberry.

```
2)<?PHP
```

```
//demo for pass by value
$Variable_Value = 10;
echo "Before the function call = " .
$Variable_Value . "<BR>";
example($Variable_Value);
echo "After the function call = " .
$Variable_Value;
function example($Variable_Value)
{
    $Variable_Value = $Variable_Value + 10;
    echo "Inside of the function = " .
$Variable_Value . "<BR>";
}
?>
```

**Result:** Before the function call = 10  
Inside of the function = 20  
After the function call = 10

In the above example value "cherry" is passed to the function fruit, even values can be passed directly as in the last example.

# Passing By Reference:

Passing the address itself rather than passing the value to the function is Passing by Reference.

## Example:

```
1)<?php
function fruit(&$string)
{
$string .= 'Cherry';
}
$str = 'This is the fruit, ';
fruit($str);
echo $str;
?>
```

## Result:

This is the fruit Cherry

In the above example the value is assigned for "\$string" inside the function. Usually in passing by value a copy of the variable is modified, rather than a variable itself as in passing by reference. So it is suitable for large codes.

2)<?PHP

```
//demo for pass by reference
$Variable_Value = 10;
echo "Before the function call = " .
$Variable_Value . "<BR>";
example($Variable_Value);
echo "After the function call = " .
$Variable_Value;
function example(&$Variable_Value)
{
$Variable_Value = $Variable_Value + 10;
echo "Inside of the function = " .
$Variable_Value . "<BR>";
}
```

## Result:

Before the function call = 10  
Inside of the function = 20  
After the function call = 20

# Returning values

Values are returned using the statement `return()`, this statement ends the code execution and passes the control to the line from which it was called. Any values can be returning like arrays, objects.

## Syntax:

`return value;`

## Example:

```
<?php
function sub($first,$second)
{
    $total = $first - $second;
    return $total;
}
$first = "3";
$second ="2";
$total = sub($first,$second);
echo "$total";  ?>
```

## Result:

1 (**Explanation :** -In the above example the function `sub` returns the value `"$total"`, when the function is called the subtracted value of 3 and 2 is returned as 1. )

# Variable Number of Arguments

PHP provides three built-in functions that assist handling variable length arguments list:

`func_num_args()`

This function returns the number of arguments.

`func_get_arg()`

This function returns the argument its indexed number was passed to  
(**example :-**

`func_get_arg(0)` returns the first argument,

`func_get_arg(1)` returns the second argument etc...).

`func_get_args()`

This function returns an array that holds all arguments.

# Variable Number of Arguments

```
<?php
function doSomething()
{
    $number_of_arguments = func_num_args();
    echo "<P>".$number_of_arguments;
    if($number_of_arguments!=0)
    {
        for($index=0; $index<$number_of_arguments; $index++)
        {
            echo "<br>".func_get_arg($index);
        }
    }
}
doSomething();
doSomething("Moshe");
doSomething("David","Moshe");
doSomething("Jane","Sam", "Rocky");
?>
```

**Result:**

0

1

Moshe

2

David

Moshe

3

Jane

Sam

Rocky

# Variable Functions

PHP has variable variables so it is not surprising we have variable functions. This particular piece of clever functionality allows you to write code like this:

## Example:

```
1) <?php
    $func = "sqrt";
    print $func(49);
?>
```

## Result:

7

## Explanation : -

PHP sees that you are calling a function using a variable, looks up the value of the variable, then calls the matching function.

```
2) <?php
    $myfunc = function ($numA,$numB)
    {
        return $numA+$numB;
    };
    echo $myfunc(4,4);
?>
```

## Result:

8

## Explanation : -

PHP allows us to define anonymous functions. We can assign the anonymous function to a variable and invoke it using that variable.

# Recursive Functions

Recursive Function calling technique is to make a function call itself

## Example:

```
<?php
function factorial($number)
{
    if ($number == 0)
        return 1;
    return $number * factorial($number - 1);
}

echo factorial(5);
?>
```

## Result:

120

# Variable scope

The **scope of a variable** is the context within which it is defined and can be used. The **variables** used in PHP functions can be of three types: **locals**, **globals** and **statics**.

Any variable defined inside a function is by default limited to the local function scope, is available only in the code within that function

## Example:

```
<?php
function test()
{
    $x = 8; echo $x;
}
echo $x; test();
?>
```

## Result:

Notice: Undefined variable: x in C:\server\www\file.php on line 7

8

**Explanation :** - The "*echo \$x*" expression outside the function returns an error because the *\$x* variable isn't defined or recognized outside the *test()* function, but the "echo" statement inside function refers to a local *\$x* variable, which is defined within *test()* function.



# Global variables

A **global variable** can be accessed in any part of the program.

If you want to use inside a function a variable defined outside it, that variable must be explicitly declared to be global in the function. This is accomplished by placing the keyword **GLOBAL** (or *global*) in front of the variable that should be recognized as global (or more variables separated by comma). This will extend the scope of that variable inside the function.

## Example:

```
<?php
$x = 8; $y = 9; $total = 0;
function getSum()
{ GLOBAL $x, $y, $total;
$total = $x + $y;
echo $x. ' + ' . $y. ' = ' . $total; }
getSum();
echo '<br />'. $total;
?>
```

## Result:

```
7 + 8 = 17
17
```

## // second way by using \$GLOBALS

```
<?php
echo 'second way by using $GLOBALS (super
global array)'. "<br>";
$name = "Minal";
function hello1() {
echo "Hello $GLOBALS[name]!"; }
hello1();
?>
```

Result: second way by using \$GLOBALS (super global array)  
Hello Minal!

**Explanation :** - By declaring \$x and \$y variables GLOBAL within the function, the values of these variables can be used inside the function.

The \$total variable being set as GLOBAL, all references to it, inside or outside function, refer to the same global version. Changing its value within the function will carry forward.

# Static variables

A **static variable** exists only in a local function scope, but it does not lose its value when program execution leaves this scope.

Declaring a variable static makes its value to be remembered by a function for the next time it is called.

To declare a variable static, place the keyword **STATIC** (or *static*) in front of the variable (or more variables separated by comma).

## Example:

```
<?php
function f_static()
{ STATIC $x = 0;
echo '<br /> x = '. $x; $x++; } /
f_static();
f_static();
f_static();
?>
```

## Result:

```
x = 0
x = 1
x = 2
```

**Explanation :** - In the *f\_static()* function, \$x is initialized only in first call of function and every time the *f\_static()* is called it will remember the value of \$x, and will print and increment it.

# PHP Forms :

## 1) \$\_POST Method

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

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

"welcome.php" looks like this:

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

**Output** could be something like this:

Welcome Minal!

You are 100 years old.

## 2) \$\_GET Method

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

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

"welcome.php" looks like this:

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

**Output** could be something like this:

Welcome Minal!

You are 28 years old.

### 3) \$\_REQUEST Method

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

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

**Example :**

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

File:welcome.php

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

**Output** could be something like this:

Welcome Minal!

You are 28 years old.

# Predefined Functions

Predefined or built in functions are those functions which comes with PHP by default. But to make some of these functions to work, one need to have some PHP extensions compiled into PHP. Otherwise it will give some fatal errors. For example to use MySql functions to work properly, PHP should be compiled with Mysql support.

It's important to know what a function returns, or if a function works directly on a passed in value etc., to improve the quality of the PHP code.

```
<?php
$Fname = $_POST["Fname"];
$Lname = $_POST["Lname"];
$gender = $_POST["gender"];
$food = $_POST["food"];
$quote = $_POST["quote"];
$education = $_POST["education"];
$TofD = $_POST["TofD"];
?>
<html>
<head>
<title>Personal INFO</title>
</head>
```

```
<body> <form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
First Name:<input type="text" size="12" maxlength="12" name="Fname"><br />
Last Name:<input type="text" size="12" maxlength="36" name="Lname"><br />
Gender:<br />
Male:<input type="radio" value="Male" name="gender"><br />
Female:<input type="radio" value="Female" name="gender"><br />
Please choose type of food:<br />
Steak:<input type="checkbox" value="Steak" name="food[]"><br />
Pizza:<input type="checkbox" value="Pizza" name="food[]"><br />
Chicken:<input type="checkbox" value="Chicken" name="food[]"><br />
<textarea rows="5" cols="20" name="quote" wrap="physical">Enter your favorite
quote!</textarea><br />
Select a Level of Education:<br />
<select name="education">
<option value="Jr.High">Jr.High</option>
<option value="HighSchool">HighSchool</option>
<option value="College">College</option></select><br />
Select your favorite time of day:<br />
<select name="TofD" size="3">
<option value="Morning">Morning</option>
<option value="Day">Day</option>
<option value="Night">Night</option></select><br />
<input type="submit" value="submit" name="submit">
</form></body></html>
```



# What is an Array?

- An array can store one or more values in a single variable name.
- Each element in the array is assigned its own ID so that it can be easily accessed.
- \$array[key] = *value*;

# 3 Kinds of Arrays

- 1) **Numeric Array**
- 2) **Associative Array**
- 3) **Multidimensional Array**

# Numeric Array

- A numeric array stores each element with a numeric ID key.
- 3 ways to write a numeric array.

# Automatically

Example:

```
$names = array("Peter","Quagmire","Joe");
```

# Manually

Example:

```
$names[0] = "Peter";
```

```
$names[1] = "Quagmire";
```

```
$names[2] = "Joe";
```

# The ID can be used in a script

## Example:

```
<?php
```

```
$names[0] = "Peter";
```

```
$names[1] = "Quagmire";
```

```
$names[2] = "Joe";
```

```
echo $names[1] . " and " . $names[2] . " are ". $names[0] . "'s  
neighbors";
```

```
?>
```

## OUTPUT:-

Quagmire and Joe are Peter's neighbors

# Associative Arrays

- 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.

# Example

Using an array to assign an age to a person.

```
$ages = array("Brent"=>42, "Andrew"=>25,  
"Joshua"16=>);
```

```
$ages['Brent'] = "42";
```

```
$ages['Andrew'] = "25";
```

```
$ages['Joshua'] = "16";
```



# The Id can be used in a script

Example: -

```
<?php
```

```
$ages['Brent'] = "42";
```

```
$ages['Andrew'] = "25";
```

```
$ages['Joshua'] = "16";
```

```
echo Brent is " . $ages['Brent'] . " years old.";
```

```
?>
```

OUTPUT : -

Brent is 42 years old.

# 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.

# Example

```
<?php
$row1 = array('a','b','c');
$row2 = array('d','e','f');
$row3 = array('g','h','i');
// make 2-dim array of rows
$matrix = array($row1, $row2, $row3);
echo $matrix[0][0]."<br>";
echo $matrix[1][2]."<br>";
echo $matrix[2][1]."<br>";
$matrix[2][2] = 'l';
echo $matrix[2][2]."<br>";
?>
```

## Result:

```
a
f
h
l
```

## Example:

```
<?php
$product1 = array('name' => 'small gumball', 'price' => 0.02);
$product2 = array('name' => 'medium gumball', 'price' => 0.05);
$product3 = array('name' => 'large gumball', 'price' => 0.07);
// array of all products
$products = array($product1, $product2, $product3);

for($i=0; $i<count($products); $i++)
{
    echo "Product: $i is {$products[$i]['name']} and costs {$products[$i]['price']} each<br>";
}
?>
```

## Result:

Product: 0 is small gumball and costs 0.02 each  
Product: 1 is medium gumball and costs 0.05 each  
Product: 2 is large gumball and costs 0.07 each

# Array Functions

Function	Description
array()	Creates an array
array_merge()	Merges one or more arrays into one array
array_pop()	Deletes the last element of an array
array_product()	Calculates the product of the values in an array
array_push()	Inserts one or more elements to the end of an array
array_rand()	Returns one or more random keys from an array
array_reverse()	Returns an array in the reverse order
array_search()	Searches an array for a given value and returns the key
array_shift()	Removes the first element from an array, and returns the value of the removed element
array_slice()	Returns selected parts of an array
array_splice()	Removes and replaces specified elements of an array

# Array Functions

Function	Description
array_sum()	Returns the sum of the values in an array
array_unique()	Removes duplicate values from an array
array_unshift()	Adds one or more elements to the beginning of an array
asort()	Sorts an array and maintain index association
count()	Counts elements in an array, or properties in an object
current()	Returns the current element in an array
next()	Advance the internal array pointer of an array
prev()	Rewinds the internal array pointer
range()	Creates an array containing a range of elements
sizeof()	Alias of count()
sort()	Sorts an array

## Example of Array Functions:

```
<?php
$colorlist=array("red","green","blue","black","white");
print_r($colorlist); // debugging purpose to print array contents
echo "<br>";
echo "<font color=Magenta>";
var_dump($colorlist); // debugging purpose to print array contents
echo "</font>";
echo "<br>";
$colorlist[0]="red";
$colorlist[1]="green";
$colorlist[2]="blue";
$colorlist[3]="black";
$colorlist[4]="white";
echo "<br>";
echo "<hr color=indigo>";
echo current($colorlist)."<br>";
echo next($colorlist)."<br>";
echo prev($colorlist)."<br>";
echo "<hr color=indigo>";
```

**//to display specific element**

```
echo $colorlist[3]."<br>";
```

```
echo "<hr color=blue>";
```

**// to display array elements by using loop**

```
for($i=0;$i<=count($colorlist);$i=$i+2)
```

```
{      echo $colorlist[$i]."<br>"; }
```

```
echo "<br>";
```

**echo "count of elements in an array by using sizeof function"."<br>";**

```
echo sizeof($colorlist)."<br>";
```

```
echo "<br>";
```

**//to display array elements by using foreach loop**

```
foreach($colorlist as $a)
```

```
{ echo $a;
```

```
  echo "<br>";
```

```
}
```

```
echo "<br>";
```

```
?>
```



## OUTPUT :

```
Array ( [0] => red [1] => green [2] => blue [3] => black [4] => white )  
array(5) { [0]=> string(3) "red" [1]=> string(5) "green" [2]=> string(4) "blue"  
[3]=> string(5) "black" [4]=> string(5) "white" }
```

```
red  
green  
red  
black  
red  
blue  
white
```

```
count of elements in an array by using sizeof function  
5
```

```
red  
green  
blue  
black  
white
```

# String Functions

Function	Description
<code>trim()</code>	Removes whitespace at beginning and end of a string.
<code>ltrim()</code>	Removes whitespace at the beginning of a string.
<code>rtrim()</code>	Removes whitespace at the end of a string.
<code>htmleentities()</code>	Escapes all HTML entities.
<code>nl2br()</code>	Inserts a <code>&lt;br /&gt;</code> tag before each newline character in a string.
<code>strtoupper()</code>	Converts a string to uppercase.
<code>strtolower()</code>	Converts a string to lowercase.
<code>ucfirst()</code>	Converts the first character of a string to uppercase.
<code>ucwords()</code>	Converts the first character of each word in a string to uppercase.

# String Functions

<code>substr(str,pos,len)</code>	Returns a len length substring beginning with the character in position pos.
<code>substr(str,pos,-len)</code>	Returns a substring beginning with the character in position pos and chopping off the last len characters of the string. If the third argument (before_needle) is false (default), then it returns the part of the haystack from the needle on.
<code>strstr(haystack,needle,before_needle)</code>	If the third argument (before_needle) is true, then it returns the part of the haystack before the needle. The needle can be a string or an integer (or a number that can be converted to an integer).
<code>stristr(haystack,needle,before_needle)</code>	Same as <code>strstr()</code> , but <i>case insensitive</i> .
<code>strpos(haystack,needle)</code>	Finds the position of the first occurrence of a specified needle in a haystack (string). The needle can be a string or an integer (or a number that can be converted to an integer).
<code>strrpos(haystack,needle)</code>	Finds the position of the last occurrence of a specified needle in a haystack (string). The needle can be a string or an integer (or a number that can be converted to an integer).
<code>str_replace()</code>	Replaces all occurrences of one string with another string.

# String Functions

`strcmp()`

Compares two strings. Returns  $< 0$  if `str1` is less than `str2`,  $> 0$  if `str1` is greater than `str2`, and  $0$  if they are equal.

`strcasecmp()`

Like `strcmp()` but case *insensitive*.

`strlen()`

Returns the length of a string.

## Example :

```
<?php
$string1="webucator";
echo "<br>";
echo "string1=".$string1;
echo "<br>";
echo "substr($string1,4)";
echo "<br>";
echo substr($string1,4);
echo "<br>";
echo "substr($string1,4,3)";
echo "<br>";
echo substr($string1,4,3);
echo "<br>";
$string1="WEB";
echo "<br>";
echo "string1=".$string1;
echo "<br>";
echo "<br>";
echo "strtolower($string1)";
echo "<br>";
echo strtolower($string1);
echo "<br>";
?>
```

## OUTPUT :

string1=webucator

substr(webucator,4)

cator

substr(webucator,4,3)

cat

string1=WEB

strtolower(WEB)

web

# Date and Calendar Function

Function	Description
<code>date( )</code>	Formats a local time/date
<code>time( )</code>	Returns the current time as a Unix timestamp
<code>cal_days_in_month( )</code>	Returns the number of days in a month for a specified year and calendar
<code>cal_info()</code>	Returns information about a given calendar

## Example of Date and calendar function:

```
<?php  
echo date("Y/m/d") . "<br />";  
echo date("Y.m.d") . "<br />";  
echo date("Y-m-d") . "<br />";
```

### OUTPUT :

```
2010/07/27  
2010.07.27  
2010-07-27
```

```
$a=time();  
echo $a;  
echo"<br>";
```

### OUTPUT :

```
1280249354
```

```
$num=cal_days_in_month(CAL_GREGORIAN,2,2010);  
echo "there was ". $num . " days in February 2010";  
echo"<br>";
```

### OUTPUT :

```
there was 28 days in  
February 2010
```

```
$info =cal_info(0);  
print_r($info);  
echo"<br>";
```

### OUTPUT :

```
Array ( [months] => Array ( [1] => January [2] => February [3] =>  
March [4] => April [5] => May [6] => June [7] => July [8] => August  
[9] => September [10] => October [11] => November [12] =>  
December ) [abbrevmonths] => Array ( [1] => Jan [2] => Feb [3] =>  
Mar [4] => Apr [5] => May [6] => Jun [7] => Jul [8] => Aug [9] =>  
Sep [10] => Oct [11] => Nov [12] => Dec ) [maxdaysinmonth] => 31  
[calname] => Gregorian [calsymbol] => CAL_GREGORIAN )
```

# Regular Expression

- A regular expression is a specially formatted pattern that can be used to find instances of one string in another.
- Several programming languages including Visual Basic, Perl, JavaScript and PHP support regular expressions.
- Regular expressions take a lot of the hassle out of writing lines and lines of repetitive code to validate a string

**There are 2 types of regular expressions:**

- POSIX Extended
- Perl Compatible



PHP has six functions that work with regular expressions. They all take a regular expression string as their first argument, and are shown below:

- **ereg:** The most common regular expression function, ereg allows us to search a string for matches of a regular expression.
- **ereg\_replace:** Allows us to search a string for a regular expression and replace any occurrence of that expression with a new string.
- **eregi:** Performs exactly the same matching as ereg, but is case insensitive.
- **eregi\_replace:** Performs exactly the same search-replace functionality as ereg\_replace, but is case insensitive.
- **split:** Allows us to search a string for a regular expression and returns the matches as an array of strings.
- **spliti:** Case insensitive version of the split function

# ereg()

- `ereg(regex, string)` searches for the pattern described in *regex* within the string *string*.
- It returns false if no match was found.
- If you call the function as `ereg(regex, string, matches)` the matches will be stored in the array *matches*. Thus *matches* will be a numeric array of the grouped parts (something in ()) of the string in the string. The first group match will be `$matches[1]`.

# ereg\_replace

- `ereg_replace ( regex, replacement, string )` searches for the pattern described in *regex* within the string *string* and replaces occurrences with *replacement*. It returns the replaced string.
- If *replacement* contains expressions of the form `\\number`, where *number* is an integer between 1 and 9, the number sub-expression is used.
  - `$better_order=ereg_replace('glass of (Karlsberg|Bruch)', 'pitcher of \\1',$order);`

# split()

- `split(regex, string, [max])` splits the string *string* at the occurrences of the pattern described by the regular expression *regex*. It returns an array. The matched pattern is not included.
- If the optional argument *max* is given, it means the maximum number of elements in the returned array. The last element then contains the unsplit rest of the string *string*.
- Use `explode()` if you are not splitting at a regular expression pattern. It is faster.

# case-insensitive function

- `eregi()` does the same as `ereg()` but work case-insensitively.
- `eregi_replace()` does the same as `ereg_replace()` but work case-insensitively.
- `spliti()` does the same as `split()` but work case-insensitively.

## Example of Regular Expression: (POSIX Extended )

```
<?php
function validateEmail($email)

{

return ereg("^([a-zA-Z]+@[a-zA-Z]+\.[a-zA-Z]+$", $email);

}

echo validateEmail("minal@gmail.com");

?>
```

A list of some of the most common regular expressions, as well as an example of how to use each one:

### **Beginning of string:**

**To search from the beginning of a string, use ^.**

For example,

```
<?php echo ereg("^hello", "hello world!"); ?>
```

Would return true, however

```
<?php echo ereg("^hello", "i say hello world!"); ?>
```

would return false, because hello wasn't at the beginning of the string.

### **End of string:**

**To search at the end of a string, use \$.**

For example,

```
<?php echo ereg("bye$", "goodbye"); ?>
```

Would return true, however

```
<?php echo ereg("bye$", "goodbye my friend"); ?>
```

would return false, because bye wasn't at the very end of the string.

## Any single character:

To search for any character, use the dot.

For example,

```
<?php echo ereg(".", "cat"); ?>
```

would return true, however

```
<?php echo ereg(".", ""); ?>
```

would return false, because our search string contains no characters. You can optionally tell the regular expression engine how many single characters it should match using curly braces. If I wanted a match on five characters only, then I would use ereg like this:

```
<?php echo ereg(".{5}$", "12345"); ?>
```

The code above tells the regular expression engine to return true if and only if at least five successive characters appear at the end of the string.



## Repeat character zero or more times

To tell the regular expression engine that a character may exist, and can be repeated, we use the **\*** character. Here are two examples that would return true:

```
<?php echo ereg("t*", "tom"); ?>
```

```
<?php echo ereg("t*", "fom"); ?>
```

Even though the second example doesn't contain the 't' character, it still returns true because the \* indicates that the character may appear, and that it doesn't have to. In fact, any normal string pattern would cause the second call to ereg above to return true, because the 't' character is optional.

## Repeat character one or more times

To tell the regular expression engine that a character must exist and that it can be repeated more than once, we use the **+** character, like this:

```
<?php echo ereg("z+", "i like the zoo"); ?>
```

The following example would also return true:

```
<?php echo ereg("z+", "i like the zzzzzzoo!"); ?>
```

## Repeat character zero or one times

We can also tell the regular expression engine that a character must either exist just once, or not at all. We use the **?** character to do so.

```
<?php echo ereg("c?", "cats are fuzzy"); ?>
```

If we wanted to, we could even entirely remove the 'c' from the search string shown above, and this expression **would still return true**. The '?' means that a 'c' **may** appear anywhere in the search string, but doesn't have to.

## The space character

To match the space character in a search string, we use the predefined **Posix class, [[:space:]]**. The square brackets indicate a related set of sequential characters, and ":space:" is the actual class to match.

We can match a single space character like this:

```
<? Php echo ereg("Minal[[:space:]]Abhyankar", "Minal Abhyankar"); ?>
```

# Example of Regular Expression :

In case of a registration form you may want to control available user names a bit. Let's suppose you don't want to allow any special character in the name except "\_.-" and of course letters and numbers. Besides this you may want to control the length of the user name to be between 4 and 20.

## Example of Regular Expression : (Perl Compatible)

```
$pattern = '/^[a-zA-Z0-9_.-]{4,20}$/';
```

```
$username = "this.is.a-demo_-";
```

```
if (preg_match($pattern,$username))
```

```
echo "Match";
```

```
else
```

```
echo "Not match";
```

# File Handling In PHP

## To count number of lines in a file.

To explain it we assume a file named `test.txt` with two lines in it.

### Step 1:

Define the file name in to a variable

Example: `$file1 = "./test.txt";`

### Step 2:

Create a instance for the file using the method **`file('filename')`**

Example: `$lines = file($file1);`

Here we create a file instance for the file by passing the variable `$file1` created in step 1. The result is stored in the variable `$lines`, which will be an array of lines.

### Step 3:

Now we can count the lines by passing the array of lines (`$lines`) to the method **`count()`**

Example: `$count = count($lines);`

The method "**`count()`**" takes an array as argument and returns the number of lines in it.

### Step 4:

Printing the result using the function `echo('string');`

Example: `echo($count)` 1-`

## Example :

```
<?php
$file1 = "./test.txt";
$lines = file($file1);
$count = count($lines);
echo($count);
?>
```

## Result:

2

# To Find File Size in php

## Step 1:

It can be calculated in one step using the php function or method

**filesize('filename');**

## Example:

```
$fsize = filesize("./test.txt");
```

This method filesize() takes the file name as an argument, finds and returns the size of the file in Bytes. The result may not be accurate for huge files with size greater than a size of 2GB.

## Example

```
<?php
```

```
$file = "./test.txt";
```

```
$fsize = filesize($file);
```

```
echo($fsize);
```

```
?>
```

**The Result is : 35**

# Reading File

## `readfile()`

### Syntax :

`int readfile ( string filename [, bool use_include_path [, resource context]])`

- If you want to output a file to the screen without doing any form of text processing on it whatsoever, *readfile()* is the easiest function to use.
- When passed a filename as its only parameter, *readfile()* will attempt to open it, read it all into memory, then output it without further question. If successful, *readfile()* will return an integer equal to the number of bytes read from the file.
- If unsuccessful, *readfile()* will return false, and there are quite a few reasons why it may fail. For example, the file might not exist, or it might exist with the wrong permissions.

## Example:

```
<?php
    readfile("/home/paul/test.txt");
    readfile("c:\boot.ini");
?>
```



# file\_get\_contents() and file()

## 1) file\_get\_contents()

string **file\_get\_contents** ( string \$filename [, bool  
\$use\_include\_path = false [, resource \$context [,  
int \$offset = -1 [, int \$maxlen ]]]] )

## 2) file()

array **file** ( string *filename* [, bool *use\_include\_path*  
[, resource *context*]])

## Example:

```
<?php
    $filestring = file_get_contents($filename);
    $filearray = explode("\n", $filestring);

    while (list($var, $val) = each($filearray))
    {
        ++$var;
        $val = trim($val);
        echo "Line $var: $val<br />";
    }
?>
```

# fopen() and fread()

## Step 1:

Define the file name in to a variable

Example: `$file = "./test.txt";`

## Step 2:

We will open a stream (connection) with the file using the method or php function **fopen('filename','type')**

**Example:** `$open = fopen($file, "r");`

The method fopen takes two arguments, first the file name and second argument that determines the file-opening mode. To read a file it should be 'r'.

## Step 3:

Now we will find the size of the file using the php method **filesize('filename');**

**Example:** `$fsize = filesize($file);`

This method takes the file name as argument and returns the size of the file in Bytes.

#### Step 4:

We will read the content in the file using the php function **fread('stream','size')**

**Example:** `$fsize = fread($open,$fsize);`

We have passed two arguments to the method fread(). The first argument is the stream obtained in step2. The second argument is the size that has to be read from the file. To read the full file we have pass the full size of the file.

**Example :**

```
<?php
$file = "./test.txt";
$open = fopen($file, "r");
$size = filesize($file);
$count = fread($open, $size);
echo($count);
?>
```

**The Result is**

Name – ABC XYZ

# Reading a file line by line in php

For an example we will use the file [test.txt](#) to explain it

The real difference comes in using the method **file()** instead of **fopen()**

## Step 1:

Define the file name in to a variable

Example: `$file1 = "../test.txt";`

## Step 2:

Create an instance of the file using the method **file('filename')**

Example: `$lines = file($file1);`

Here we create a file instance for the file by passing the variable `$file1` created in step 1. The result is stored in the variable `$lines`, which will be an array of lines.

## Step 3:

Now we will use a loop to capture every element in the array one by one and print the lines. Example: `foreach($lines as $line_num => $line)`

```
{  
echo $line;  
echo "<br>";  
}
```

In the above code we used the loop **foreach** to capture the elements or objects in the array of lines (\$lines). Each line is printed used the code "echo \$line";

**Example :**

```
<?php
$file1 = "./test.txt";
$lines = file($file1);
foreach($lines as $line_num => $line)
{
echo $line;
echo "<br>";
}
?>
```

**The Result is**

ABC

XYZ

# Creating a File in PHP .

Creating a new file in php is done using the function **fopen()**.

**Example:** `fopen('filename','w');`

To create a file using php

- a) call the function fopen with two required arguments.
- b) the first argument should be the file name. The new file will be created with the file name under the current(same) directory.
- c) the second argument should be 'w' indicating it as write mode.

## **Creating a new file under different Directory:**

To create a file in a different directory we have to pass the full file path as the argument for file name.

**Example:** `fopen('./images/filename.txt','w');`

This will create a file named filename.txt under the directory "images/"  
It is important to note that the directory specified in the path should exist.

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

Modes	Description
r	Read only. Starts at the beginning of the file
r+	Read/Write. Starts at the beginning of the file
w	Write only. Opens and clears the contents of file; or creates a new file if it doesn't exist
w+	Read/Write. Opens and clears the contents of file; or creates a new file if it doesn't exist
a	Append. Opens and writes to the end of the file or creates a new file if it doesn't exist
a+	Read/Append. Preserves file content by writing to the end of the file
x	Write only. Creates a new file. Returns FALSE and an error if file already exists
x+	Read/Write. Creates a new file. Returns FALSE and an error if file already exists. <b>Note:</b> If the fopen() function is unable to open the specified file, it returns 0 (false).

**Note:** If the fopen() function is unable to open the specified file, it returns 0 (false).



# Writing in to a file in php

## Step 1:

We will open a stream (connection) with the file using the php method or function **fopen('filename','type')**

Example: `$open = fopen("./test1.txt", "w");`

The method fopen takes two arguments, first the file name and second argument that determines the file opening mode. To write in to a file it should be 'w'.

## **Note:**

- This method will create a new file if file does not exist.
- It will overwrite the contents in the file if it exists

## Step 2:

Now we can write in to the file using the method **fwrite()**

Example: `fwrite($open, "Any thing to be written");`

This method takes two arguments. First the stream to file obtained as in step1. Second the content to be written. We can call as many write statements as required before closing the stream.

### Step 3:

After writing in the file we can close the stream opened on the file using the function **fclose()**

**Example:** `fclose($open);`

The function `fclose()` takes the stream to be closed as an argument and closes the stream.

**Example :**

```
<?php
$file = "./test1.txt";
$open = fopen($file, "w");
fwrite($open, "Hai this is test message");
?>
```

# Append in to file in PHP

It is much similar to file writing. The only difference is that we have to open the file in append mode ('a') instead of write mode ('w').

The below code will explain you, how we do it

**Example :**

```
<?php
```

```
$file = "test2.txt"; // Here we define the file path
```

```
$open = fopen($file, "a"); // Here we open the file in append mode using the php  
method fopen()
```

```
fwrite($open, "Hai this is test message"); // Here we append in to the file using  
the method fwrite('instance', 'message') passing fopen instance and the string as  
arguments
```

```
?>
```

# Temporary files in PHP

PHP has a function called *tmpfile()* which takes no parameters, but will create a temporary file on the system, *fopen()* it for you, and send back the file handle as its return value.

That file is then yours to read from and write to all you wish, and is deleted as soon as you *fclose()* the file handle or the script ends.

## Example:

```
<?php
$myarray[] = "This is line one";
$myarray[] = "This is line two";
$myarray[] = "This is line three";
$string = implode("\n", $myarray);
echo $string;
$handle = tmpfile();
$numbytes = fwrite($handle, $string);
fclose($handle);
echo $numbytes . " bytes written\n";
?
```

## Result :

52 bytes written

## *Other File Functions :*

➤ *Rewind()* is a very helpful function that moves the file pointer for a specified file handle (parameter one) back to the beginning. That is, if you call `rewind($handle)`, the file pointer of `$handle` gets reset to the beginning - this allows you to re-read in a file, or write over whatever you have already written.

➤ *Fseek()* allows you to move a file handle's pointer to an arbitrary position, specified by parameter two, with parameter one being the file handle to work with. If you do not specify a third parameter, *fseek()* sets the file pointer to be from the start of the file, meaning that passing 23 will move to byte 24 of the file (files start from byte 0, remember). For the third parameter you can either pass `SEEK_SET`, the default, which means "from the beginning of the file", `SEEK_CUR`, which means "relative to the current location", or `SEEK_END`, which means "from the end of the file".

# Example of fseek and rewind

```
<?php
$filename="test.txt";
    $handle = fopen($filename, "w+");
    fwrite($handle, "Mnnkyys");
    rewind($handle);
    fseek($handle, 1);
    echo ftell($handle)."<br>";
    fwrite($handle, "o");
    fseek($handle, 2, SEEK_CUR);
    echo ftell($handle)."<br>";
    fwrite($handle, "e");
    fclose($handle);
?>
```

Result:

Monkeys

Contents of test.txt

Hello

Welcome to php.

# Checking whether a file exists

## Syntax:

`bool file_exists ( string filename )`

you specify the filename to check as the only parameter, and it returns true if the file exists and false otherwise.

## Example:

```
<?php
if (file_exists("test1.txt"))
{
    print "test.txt exists!\n";
}
else
{
    print "test.txt does not exist!\n";
}
?>
```

## Result:

Test1.txt exists!

The same unlink function can be used along with directory handler to list and delete all the files present inside.

**Example:**

```
<?php
function EmptyDir($dir) {
    $handle=opendir($dir);

    while (($file = readdir($handle))!==false)
    {
        echo "$file <br>";
        @unlink($dir.'/'.$file);
    }

    closedir($handle);
}

EmptyDir('images');
?>
```



# PHP File delete by unlink function

## Syntax:

```
unlink($path);
```

Here *\$path* is the path of the file.

## Example:

```
<?php  
$path="test.txt";  
if(unlink($path))  
echo "Deleted file ";  
?>
```

## Result:

Deleted file

# Dissecting filename information

## Syntax:

```
array pathinfo ( string path)  
string basename ( string path [, string suffix])
```

➤ The *pathinfo()* function is designed to take a filename and return the same filename broken into various components. It takes a filename as its only parameter, and returns an array with three elements: **dirname**, **basename**, and **extension**. **Dirname** contains the name of the directory the file is in (e.g. "c:\windows" or "/var/www/public\_html"), **basename** contains the base filename (e.g. index.html, or somefile.txt), and **extension** contains the file extension if any (e.g. "html" or "txt").

### Example:

```
<?php
    $filename="test1.txt";
    $fileinfo = pathinfo($filename);
    var_dump($fileinfo);
    $filename = basename("/home/minal/somefile.txt");
    echo $filename."<br>";
    $filename = basename("/home/minal/somefile.txt", ".php");
    echo $filename."<br>";
    $filename = basename("/home/minal/somefile.txt", ".txt");
    echo $filename."<br>";
?>
```

### Result:

```
array(3) { ["dirname"]=> string(1) "." ["basename"]=> string(8) "test.txt"
["extension"]=> string(3) "txt" }
```

```
somefile.txt
somefile.txt
somefile
```

# Getting the last updated time of file

the last updated date of any file by using **filemtime** function

This function *filemtime()* uses the server file system so it works for local systems only, we can't use this function to get the modified time of any remote file system.

**Example:**

```
<?php  
echo date("m/d/Y",filemtime("test.txt"));  
?>
```

**Result:**

08/02/2010

**Note :** that we have used date function to convert the Unix Timestamp time returned by filemtime() function.

# Including other files in PHP

One of the most basic operations in PHP is including one script from another, thereby sharing functionality.

## 1) Include

The `include( )` statement includes and evaluates the specified file.

### Example:

`internal.php`

```
<?php  
$i = 100;  
?>
```

`test.php`

```
<?php  
echo "Value of i before include is: -$i-<br/>";  
include("internal.php");  
echo "Value of i after include is: -$i-<br/>";  
?>
```

### Output:

```
Value of i before include is: --  
Value of i after include is: -100-
```

## 2)Include Once

The `include_once( )` statement includes and evaluates the specified file during the execution of the script. This is a behavior similar to the `include( )` statement, with the only difference being that if the code from a file has already been included, it will not be included again. As the name suggests, it will be included just once.

**Example : test.php**

```
<?php
include_once("internal.php");
echo "Multiple include test.<br/>";
include_once("internal.php");
echo "Now include it again.<br/>";
include_once("internal.php");
echo "Multiple include test.<br/>";
include_once("internal.php");
?>
```

**Internal.php**

```
<?php $i = 100;
echo "Value of i after include is: -$i-<br/>"; ?>
```

**Output:**

```
Value of i after include is: -100-
Multiple include test.
Now include it again.
Multiple include test.
```

### 3) Require

require() and include() are identical in every way except how they handle failure. They both produce a Warning, but require() results in a Fatal Error. In other words, don't hesitate to use require() if you want a missing file to halt processing of the page.

#### Example:

```
<?php
echo "Try to include wrong file first with include.<br/>";
include ('internalwrong.php');
echo "Now try it with require.<br/>";
require ('internalwrong.php');
echo "Finish.";
?>
```

**Result:** Try to include wrong file first with include.

**Warning:** main(internal1.php): failed to open stream: No such file or directory in **c:\program files\easyphp1-8\www\test5.php** on line 3

**Warning:** main(): Failed opening 'internal1.php' for inclusion (include\_path='.;C:/Program Files/EasyPHP1-8\php\pear\') in **c:\program files\easyphp1-8\www\test5.php** on line 3

Now try it with require.

**Warning:** main(internal1.php): failed to open stream: No such file or directory in **c:\program files\easyphp1-8\www\test5.php** on line 5

**Fatal error:** main(): Failed opening required 'internal1.php' (include\_path='.;C:/Program Files/EasyPHP1-8\php\pear\') in **c:\program files\easyphp1-8\www\test5.php** on line 5

#### 4) **Require Once**

The `require_once()` statement includes and evaluates the specified file during the execution of the script. This is a behavior similar to the `require()` statement, with the only difference being that if the code from a file has already been included, it will not be included again.



# Type of Errors in PHP

1-Notices

2-Warnings

3-Fatal errors

## **Notices:**

These are trivial, non-critical errors. that does not terminate script .

Condition:

1- Accessing a variable that not define.

## **Warnings:**

These are more serious errors

Condition:

1-attempting to include() a file which does not exist.

## **Fatal errors:**

These are critical errors that terminate script and stop

Condition:

1-instantiating an object of a non-existent class

2- Calling a non-existent function

3-Missing semicolon

4-missing braces

5- Destroyed DOM

In PHP we can handle errors easily. Because when error comes it gives error line with their respective line and send error message to the web browser.

# Error Handling in PHP :

## 1) Stop error reporting completely in PHP:

To completely turn off error or warning messages on web pages you can use the function **"error\_reporting(0);"**

For example, here we try to open a file that does not exist, we will get a warning as below

**Example:**

```
<?php
fopen('test.txt',r);
echo("test for error message");
?>
```

**Result:**

**Warning:** fopen(test1.txt): failed to open stream: No such file or directory in  
**c:\program files\easyphp1-8\www\test7.php on line 2**  
test for error message

# Example of error\_reporting:

```
<?php  
error_reporting(0);  
fopen('test.txt',r);  
echo("test for error message");  
?>
```

**Result:**

test for error message

## 2) Use of Function ini\_set() :

We can also use ini\_set to turn off any error message being displayed. The property to be used is "display\_errors". Note that using ini\_set will overwrite all other related properties set using functions.

Example:

```
<?php
ini_set("display_errors",0);
.....
.....
.....
?>
```

To reset, use value as 1 instead of 0.

Some times we will have to block warning message for certain operations.

## 2) Stop error reporting for specific operation in PHP:

To stop error or warning messages on web pages for certain method or operation you can use the character "@"

Example:

```
<?php  
echo("test for error message 1");  
fopen(" test.txt " , " r ");  
echo("test for error message 2");  
include "test.php";  
?>
```

Result:

test for error message 1

test for error message 2

**Warning:** main(test.php): failed to open stream: No such file or directory in **c:\program files\easyphp1-8\www\test8.php** on line 5

**Warning:** main(): Failed opening 'test.php' for inclusion (include\_path='.;C:/Program Files/EasyPHP1-8\php\pear\') in **c:\program files\easyphp1-8\www\test8.php** on line 5

To restrict errors, we have to use the character @ before the function.  
i.e instead of **include....** we will use **@include....**

**Example :**

```
<?php  
echo("test for error message 1");  
fopen(" test.txt " , " r ");  
echo("test for error message 2");  
@include "test.php";  
?>
```

**Result:**

```
test for error message 1  
test for error message 2
```

Thus we can prevent the errors and warnings from being shown on the web page.

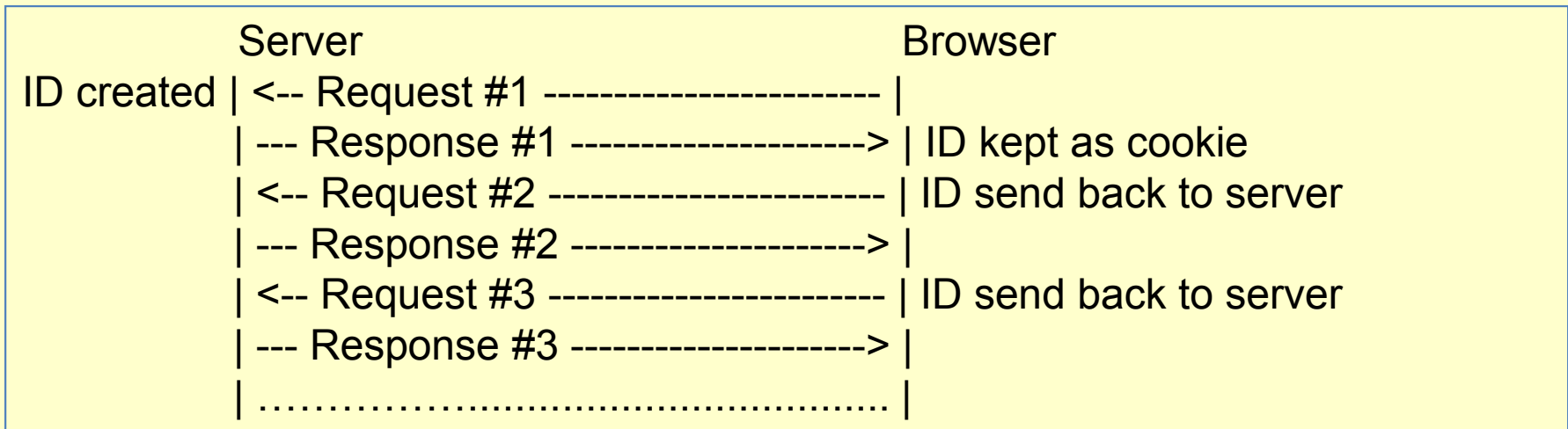
# Session Management in PHP

## What Is a Session?

- A session is an abstract concept to represent a series of HTTP requests and responses exchanged between a specific Web browser and a specific Web server. The session concept is very useful for Web based applications to pass and share information from one Web page (request) to another Web page (request).
- In short, A session is a combination of a server-side file containing all the data you wish to store, and a client-side cookie containing a reference to the server data

# Introduction:

The key design element of session support is about how to identify a session with an ID (identification) and how to maintain the session ID. One common way to maintain the session ID is use the cookie technology. The following diagram shows you how to do this:



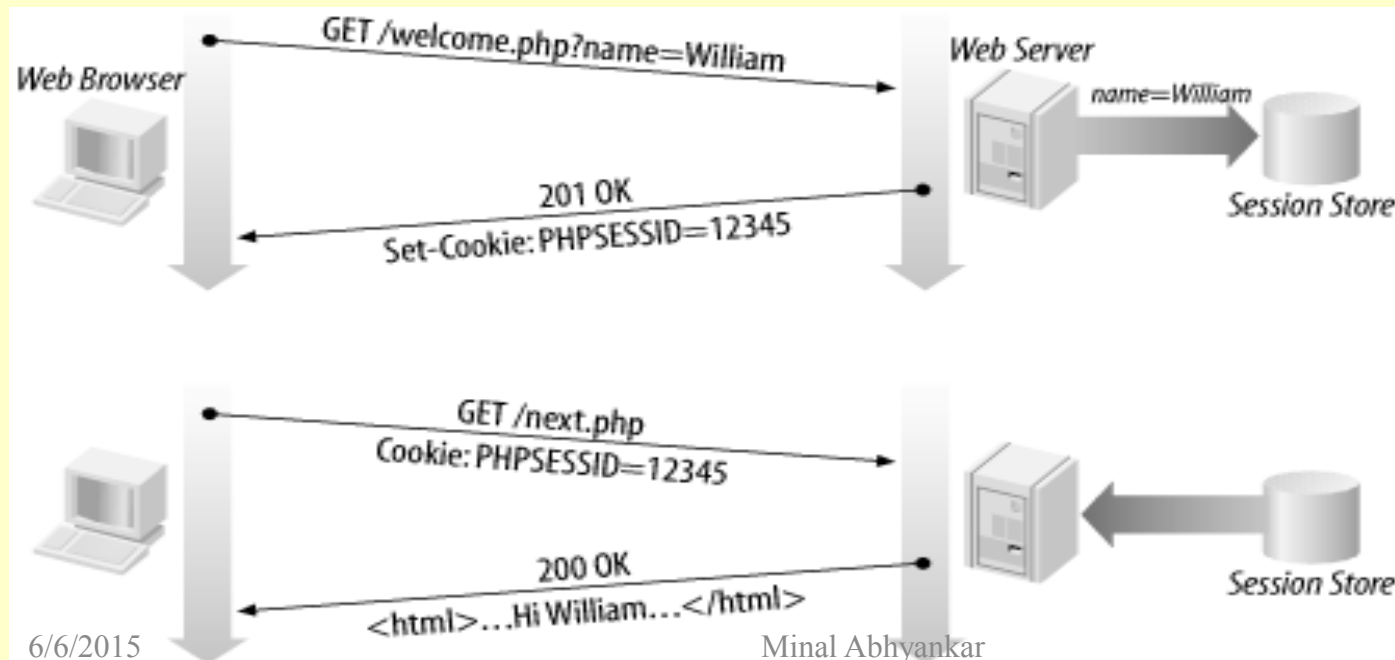
When the first request comes from a browser on a client host, the server should create a new session, and assigns a new session ID. The session ID will be then sent back to the same browser as a cookie. The browser will remember this ID, and send the ID back to the server in subsequent requests. When the server receives a request containing the same session ID, it knows that this request is a continuation of an existing session.



## Overview of Session Management

When a user first enters the session-based application by making a request to a page that starts a session, PHP generates a session ID and creates a file that stores the session-related variables. PHP sets a cookie to hold the session ID in the response the script generates. The browser then records the cookie and includes it in subsequent requests. In the example shown, the script *welcome.php* records session variables in the session store, and a request to *next.php* then has access to those variables because of the session ID.

### **The interaction between the browser and the server when initial requests are made to a session-based application**



# Session - continued

If there is no subsequent request coming back for a long time for a particular session ID, that session should be timed out. After the session has been timed out, if the browser comes back again with the same session ID, the server should give an invalid session error.

## How Sessions Are Support in PHP?

Session IDs are passed as cookies or GET/POST variables. `session_start()` is the built-in function to start a session. `$_SESSION` is the built-in array to manage session data.

**`session_start()`** - A built-in function to create a new session or resume an existing session. When `session_start()` is called, the PHP engine will check the current HTTP request to see if an existing session ID is included or not.

➤ If no session ID is found, the PHP engine will create a new session with a new session ID.

➤ If a session ID is found, the PHP engine will restore the session identified by this session ID. If the restored session has been timed out, an error will be issued.

# What is a Session Variable?

A session variable is a regular global variable that, when registered as a session variable, keeps its value on all pages that use PHP4 sessions. To register a session variable, assign a value to a variable that is to become a session variable and call **session\_register("variable\_name")**. On all subsequent pages that uses sessions (by calling **session\_start()**), the variable `variable_name` will have the value assigned to it before it was registered as a session variable. Changes to the variable value will be automatically registered in the session and saved for further reference.

**To set a session variable, use syntax like this:**

```
$_SESSION['var'] = $val;  
$_SESSION['FirstName'] = "Jim";
```

# Reading session data

Once you have put your data safely , it becomes immediately available in the `$_SESSION` superglobal array with the key of the variable name you gave it.

Example :

```
<?php
    $_SESSION['foo'] = 'bar';
    echo $_SESSION['foo'];
?>
```

Output:

bar

# Removing session data

Removing a specific value from a session is as simple as using the function *unset()*, just as you would for any other variable. It is important that you should unset only specific elements of the `$_SESSION` array, not the `$_SESSION` array itself, because that would leave you without any way to manipulate the session data at all.

Example :

```
<?php
    $_SESSION['foo'] = 'bar';
    echo $_SESSION['foo'];
    unset($_SESSION['foo']);
?>
```

# Ending session

- A session lasts until your visitor closes their browser.
- If you want to explicitly end a user's and delete their data without them having to close their browser, you need to clear the `$_SESSION` array, then use the `session_destroy()` function. *Session\_destroy()* removes all session data stored on your hard disk, leaving you with a clean slate.

## Example :

```
<?php
    session_start();
    $_SESSION = array();
    session_destroy();
?>
```

# Checking Session Data

You can check whether a variable has been set in a user's session using the function *isset()*, as you would a normal variable. Because the `$_SESSION` superglobal is only initialised once *session\_start()* has been called, you need to call *session\_start()* before using *isset()* on a session variable.

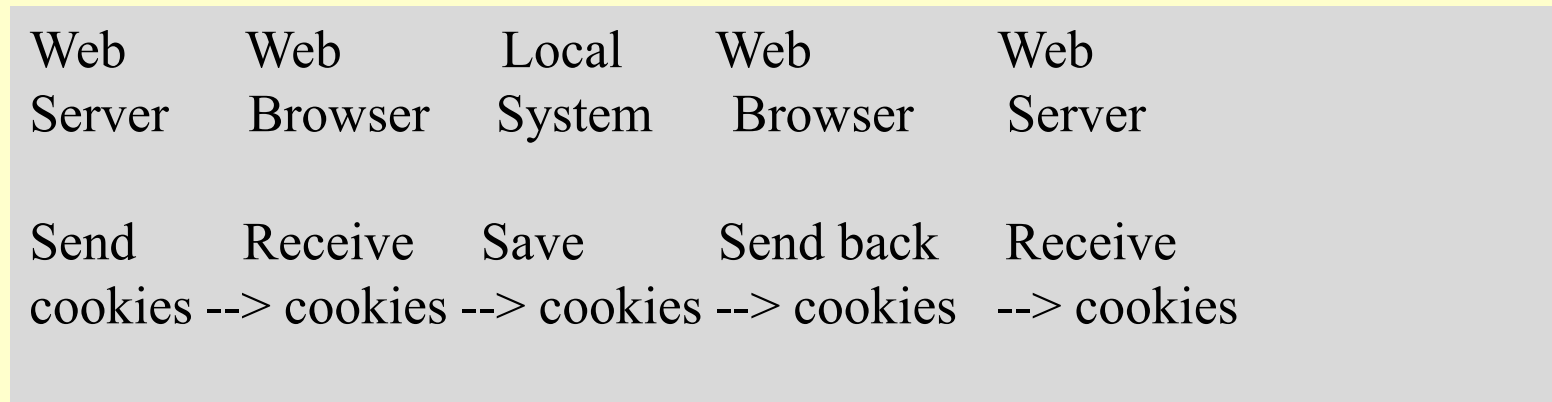
Example :

```
<?php
    session_start();
    if (isset($_SESSION['FirstName']))
    {
        /// your code here
    }
?>
```

# Cookie Management

## What Is a Cookie?

A cookie is a piece of information sent by a Web server to a Web browser, saved by the browser, and sent back to the server later. Cookies are transmitted inside the HTTP header. Cookies move from server to browser and back to server as shown in the following diagram:



As you can see from the diagram, cookies are actually saved by the local system in memory or on the hard disk of Web browser user's machines. **Cookies are mainly used to pass information from one PHP script to the next script.**



## PHP supports cookies with following elements:

1. `setcookie()` - A built-in function that defines a cookie to be sent in the HTTP response as a header line. Like other header lines, cookies must be sent before any output from your script (this is a protocol restriction). This requires that you place calls to this function prior to any output, including `<html>` and `<head>` tags as well as any whitespace. If output exists prior to calling this function, `setcookie()` will fail and return `FALSE`.

The syntax of calling `setcookie()` is:

`bool setcookie(string name, string value, int expire)`

➤ "name" is the name of this cookie.

➤ "value" is the value of this cookie. "value" is optional. If "value" is not provided, this cookie will be set in the HTTP response without any value.

➤ "expire" is the time when this cookie should be expired. "expire" is optional. If "expire" is not provided, this cookie will be saved in browser memory only. If "expire" is provided, it represents a time in number of seconds since the epoch. If the provided time is a future time, this cookie will be saved on the hard disk of the browser system.

2. **`$_COOKIE`** - A built-in array that stores cookies returned back from the browser in the next HTTP request. Cookie names will be used as array keys.

```
<?php
$numCookies = count( array_keys($_COOKIE) );
$numCookies++;
$cookieName = "Cookie_$numCookies";
$cookieValue = "My cookie value";
setcookie($cookieName, $cookieValue);
echo("Cookies added by the server:");
echo "<br>";
echo(" $cookieName: $cookieValue");
echo "<br>";
echo("\Cookies received by the server:"); echo "<br>";
foreach ($_COOKIE as $k => $v)
{
    echo " $k = $v\n"; echo "<br>";
}
echo "<br>";?>
```

# Database Concepts Refresher

## Database :

A database is a collection of data that is organized so that its contents can be easily accessed, managed and updated. The software used to manage and query a database is known as a **Database Management System (DBMS)**. Then came the concept of **Relational Database Management System(RDBMS)**.

Relational database is a database where data are stored in more than one table, each one containing different types of data. The different tables can be linked so that information from the separate files can be used together.

# Database Concepts Refresher

## ➤ Entity: an object, concept, or event.

Examples of an entity include real world object like a Vehicle, Employee, Order, or events like a Service Outage or a System Error. It's the “thing” or set of things that a database represents, and the real-world objects around which businesses are based.

## ➤ Field (column): describes a single characteristic of an entity.

Examples of Vehicle fields include properties of the Vehicle entity, such as the Vehicle year, make, model, and mileage.

# Database Concepts Refresher

➤ Record (row): collection of fields (characteristics) describing each entity.

A record represents all the fields that describe the entity. If our database contains 3000 Vehicles, then it will have 3000 rows of vehicles, each with their own distinct values in each field.

➤ Table: collection of records for a specific entity.

A table is used to keep records for one entity separate from other entities, so that Vehicle records are stored separately from Employee records. In some ways, you can think of a table as a spreadsheet containing multiple rows and columns all related to the same entity.

# Database Concepts Refresher

## ➤ Database: collection of tables.

MySQL server contains multiple databases, which is common for database servers. Each database usually contains tables that are specific to an application or business group.

## MySQL

➤ Open Source relational database management system:

[www.mysql.com](http://www.mysql.com)

➤ Supports Structured Query Language (SQL) – a standardized way to communicate with databases.

➤ Very popular

# Navigating in MySQL

- Connecting to MySQL through command prompt

`c:\mysql\bin>mysql`

- Get a list of existing databases:

- `mysql> show databases;`

- Specify the database to use:

- `mysql> use mysql;` (mysql is the name of database)

- Get a list of tables in the database:

- `mysql> show tables;`

- Describe a table:

- `mysql> describe user;`

# Navigating in MySQL

## ➤ Get a name of user:

- `mysql> select user( );`

## ➤ Get a date and time:

- `mysql> select now( );`

## ➤ Get a version :

- `mysql> select version( );`

## ➤ Get a database :

- `mysql> select database( );`



# Navigating in MySQL

➤ Can combine all in one as follow

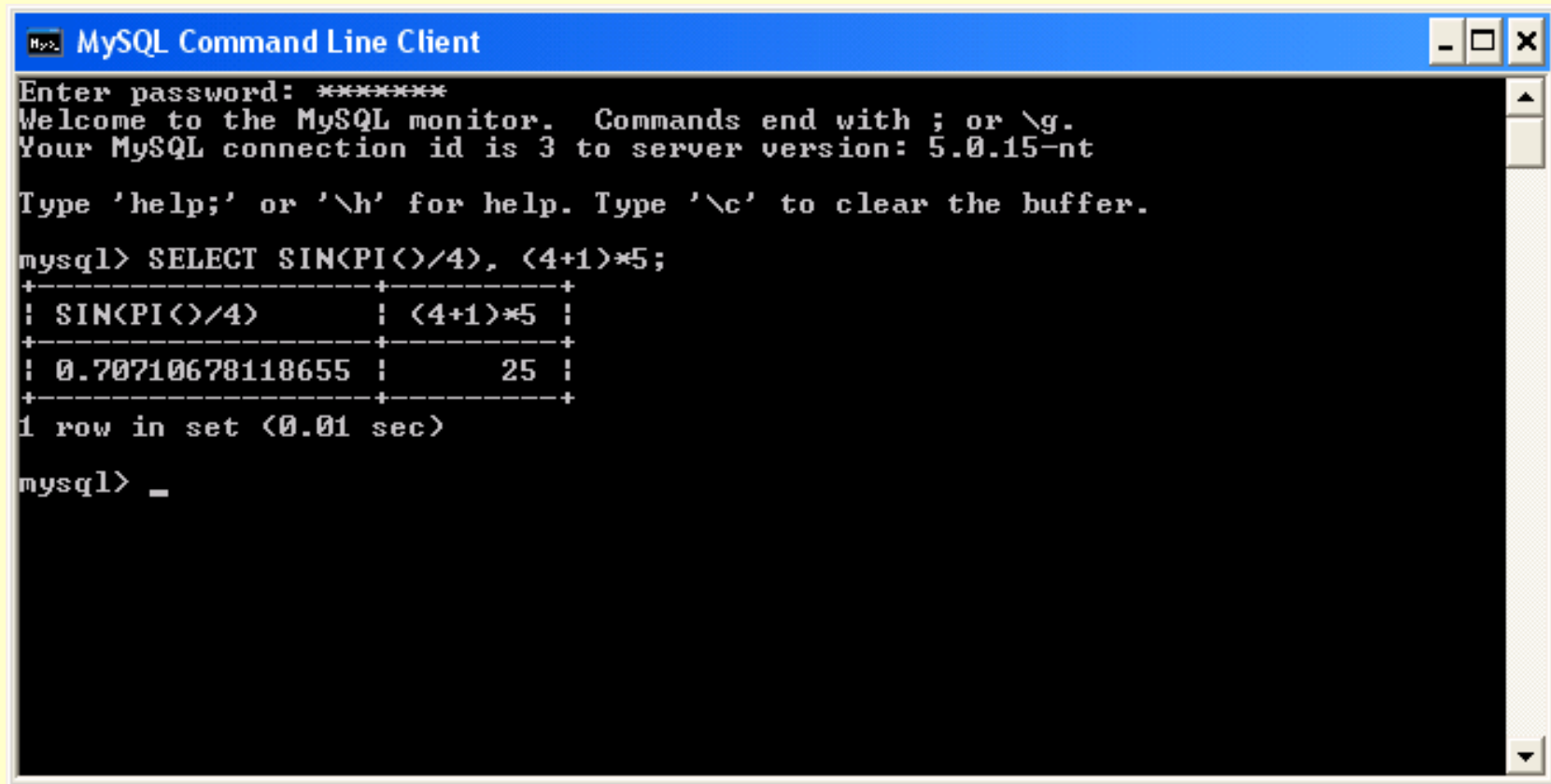
▪ `mysql> select user( ), now(), version(), database();`

□ Output :

<code>user( )</code>	<code>now( )</code>	<code>version( )</code>	<code>database( )</code>
localhost	2003-01-05 21:24:27	4.0.1-alpha-nt	test

# Entering Queries

- Here's another query. It demonstrates that you can use **mysql** as a simple calculator



The screenshot shows a window titled "MySQL Command Line Client". The text inside the window is as follows:

```
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 5.0.15-nt

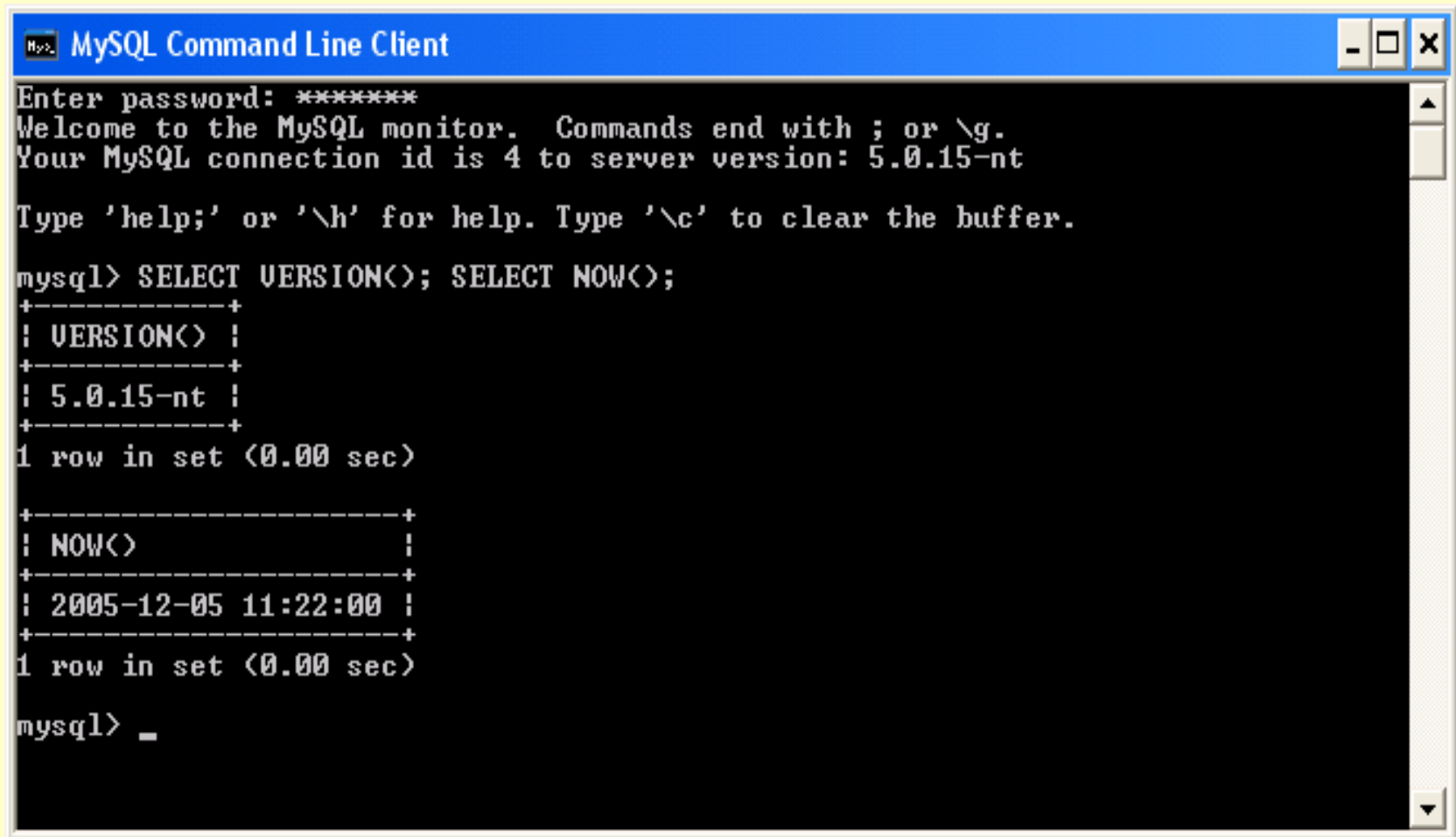
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT SIN(PI()/4), (4+1)*5;
+-----+-----+
| SIN(PI()/4) | (4+1)*5 |
+-----+-----+
| 0.70710678118655 |      25 |
+-----+-----+
1 row in set (0.01 sec)

mysql> _
```

# Entering Queries

- You can even enter multiple statements on a single line. Just end each one with a semicolon

A screenshot of the MySQL Command Line Client window. The title bar is blue and says "MySQL Command Line Client". The main area is black with white text. It shows the prompt "Enter password: \*\*\*\*\*", a welcome message, and connection details. Then, a query "mysql> SELECT VERSION(); SELECT NOW();" is entered. The output shows the MySQL version "5.0.15-nt" and the current timestamp "2005-12-05 11:22:00". The prompt "mysql> \_" is shown at the bottom.

```
MySQL Command Line Client
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4 to server version: 5.0.15-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT VERSION(); SELECT NOW();
+-----+
| VERSION() |
+-----+
| 5.0.15-nt |
+-----+
1 row in set (0.00 sec)

+-----+
| NOW() |
+-----+
| 2005-12-05 11:22:00 |
+-----+
1 row in set (0.00 sec)

mysql> _
```

# MySQL CRUD Operations

## ➤ Create a new database:

```
mysql> CREATE DATABASE cis;
```

## ➤ Create a new table:

```
mysql> use cis;
```

```
-> CREATE TABLE vehicles
```

```
-> (VIN varchar(17), Year int(4), Make varchar(20),
```

```
-> Model varchar(20), Mileage int(10));
```

## ➤ Insert data into the table:

```
mysql> INSERT INTO vehicles (VIN, Year, Make, Model,  
Mileage) VALUES ('8YTR754', 2002, 'Ford', 'Mustang', 21000);
```

## ➤ Create a SQL script describing the database:

```
C:\mysql\bin> mysqldump -u root -p cis > cis.sql
```

## ➤ Create a database and data using a SQL script:

```
mysql> source C:\mysql\bin\cis.sql (NOTE: no semi-colon)
```

# MySQL CRUD Operations

➤ Retrieving all data from a table:

```
mysql> SELECT * FROM vehicles;
```

Output:

VIN	Year	Make	Model	Mileage
8YTR754	2002	Ford	Mustang	21000
4GKU390	2001	Chevrolet	Corvette	35000
92A84UJ	1998	Dodge	Viper	89256

➤ Selecting a specific row of data:

```
mysql> SELECT * FROM vehicles WHERE VIN = '8YTR754';
```

Output :

VIN	Year	Make	Model	Mileage
8YTR754	2002	Ford	Mustang	21000

# MySQL CRUD Operations

## ➤ Update all records in a table

```
mysql> UPDATE vehicles SET Mileage = 25000;
```

## ➤ Update specific records

```
mysql> UPDATE vehicles SET Mileage = 25000  
-> WHERE VIN = '8YTR754';
```

## ➤ Update multiple columns of a specific record

```
mysql> UPDATE vehicles  
-> SET Mileage = 25000, LastServiceDate = '08/30/2003'  
-> WHERE VIN = '8YTR754';
```

# MySQL CRUD Operations

- Delete all rows in a table (does not delete table)

```
mysql> DELETE FROM vehicles;
```

- Delete specific rows in a table

```
mysql> DELETE FROM vehicles  
-> WHERE VIN = '8YTR754';
```

- Delete the entire table (remove from database)

```
mysql> DROP TABLE vehicles;
```

# Data types :

**Definition :** Data type is the characteristic of columns and variables that defines what types of data values they can store. The characteristic indicating whether a data item represents a number, date, character string, etc.

Data types are used to indicate the type of the field we are creating into the table. MySQL supports a number of datatypes in three important categories:

- **Numeric types**
- **Date and Time types**
- **String(Character) types**



# MySQL Numeric Datatypes :

The numeric data types are as follows:

- BIT
- TINYINT
- BOOLEAN
- SMALLINT
- MEDIUMINT
- INT
- INTEGER
- BIGINT
- FLOAT
- DOUBLE
- DECIMAL

# Date and Time Data Types :

- DATE
- TIME
- DATETIME
- TIMESTAMP
- YEAR

# String data types :

- CHAR
- VARCHAR
- TINYTEXT
- TEXT
- BLOB
- MEDIUMTEXT
- LONGTEXT
- BINARY
- VARBINARY
- ENUM
- SET

## Numeric Datatypes :

### BIT :

BIT is a synonym for **TINYINT(1)**.

### TINYINT[(M)] :

A very small integer. The signed range is **-128** to **127**. The unsigned range is **0** to **255**.

### BOOL, BOOLEAN :

These types are synonyms for **TINYINT(1)**. A value of zero is considered false. Non-zero values are considered true.

### SMALLINT :

A small integer. The signed range is **-32768** to **32767**. The unsigned range is **0** to **65535**.

### MEDIUMINT :

A medium-sized integer. The signed range is **-8388608** to **8388607**. The unsigned range is **0** to **16777215**.

### INT :

A normal-size integer. The signed range is **-2147483648** to **2147483647**. The unsigned range is **0** to **4294967295**.

### INTEGER :

This type is a synonym for **INT**.

## Numeric Datatypes (continued):

### **BIGINT :**

A large integer. The signed range is **-9223372036854775808** to **9223372036854775807**. The unsigned range is **0** to **18446744073709551615**.

### **FLOAT :**

A small(single-precision) floating-point number. The values are from **3.402823466E+38** to **-1.175494351E-38**, **0**, and **1.175494351E-38** to **3.402823466E+38**.

### **DOUBLE :**

A normal-size(double-precision) floating-point number. The values are from **1.7976931348623157E+308** to **-2.2250738585072014E-308**, **0**, and **2.2250738585072014E-308** to **1.7976931348623157E+308**.

### **DECIMAL :**

The maximum number of digits(M) for **DECIMAL** is **64**.

# Date and Time Data Types :

## DATE :

A Date. The range is **1000-01-01** to **9999-12-31**. The date values are displayed in **YYYY-MM-DD** format.

## TIME :

A Time. The range is **-838:59:59** to **838:59:59**. The time values are displayed in **HH:MM:SS** format.

## DATETIME :

A Date and Time combination. The range is **1000-01-01 00:00:00** to **9999-12-31 23:59:59**. The datetime values are displayed in **YYYY-MM-DD HH:MM:SS** format.

## TIMESTAMP :

A Timestamp. The range is **1970-01-01 00:00:01** UTC to partway through the year **2037**. A **TIMESTAMP** column is useful for recording the date and time of an **INSERT** or **UPDATE** operation.

## YEAR :

A Year. The year values are displayed either in two-digit or four-digit format. The range of values for a four-digit is **1901** to **2155**. For two-digit, the range is **70** to **69**, representing years from **1970** to **2069**.

For all the date and time columns, we can also assign the values using either string or numbers.

# String data types

## **CHAR() :**

It is a fixed length string and is mainly used when the data is not going to vary much in it's length. It ranges from **0** to **255** characters long. While storing **CHAR** values they are right padded with spaces to the specified length. When retrieving the **CHAR** values, trailing spaces are removed.

## **VARCHAR() :**

It is a variable length string and is mainly used when the data may vary in length. It ranges from **0** to **255** characters long. **VARCHAR** values are not padded when they are stored.

## **TINYTEXT, TINYBLOB :**

A string with a maximum length of **255** characters.

## **TEXT :**

**TEXT** columns are treated as character strings(non-binary strings). It contains a maximum length of **65535** characters.

## **BLOB :**

**BLOB** stands for **B**inary **L**arge **O**bject. It can hold a variable amount of data. **BLOB** columns are treated as byte strings(binary strings). It contains a maximum length of **65535** characters.

# String data types (continued..)

## **MEDIUMTEXT, MEDIUMBLOB :**

It has a maximum length of **16777215** characters.

## **LONGTEXT, LONGBLOB :**

It has a maximum length of **4294967295** characters.

## **BINARY :**

The **BINARY** is similar to the **CHAR** type. It stores the value as binary byte strings instead of non-binary character strings.

## **VARBINARY :**

The **VARBINARY** is similar to the **VARCHAR** type. It stores the value as binary byte strings instead of non-binary character strings.

## **ENUM() :**

An enumeration. Each column may have one of a specified possible values. It can store only one of the values that are declared in the specified list contained in the ( ) brackets. The **ENUM** list ranges up to **65535** values.



## String data types (continued..)

### SET() :

A set. Each column may have more than one of the specified possible values. It contains up to **64** list items and can store more than one choice. **SET** values are represented internally as integers.

If **CHAR** and **VARCHAR** options are used in the same table, then MySQL will automatically change the **CHAR** into **VARCHAR** for compatability reasons. The ( ) bracket allows to enter a maximum number of characters that will be used in the column.

```
CREATE TABLE `customers` (  
  `customerNumber` int(11) NOT NULL,  
  `customerName` varchar(50) NOT NULL,  
  `contactLastName` varchar(50) NOT NULL,  
  `contactFirstName` varchar(50) NOT NULL,  
  `phone` varchar(50) NOT NULL,  
  `addressLine1` varchar(50) NOT NULL,  
  `addressLine2` varchar(50) default NULL,  
  `city` varchar(50) NOT NULL,  
  `state` varchar(50) default NULL,  
  `postalCode` varchar(15) default NULL,  
  `country` varchar(50) NOT NULL,  
  `salesRepEmployeeNumber` int(11) default NULL,  
  `creditLimit` double default NULL,  
  PRIMARY KEY (`customerNumber`)
```

## Customers

customerNumber int(11) NOT NULL,  
customerName varchar(50) NOT NULL,  
contactLastName varchar(50) NOT NULL,  
contactFirstName varchar(50) NOT NULL,  
Phone varchar(50) NOT NULL,  
addressLine1 varchar(50) NOT NULL,  
addressLine2 varchar(50) default NULL,  
city varchar(50) NOT NULL,  
state varchar(50) default NULL,  
postalCode varchar(15) default NULL,  
country varchar(50) NOT NULL,  
salesRepEmployeeNumber int(11) default NULL,  
creditLimit` double default NULL,

```
CREATE TABLE `orders`  
  `orderNumber` int(11) NOT NULL,  
  `orderDate` datetime NOT NULL,  
  `requiredDate` datetime NOT NULL,  
  `shippedDate` datetime default NULL,  
  `status` varchar(15) NOT NULL,  
  `comments` text,  
  `customerNumber` int(11) NOT NULL,  
  PRIMARY KEY (`orderNumber`)
```

# Using MySQL with PHP

## Connecting to a MySQL database

To connect to MySQL, there are two key functions: *mysql\_connect()* and *mysql\_select\_db()*.

Example:

```
<?php
    mysql_connect("localhost", "phpuser", "alm65z");
    mysql_select_db("phpdb");
?>
```

# Example for creating database

```
<?php
$host = "localhost"; // your host name
$username = "yourusername"; // your user name to access MySQL
$password = "yourpassword"; // your password to access MySQL
$database = "yourdatabase"; // The name of the database

if (!$link = @mysql_connect($host,$username,$password))
{
    die('Could not connect:'. mysql_error());
}
$sql = 'CREATE DATABASE '.$database;
if (mysql_query($sql, $link))
{
    echo "Database".$database."created successfully\n";
}
else
{
    echo 'Error creating database: ' . mysql_error() . "\n";
}
mysql_close($link); ?>
```

# Example for creating table

```
<?php
$host = "localhost"; // your host name
$username = "yourusername"; // your user name to access MySQL
$password = "yourpassword"; // your password to access MySQL
$database = "yourdatabase"; // The name of the database
if (!$link = @mysql_connect($host,$username,$password))
{die('Could not connect:'. mysql_error()); }
@mysql_select_db ($database) or die( "Unable to select database");
$sql='CREATE TABLE `books` (
`id` INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
`title` VARCHAR(255) NOT NULL,
`author` VARCHAR(150) NOT NULL,
`category` VARCHAR(45) NOT NULL,
`price` VARCHAR(20) NOT NULL,
PRIMARY KEY (`id`)
) ‘;
mysql_query($sql);
mysql_close();
?>
```

# Example for inserting in a table

```
<?php
$host = "localhost"; // your host name
$username = "root"; // your user name to access MySQL
$password = ""; // your password to access MySQL
$dbase = "try"; // The name of the database

// $link = mysql_connect($host,$username,$password);
if (!$link = @mysql_connect($host,$username,$password,true))
{die('Could not connect:'. mysql_error()); }

@mysql_select_db($dbase) or die( "Unable to select database");

$sql="INSERT INTO books ( title, author, category, price) VALUES ('Advanced
Database Technology and Design', 'Mario Piattini and Oscar Diaz', 'Database','$
89.00' ) ";
if(!$result = @mysql_query($sql,$link ))
{
echo('Could not insert:'.@mysql_error());
return false;
}
$rows_affected = mysql_affected_rows();
echo $rows_affected; ?>
```



# Inserting multiple records in a table

```
<?php
    $host = "localhost"; // your host name
    $username = "root"; // your user name to access MySql
    $password = ""; // your password to access MySql
    $database = "try"; // The name of the database
    //$link = mysql_connect($host,$username,$password);
    if (!$link = @mysql_connect($host,$username,$password,true))
    {die('Could not connect:'. mysql_error()); }
    @mysql_select_db($database) or die( "Unable to select database");
    $sql="INSERT INTO books ( title, author, category, price) VALUES ('Advanced
    Database Technology and Design', 'Mario Piattini and Oscar Diaz', 'Database','$
    89.00' ),
    ('Conceptual Database Design', 'Carol Batini Shamkant Navathe', 'Database', '$
    96.80'), ('A Guide to MySQL' , 'Philip J.Pratt', 'MySql', '$ 55.35'),
    ('Beginning Databases with MySQL', 'Richard Stones', 'MySql', '$ 8.90)";
    if(!$result = @mysql_query($sql,$link )){
    echo('Could not insert:'.@mysql_error());
    return false;}
    $rows_affected = mysql_affected_rows();
    echo $rows_affected;    ?>
```

# Example for retrieving data from a table

```
<?php
$host = "localhost"; // your host name
$username = "root"; // your user name to access MySql
$password = ""; // your password to access MySql
$database = "try"; // The name of the database

// $link = mysql_connect($host,$username,$password);
if (!$link = @mysql_connect($host,$username,$password,true))
{
    die('Could not connect:'. mysql_error());
}
@mysql_select_db($database) or die( "Unable to select database");
$sql="SELECT * FROM books";

$result = mysql_query($sql);
if (!$result)
{
    die('Invalid query: ' . mysql_error());
}
```

# Example for retrieving data from a table

```
while($row=mysql_fetch_array($result))
{
echo $row['title'].", ";
echo $row['author'].", ";
echo $row['category'].", ";
echo $row['price']. "<br>";
}
mysql_free_result($result);
?>
```

## OUTPUT:

Advanced Database Technology and Design, Mario Piattini and Oscar Diaz,  
Database, \$ 89.00  
Conceptual Database Design, Carol Batini Shamkant Navathe, Database, \$ 96.80  
A Guide to MySQL, Philip J.Pratt, MySql, \$ 55.35  
Beginning Databases with MySQL, Richard Stones, MySql, \$ 8.90

# Example for Updating data from a table

```
<?php
$host = "localhost"; // your host name
$username = "root"; // your user name to access MySql
$password = ""; // your password to access MySql
$dbase = "try"; // The name of the database

if (!$link = @mysql_connect($host,$username,$password,true))
{die('Could not connect:'. mysql_error()); }
@mysql_select_db($dbase) or die( "Unable to select database");

//echo $rows_affected;

$sql="UPDATE books SET price ='$ 95.2' WHERE id=1" ;

$result = mysql_query($sql);

if (!$result) {
die('Invalid query: ' . mysql_error());
}
$rows_affected=mysql_affected_rows();
echo $rows_affected;
?>
```

# Example for Deleting data from a table

```
<?php
```

```
$host = "localhost"; // your host name
```

```
$username = "root"; // your user name to access MySql
```

```
$password = ""; // your password to access MySql
```

```
$database = "try"; // The name of the database
```

```
// $link = mysql_connect($host,$username,$password);
```

```
if (!$link = @mysql_connect($host,$username,$password,true))
```

```
{die('Could not connect:'. mysql_error()); }
```

```
@mysql_select_db($database) or die( "Unable to select database");
```

```
//echo $rows_affected;
```

```
$sql="DELETE FROM books WHERE title='Advanced Database Technology and  
Design ' ;
```

```
$result = mysql_query($sql);
```

```
if (!$result) { die('Invalid query: ' . mysql_error());}
```

```
$rows_affected=mysql_affected_rows();
```

```
echo $rows_affected;
```

```
?>
```

# SERVER Variables in PHP

Server Variables are those variables which are inside the super global array named `$_SERVER` available in PHP. There are many server variables in PHP. Some of them which are used more commonly are

1) **`$_SERVER['REQUEST_URI']`** - It return the URL in to access the page which is executing the script. If you need to type

`http://www.example.com/product.php?id=5` to access the page then

`$_SERVER['REQUEST_URI']` returns `"/product.php?id=5"`.

2) **`$_SERVER['DOCUMENT_ROOT']`** – Returns the root directory of the server which is specified in the configuration file of server. This variable usually returns the path like `"/usr/yoursite/www"` in Linux and `"D:/xamps/xampp/htdocs"` in windows.

3) **`$_SERVER['HTTP_HOST']`** – Returns the host's name as found in the http header. This variable usually returns the path like `"example.com"` when the you find `"http://example.com"` in browser's address-bar and return `"www.example.com"` when you see `http://www.example.com` in the address-bar. This is quite useful when you've to preserve session while making online payment using PHP since session stored for `"http://example.com"` is not same as for the `http://www.example.com`.

4) **`$_SERVER['HTTP_USER_AGENT']`** - Returns the user agent's (browser) detail accessing the web page. We can use `strpos($_SERVER["HTTP_USER_AGENT"],"MSIE")` to detect Microsoft Internet explorer or you can use `strpos($_SERVER["HTTP_USER_AGENT"],"Firefox")` to detect firefox browser in PHP.

5) **`$_SERVER['PHP_SELF']`** - Returns the file-name of the currently executing script. Let's suppose that you're accessing the URL `http://www.example.com/product.php?id=5` then `$_SERVER['PHP_SELF']` returns `"/product.php"` in your script.

6) **`$_SERVER['QUERY_STRING']`** – Returns the query string if query string is used to access the script currently executing. Query strings are those string which is available after “?” sign. if you use `$_SERVER['QUERY_STRING']` in the script executing the following URL

`“http://www.example.com/index.php?id=5&page=product”` then it returns `“id=5&page=product”` in your script.

7) **`$_SERVER['REMOTE_ADDR']`** – Returns the IP address of remote machine accessing the current page.

8) **`$_SERVER['SCRIPT_FILENAME']`** - Returns the absolute path of the file which is currently executing. It returns path like `“var/example.com/www/product.php”` in Linux and path like `“D:/xampp/xampp/htdocs/test/example.php”` in windows.

# EXAMPLE to list server variables in PHP

```
<?php
echo "list of server variables in php“.”<br>”;
foreach($_SERVER as $key=>$value)
{
    echo $key ." = " . $value."<br>";
}
?>
```



# Regular Expression

- A regular expression is a specially formatted pattern that can be used to find instances of one string in another.
- Several programming languages including Visual Basic, Perl, JavaScript and PHP support regular expressions.
- Regular expressions take a lot of the hassle out of writing lines and lines of repetitive code to validate a string

**There are 2 types of regular expressions:**

- POSIX Extended
- Perl Compatible

**PHP has six functions that work with regular expressions.** They all take a regular expression string as their first argument, and are shown below:

- **ereg:** The most common regular expression function, ereg allows us to search a string for matches of a regular expression.
- **ereg\_replace:** Allows us to search a string for a regular expression and replace any occurrence of that expression with a new string.
- **eregi:** Performs exactly the same matching as ereg, but is case insensitive.
- **eregi\_replace:** Performs exactly the same search-replace functionality as ereg\_replace, but is case insensitive.
- **split:** Allows us to search a string for a regular expression and returns the matches as an array of strings.
- **spliti:** Case insensitive version of the split function

# ereg()

- `ereg(regex, string)` searches for the pattern described in *regex* within the string *string*.
- It returns false if no match was found.
- If you call the function as `ereg(regex, string, matches)` the matches will be stored in the array *matches*. Thus *matches* will be a numeric array of the grouped parts (something in ()) of the string in the string. The first group match will be `$matches[1]`.

# ereg\_replace

- `ereg_replace ( regex, replacement, string )` searches for the pattern described in *regex* within the string *string* and replaces occurrences with *replacement*. It returns the replaced string.
- If *replacement* contains expressions of the form `\\number`, where *number* is an integer between 1 and 9, the number sub-expression is used.
  - `$better_order=ereg_replace('glass of (Karlsberg|Bruch)', 'pitcher of \\1',$order);`

# split()

- `split(regex, string, [max])` splits the string *string* at the occurrences of the pattern described by the regular expression *regex*. It returns an array. The matched pattern is not included.
- If the optional argument *max* is given, it means the maximum number of elements in the returned array. The last element then contains the unsplit rest of the string *string*.
- Use `explode()` if you are not splitting at a regular expression pattern. It is faster.

# case-insensitive function

- `eregi()` does the same as `ereg()` but work case-insensitively.
- `eregi_replace()` does the same as `ereg_replace()` but work case-insensitively.
- `spliti()` does the same as `split()` but work case-insensitively.

## Example of Regular Expression: (POSIX Extended )

```
<?php
function validateEmail($email)

{

return ereg("^([a-zA-Z]+@[a-zA-Z]+\.[a-zA-Z]+$", $email);

}

echo validateEmail("minal@gmail.com");

?>
```

A list of some of the most common regular expressions, as well as an example of how to use each one:

### **Beginning of string:**

**To search from the beginning of a string, use ^.**

For example,

```
<?php echo ereg("^hello", "hello world!"); ?>
```

Would return true, however

```
<?php echo ereg("^hello", "i say hello world!"); ?>
```

would return false, because hello wasn't at the beginning of the string.

### **End of string:**

**To search at the end of a string, use \$.**

For example,

```
<?php echo ereg("bye$", "goodbye"); ?>
```

Would return true, however

```
<?php echo ereg("bye$", "goodbye my friend"); ?>
```

would return false, because bye wasn't at the very end of the string.



## Any single character:

To search for any character, use the dot.

For example,

```
<?php echo ereg(".", "cat"); ?>
```

would return true, however

```
<?php echo ereg(".", ""); ?>
```

would return false, because our search string contains no characters. You can optionally tell the regular expression engine how many single characters it should match using curly braces. If I wanted a match on five characters only, then I would use ereg like this:

```
<?php echo ereg(".{5}$", "12345"); ?>
```

The code above tells the regular expression engine to return true if and only if at least five successive characters appear at the end of the string.

## Repeat character zero or more times

To tell the regular expression engine that a character may exist, and can be repeated, we use the **\*** character. Here are two examples that would return true:

```
<?php echo ereg("t*", "tom"); ?>
```

```
<?php echo ereg("t*", "fom"); ?>
```

Even though the second example doesn't contain the 't' character, it still returns true because the \* indicates that the character may appear, and that it doesn't have to. In fact, any normal string pattern would cause the second call to ereg above to return true, because the 't' character is optional.

## Repeat character one or more times

To tell the regular expression engine that a character must exist and that it can be repeated more than once, we use the **+** character, like this:

```
<?php echo ereg("z+", "i like the zoo"); ?>
```

The following example would also return true:

```
<?php echo ereg("z+", "i like the zzzzzzoo!"); ?>
```

## Repeat character zero or one times

We can also tell the regular expression engine that a character must either exist just once, or not at all. We use the **?** character to do so.

```
<?php echo ereg("c?", "cats are fuzzy"); ?>
```

If we wanted to, we could even entirely remove the 'c' from the search string shown above, and this expression **would still return true**. The '?' means that a 'c' **may** appear anywhere in the search string, but doesn't have to.

## The space character

To match the space character in a search string, we use the predefined **Posix class, [[:space:]]**. The square brackets indicate a related set of sequential characters, and ":space:" is the actual class to match.

We can match a single space character like this:

```
<? Php echo ereg("Minal[[:space:]]Abhyankar", "Minal Abhyankar"); ?>
```

# Example of Regular Expression :

In case of a registration form you may want to control available user names a bit. Let's suppose you don't want to allow any special character in the name except "\_.-" and of course letters and numbers. Besides this you may want to control the length of the user name to be between 4 and 20.

## Example of Regular Expression : (Perl Compatible)

```
$pattern = '/^[a-zA-Z0-9_.-]{4,20}$/';
```

```
$username = "this.is.a-demo_-";
```

```
if (preg_match($pattern,$username))
```

```
echo "Match";
```

```
else
```

```
echo "Not match";
```

## Example of Regular Expression

```
<?php
$string = 'abcefg hijklmnopqrstuvwxyz0123456789';

// match any character that is not a number between 0 and 9
preg_match_all("/[^0-9]/", $string, $matches);

// loop through the matches with foreach
foreach($matches[0] as $value)
{
    echo $value;
}

echo "<hr color=red>";
//to split the string by any number of commas or space characters:
$keywords = preg_split("/[s,]+/", "php,regular expressions");
print_r( $keywords );

echo "<hr color=red>";
echo preg_replace("/([Cc]opyright) 200(3|4|5|6)/", "$1 2007", "Copyright
2005");
//change the date format from "yyyy-mm-dd" to "mm/dd/yyyy":
echo "<hr color=red>";
```

```

echo preg_replace("/(\d+)-(\d+)-(\d+)/", "$2/$3/$1", "2007-01-25");
echo "<hr color=red>";
function validateEmail($pattern,$username)
{
if (preg_match($pattern,$username))
echo "Match";
else
echo "Not match";
}
$pattern="/^[a-zA-Z]+@[a-zA-Z]+\.[a-zA-Z]+$"/;
echo validateEmail($pattern,"minal@gmail.com");
echo "<hr color=red>";
$password = "Fyfjk34sdfjfsjq7";

if (preg_match("/^.*(?=.{8,})(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).*$/", $password)) {
    echo "Your passwords is strong.";
} else {
    echo "Your password is weak.";
}
?>

```