1. Website

Website is a collection of related WebPages and media content which can be accessed through network. In WAN websites can be accessed using internet protocols (IP) and in LAN Websites can be accessed using URL-uniform resource locator.

Functions of website

Main function of website is to provide particular information using network and internet worldwide. Various types of information can be provided worldwide users by website, like textual information, graphical information and some information like audio and video. Now a day's complex map and rout information are also available over the internet.

Examples of websites

Informative websites

a. www.Yahoo.com
b. www.msn.com
c. www.rediffmail.com

Search engine websites

a. www.google.com
b. www.bing.com
c. www.ask.com

Ecommerce websites

a. www.amazon.com
b. www.flipkart.com
c. www.uber.com

Job portal websites

a. https://ojas.gujarat.gov.in
b. www.monsterjob.com
c. www.jobs.com

Government websites

a. www.Incometaxindiaefiling.com
b. www.Uidai.gov.in
c. www.Digitalgujarat.gov.in

Now a day's companies and individuals also have their websites to deliver information using this technology.

2. Server side Scripting Language.

Server side scripting language involves script which gives response to the request of user by sending content of pages. Sometimes script at server side generates dynamic response as per clients request and sometimes it gives static pages in response.

Examples of serverside scripting language

1. PHP
2. ASP
3. Python
4. Rubby
5. R

3. Client side scripting language

Client side scripting language accepts content from server and displays it in a clients software like browser. Client side scripting language is generally used to display various kind of information in an attractive way.

Example

1. HTML
2. CSS
3. Javascript

4. Introduction to PHP

PHP (recursive acronym for *PHP: Hypertext Preprocessor*) is a widely-used open source general-purpose scripting language. It is especially suited for web development and can be embedded into HTML

PHP code is executed on the server, generating HTML which is then sent to the client. The client would receive the results of running that script, but would not know the PHP code.

PHP is a server side scripting language; server side scripting language is one, which has the capability of executing the script on the server and serving the output as a HTML File, server side script has the main advantage of interacting with the databases and to perform all types of server manipulations directly. Server side scripting language is responsible for manipulating the data which is filled in the entire web forms, anywhere in the net

How PHP Works?

PHP sits between your browser and the web server. When you type in the URL of a PHP website in your browser, your browser sends out a request to the web server. The web server then calls the PHP script on that page. The PHP module executes the script, which then sends out the result

in the form of HTML back to your browser, which you see on the screen. Here is a basic php diagram which illustrates the process.

PHP Process Diagram

Features of PHP

- PHP is cross-platform. It can be run on Windows, Linux, BSD, Mac OS X, and Solaris, as well as a variety of other platforms.
- PHP is free. You can download the source code, use it, and even make changes to it without ever having to pay any licensing costs. You can even give away your own modified version of PHP. Note to critics: just because PHP is free, it doesn't mean you need to give your scripts away for free.
- PHP is fast. In the majority of scripts beyond basic benchmarks, PHP will easily compete with both Perl and Python, and usually match Microsoft's ASP.NET. Add to that the fact that PHP code can be cached for execution, and PHP's performance is first-class.
- PHP is capable. There are thousands of pre-written functions to perform a wide variety of helpful tasks - handling databases of all sorts (MySQL, Oracle, MS SQL, PostgreSQL, and many others), file uploads, FTP, email, graphical interfaces, generating Flash movies, and more.
- PHP is extendable. Writing your own extension to PHP is a common and easy way to implement speed-critical functionality, and PHP's extension API is a particularly rich and flexible system.
- PHP is reliable. It already runs on millions of servers around the world, which means it is mature enough for even the most demanding of situations.
- PHP is easy to debug. There are a number of debuggers, both commercial and freeware, that make debugging PHP a snap.

Example

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example</title>
  </head>
  <body>

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

  </body>
</html>
```

Advantages of PHP

- PHP can be embedded with HTML
- PHP gives freedom to write code in OOP OR with procedure oriented programming.
- PHP has ability of outputting PDF,Images and Flash.
- PHP also has support for talking to other services using protocols such as LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM
- PHP also supports JSON and SOAP
- PHP supports wide range of databases including MYSQL and ORACLE
-

Client side scripting	Server side scripting
1. It executes on client.	1. It executes on server
2. Examples : DHTML, JavaScript, VBScript, Flash Animation, css, Java Applets	2. Online forms with some dropdown lists That assembles on server.
3. It is glamorous, eye catching part of web Development	3. It is invisible to user.
4. It reduces the round trip so quickly response to user input.	4. It has round trip.

5. They depend entirely on Browsers. Wide variations exist in capability of each browsers, some people disable JavaScript for security reasons.	5. They do not depends entirely on browsers.

5. Installing PHP with wamp

To work with php we need PHP,Apache server and Mysql(to use database),WAMP(windows apache mysql php) software provides bundle of These three things .PHP can be installed and configured with apache and mysql separately too but to work in local computer for developing purpose, wamp is easy way to install PHP

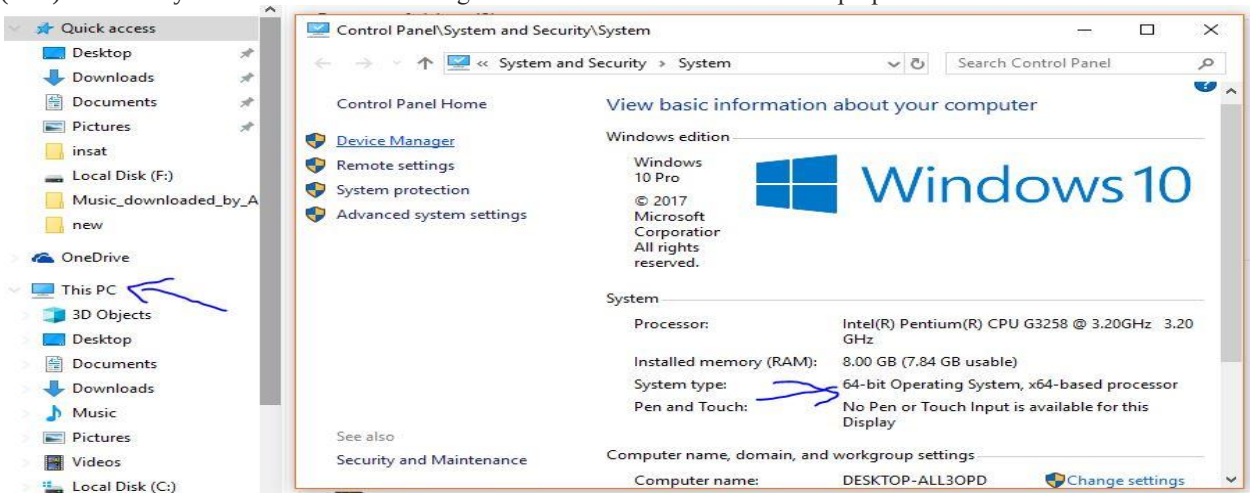
WAMP provides preconfigured package of all these.

Steps to install wamp (apache mysql php) on windows

System requirement
For php 5.6 and above, windows 8 32 bit or 64 bit is needed and VC runtime 2011 from Microsoft or updated is needed.

Step 1: Download the Wamp server

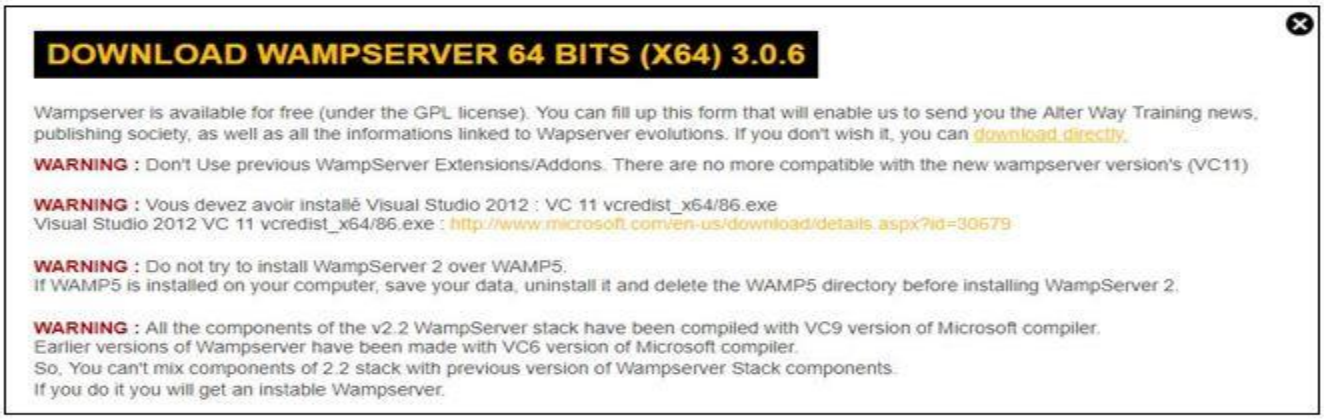
Go to the official WAMP server website (http://www.wampserver.com/en/) and download the WampServer setup according to your Windows 10 version. If you have 64bit then download the Wamp server (x64) otherwise the 32-bit (X86). To check your Windows 10 version right click on "This PC" and select the properties.



Step 2: Warnings and download link

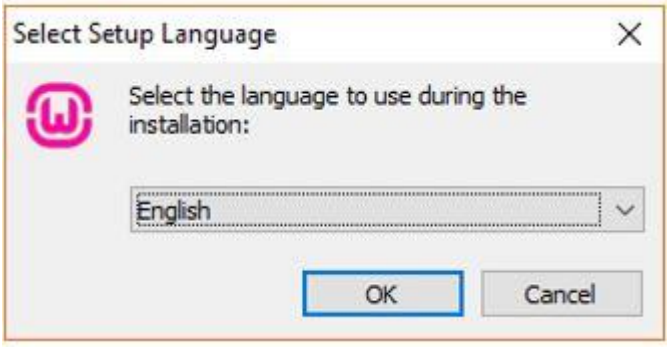
When you click on the Wamp server to download button, a pop-a window with a download link will open that takes you to Sourceforge and also it shows some warnings that need to see too, if you using some previous version of Wamp server.

Here are those:
WARNING: Don't use previous WampServer Extensions/Add-ons. There are no more compatible with the new Wampserver version's (VC11)
WARNING: Do not try to install WampServer 2 over WAMP5. If WAMP5 is installed on your computer, save your data, uninstall it and delete the WAMP5 directory before installing WampServer 2.
WARNING: All the components of the v2.2 WampServer stack have been compiled with VC9 version of Microsoft compiler. Earlier versions of Wampserver have been made with a VC6 version of Microsoft compiler. So, You can't mix components of 2.2 stack with the previous version of Wampserver Stack components. If you do it you will get an unstable Wampserver.
About: Visual studio, in most of the cases, you don't need to install because it already as system update and if not then while installation you get pop-up to download it.

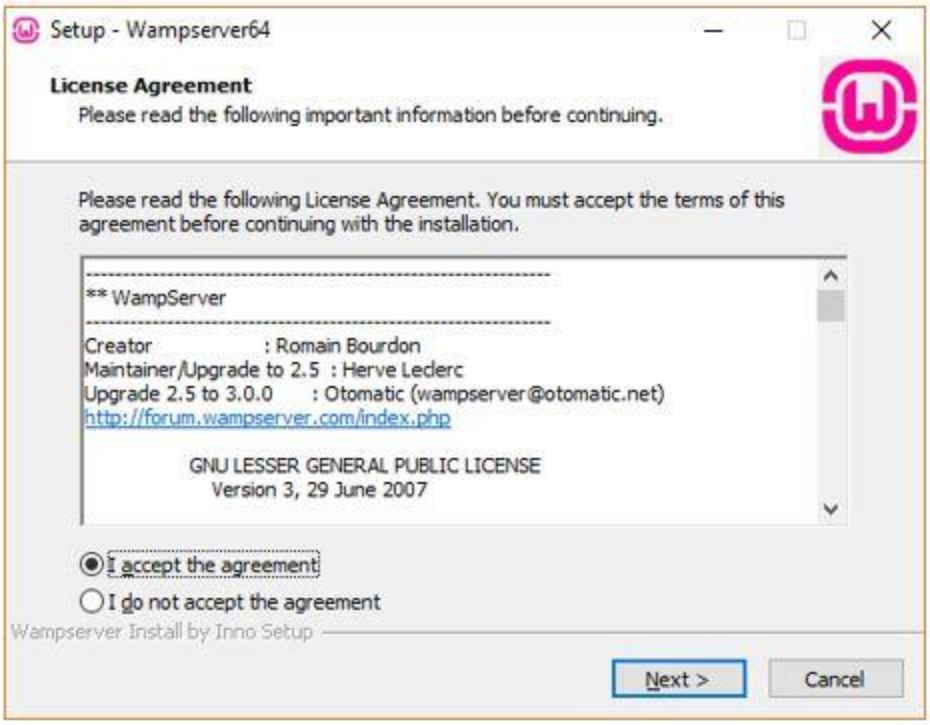


Step 3: WAMP server Installation

Run the setup and select the language in which you want to install the Wamp server for Windows 10.

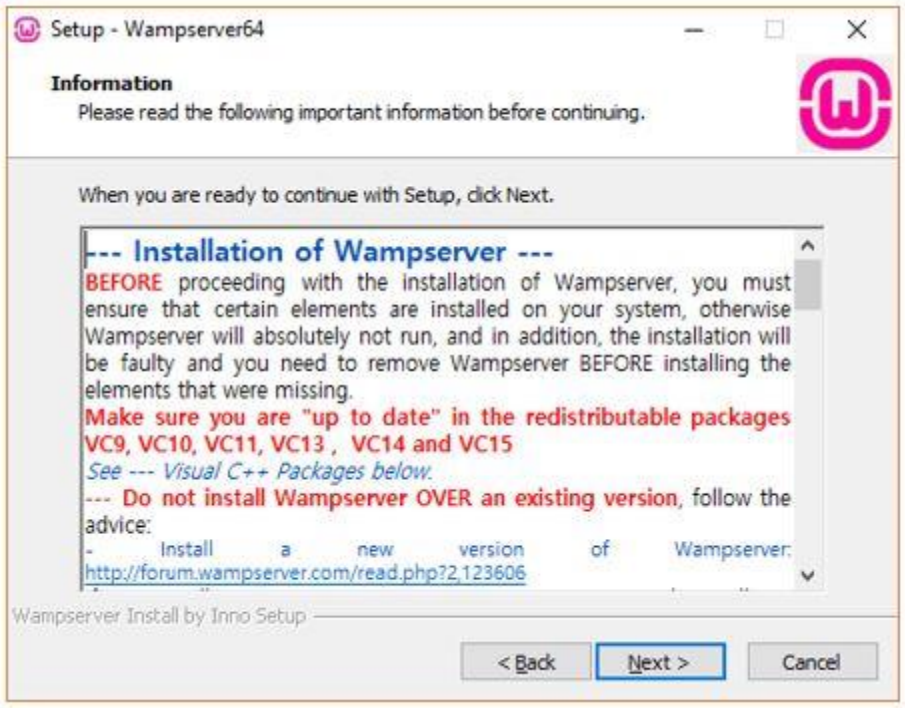


Select the "I accept the agreement"



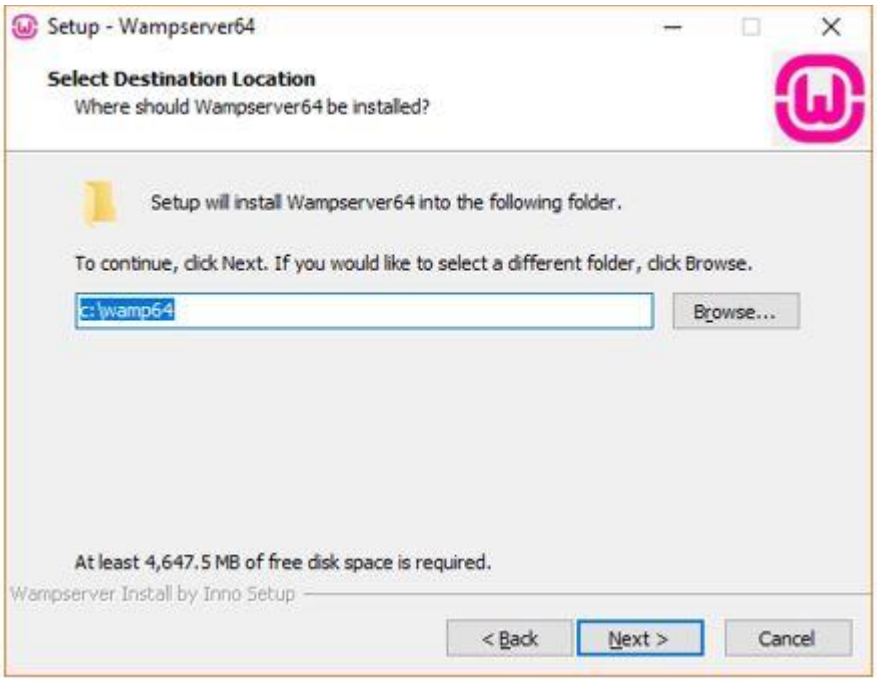
As I said it needs up to date redistributable packages of VC9, VC10, VC13, VC14, and Vc15. Either go to your Control Panel-> Program and features and check whether you have the Visual C++ packages installed or not, or just Click on the Next button and program will automatically identify it. In case, the program not able to find the packages, it will warn you and then you can install them from below given links:

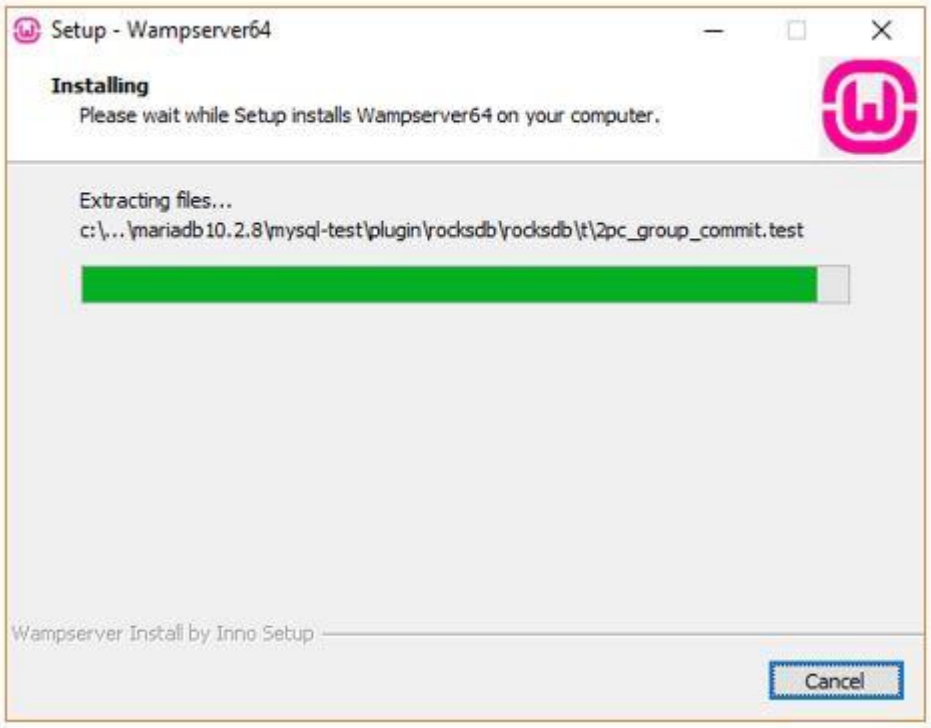
1. Visual C++ Redistributable for Visual Studio 2015
2. Visual C++ Redistributable Packages for Visual Studio 2013
3. Visual C++ Redistributable for Visual Studio 2012 Update 4
4. Microsoft Visual C++ 2010 Redistributable Package (x64)
5. Microsoft Visual C++ 2010 Redistributable Package (x86)
6. Microsoft Visual C++ 2010 SP1 Redistributable Package (x86)
7. Microsoft Visual C++ 2010 SP1 Redistributable Package (x64)



Step 4: Choose Installation directory or location Wamp server

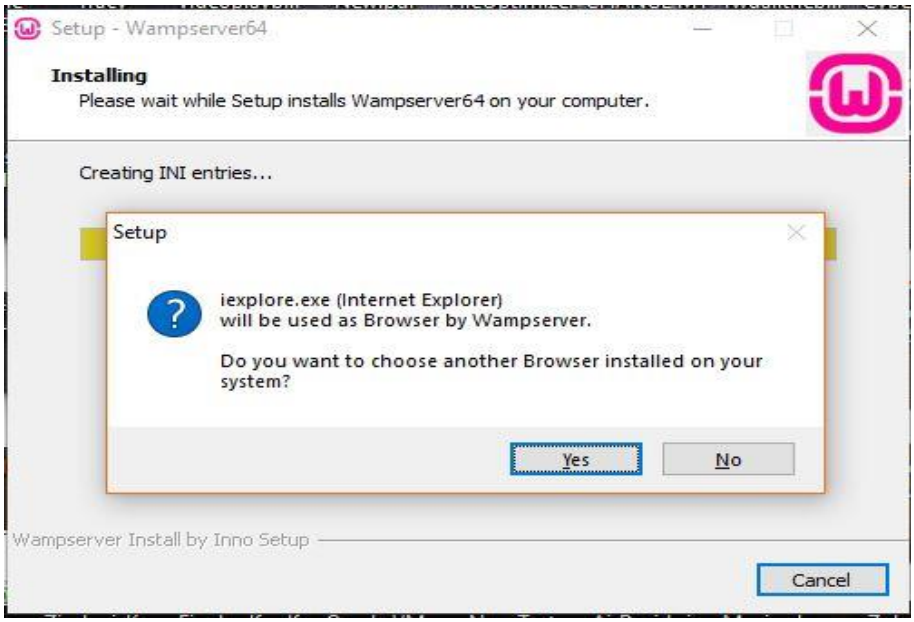
By default all programs you installed in Windows system go to your system's C: drive but if you want to install the Wamp server in some other drive or folder then click on the browse button and choose the location. And then click on NEXT.



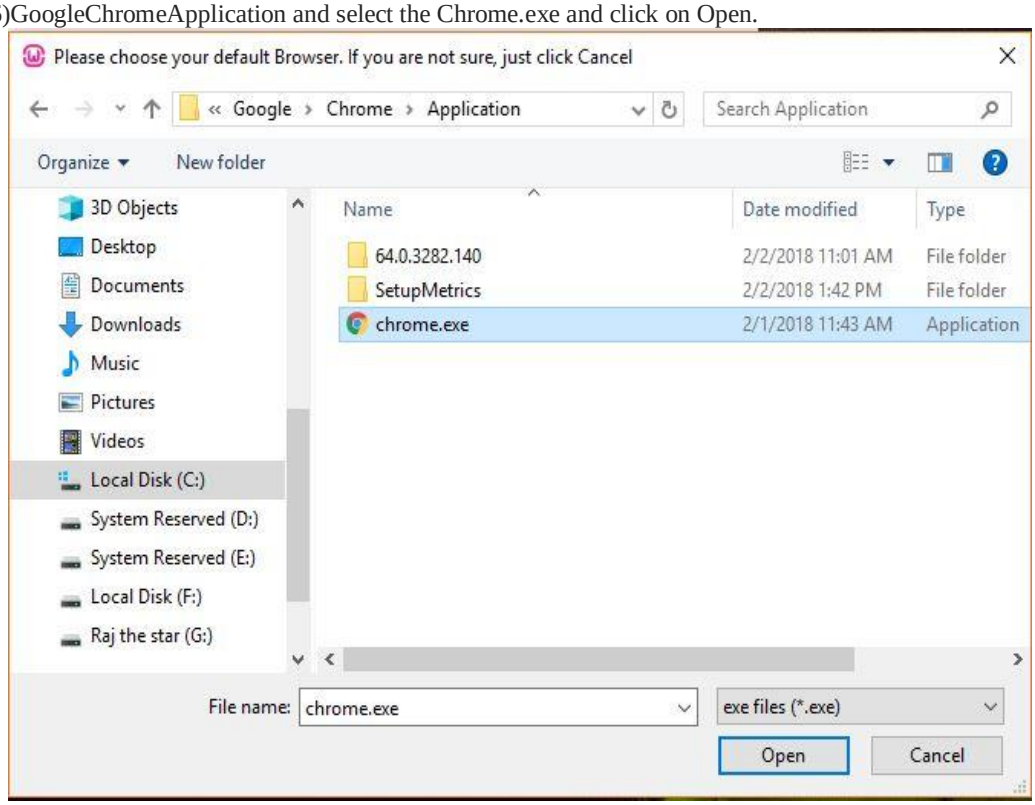


Step 5: Change the Internet Explorer Browser and Text editor of WAMP server on Windows 10

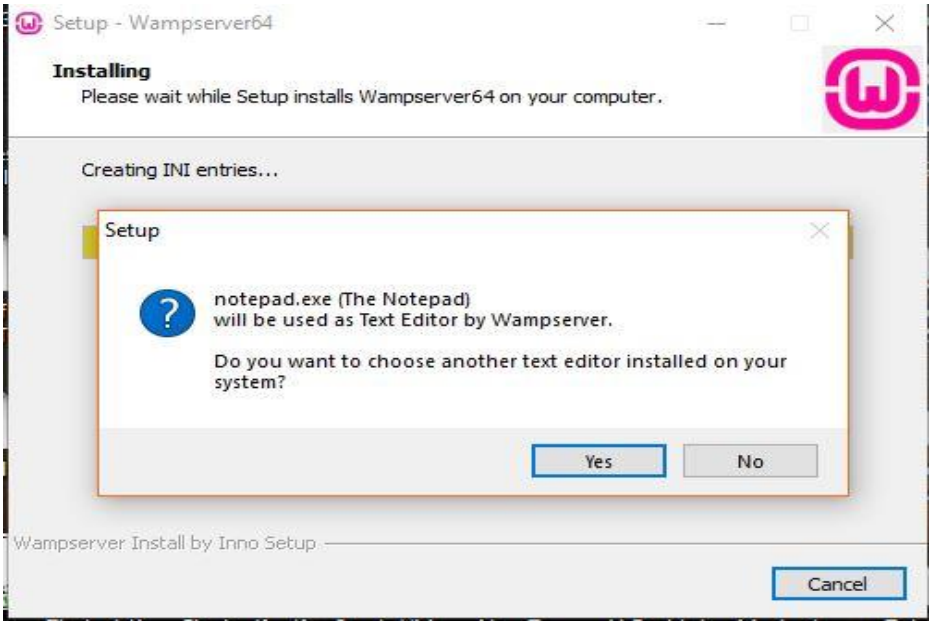
While installing the setup of WampServer, it will ask which browser you want as default in the WAMP server. By default, it uses the iexplore.exe (Internet Explorer) to change it to Google Chrome or Mozilla Firefox click on YES otherwise NO.



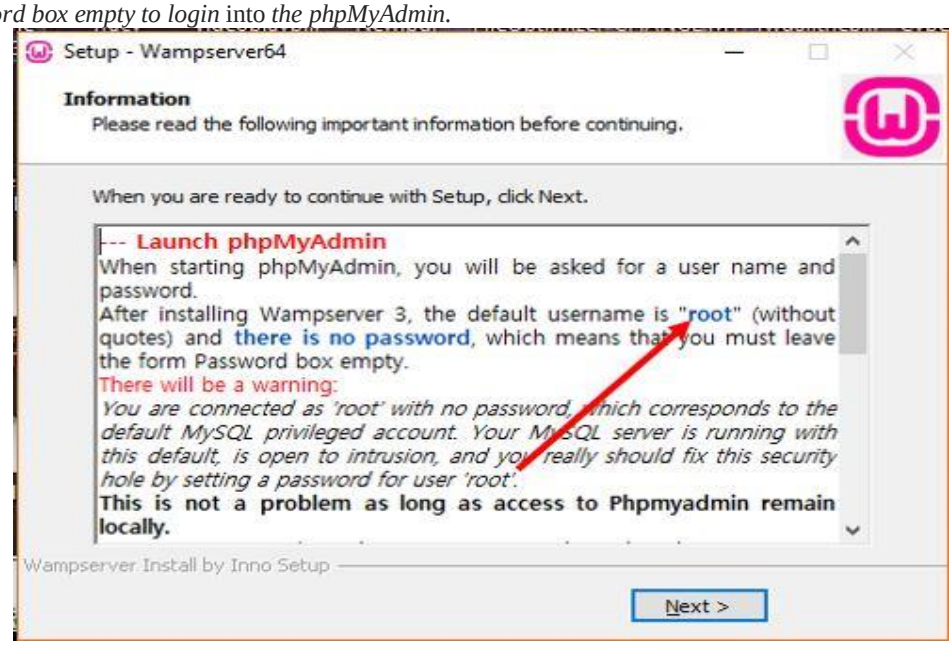
When you click on YES, the Windows Explorer will open from where go to C:Program Files (x86)GoogleChromeApplication and select the Chrome.exe and click on Open.



After selecting the default browser, the Wamp Server will also ask to select the default Text editor which we need in case we want to edit Apache or PHP config files such as http.config or php.ini. By default, it uses the notepad of Windows, I think it doesn't need to be changed, so that's why just click on NO. However, if you want to change the Text editor then click on YES and locate your third-party installed text editor as we do above for selecting the Google chrome.



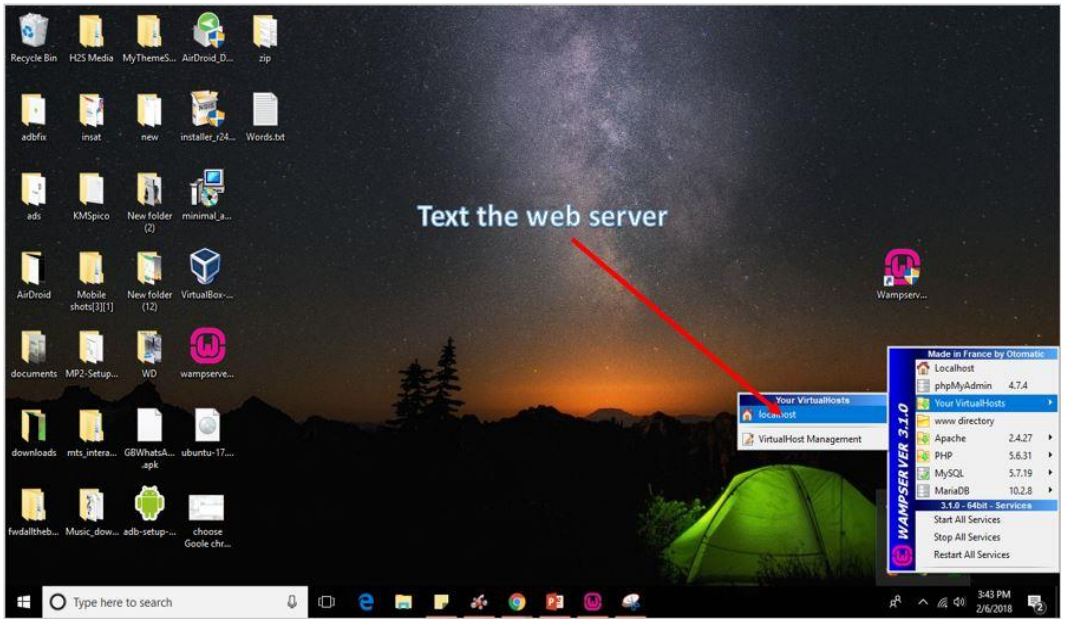
While completing the installation the setup also provides you login information of your phpMyAdmin; "A*fter Installing Wampserver the default username is "root*" without *quotes and there is no password, which means you must leave Password box empty to login* into *the phpMyAdmin.*



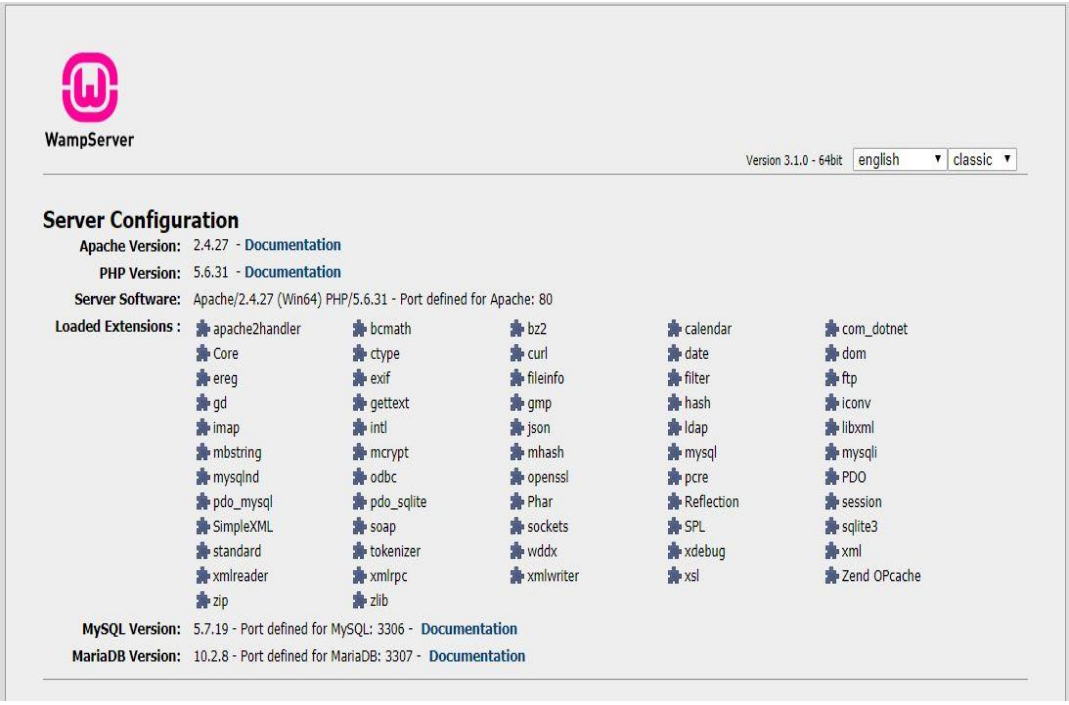
Step 6: Check the Wamp Server Apache

After successful installation, click on the system tray and then on WAMP Server icon. As you click you will find shortcuts to check and use the Apache, MySQL, PHPMyAdmin, MariaDB, Stop all services, Start all services and Reset all services.

To ensure whether our Apache web server is up and running on our Windows 10 click on Localhost.

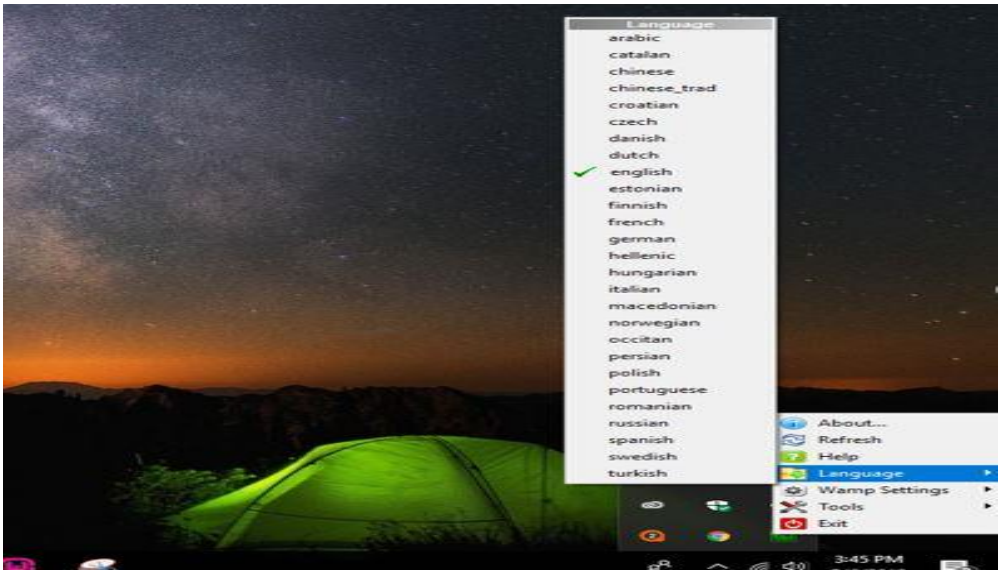


When you click on Localhost a tab will open with a page showing all server configuration in your browser. For reference see the screenshot.



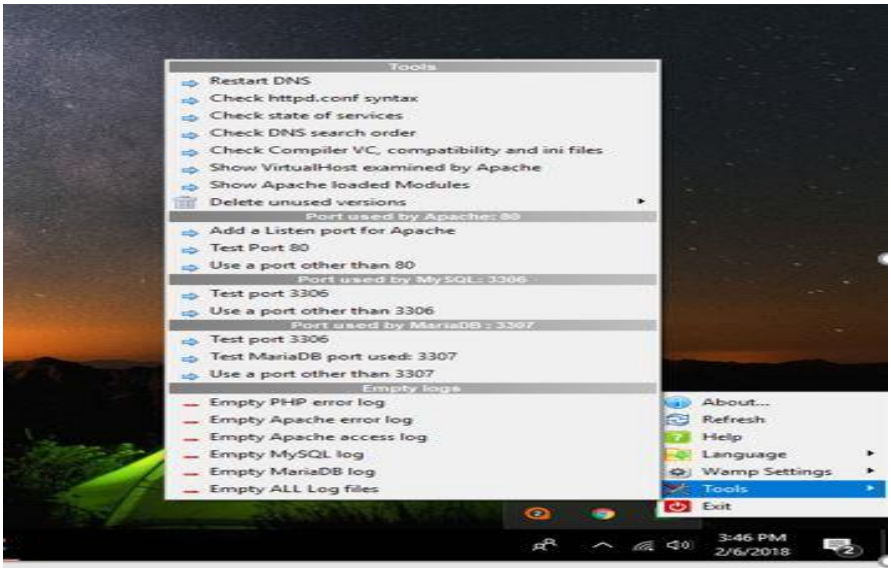
Step 7: Other important Wamp server tools and settings

If you right click on the same WAMP server icon resides in your system tray, it will open several other options such as About, Refresh, Help, language, WAMP settings, Tool, and Exit. In case you want to change the software language you can choose your local language from the language option.



If you want to know which port is using which service of WAMP server or some other information, you just need to go tools:

- Restart DNS
- Check HTTP.conf syntax
- Check stats of services
- Check DNS search order
- Check compiler VC, compatibility and ini files
- Show VirtualHost examined by Apache
- Show Apache loaded Modules
- Delete Unused PHP versions
- And more…



6. Syntext of writing PHP

Php generally written by opening ‘ <?php ‘ tag and ends with ‘ ?> ‘ tag.
For example <?php //code ?>
Shorthand tag also can be used like <? // code ?> if it is enabled from php.ini file
An ASP style can also be used like <% //code %> if it is enabled from php.ini file
Each instruction of php ends with ‘ ; ’ but last statement of code does not need ‘ ; ’ compulsory Like

```
<?php
Statement 1;
Statement 2;
Statement 3;
?> or it can omit ‘ ; ‘ from last statement like
<?php
Statement 1;
Statement 2;
Statement 3
?>
```

7. Embedding PHP with HTML

PHP can be embedded (mixed up) with HTML. It's very easy to write php code in HTML anywhere by starting and ending php tag <?php … ?> in between HTML. HTML tags can also be added with echo or print command in double quote. Like <?php echo " $a ".

```
Example.
<html>
        <head>
                <title>embedding php with html</title>
</head>
<body>
<p><b> Hello how are you
<?php echo "<u> $username </u>";  ?>
 </b></p>
</body>
</html>
```

8. Creating web pages

Fully dynamic pages which are generated by on request of user are examples of web pages created by PHP. There are so many websites which creates dynamic pages as requested by user using PHP.

For example some php script can generate students result page using seat number and semester. Result page is fully dynamic as per seat number created by Server side scripts like PHP.

9. Variables in PHP

Variables in PHP are defined by placing '$' sign before variable name. This PHP parser and interpreter considers all other character as a string but when it finds '$' sign before any character, it considers that as a variable.

PHP is a Loosely Typed Language

- In PHP, a variable does not need to be declared before adding a value to it.
- In the example above, you see that you do not have to tell PHP which data type the variable is.
- PHP automatically converts the variable to the correct data type, depending on its value.

- In a strongly typed programming language, you have to declare (define) the type and name of the variable before using it.
- In PHP, the variable is declared automatically when you use it.

Syntex of defining variables in PHP

1.$varname=value;
2. settypy($varname,datatype) – to set variable or to convert variable's datatype

Example

```
<?php
$a=10;          //setting integer variable $a

$b="hello";   // setting string variable $b

$c=10.20;      // setting float variable $c

$d=TRUE ;   // setting Boolean variable $d

$e= NULL;   // setting null variable $e;

$f=new HelloWorldClass();                  //setting object variable $f

$g=array(45,65,77);          // setting array variable $g

$cn=mysqli_connect($host,$user,$password,$dbaname,$port);     //setting resource variable $cn

?>
```

10. Data Types in PHP-

PHP supports ten primitive types- data type.

a. **Four scalar types**:
   - boolean – accepts TRUE or False;
   - integer – accepts whole number without decimal
   - float (floating-point number, aka double) - accepts number with decimal
   - string - accepts string or character

b. **Four compound types**:

- **Array** – accepts more than one value in a single variable, it can accepts two dimensional or malty dimensional value too.
- **object** – accepts object of any class type.

- **callable** - Some functions like call_user_func() or usort() accept user-defined callback functions as a parameter. Callback functions can not only be simple functions, but also object methods, including static class methods
  it can call any user function by its name as a string
- **iterable** - terable can be used as a parameter type to indicate that a function requires a set of values, but does not care about the form of the value set since it will be used with foreach.

c. And finally two special types:

- **resource** -As resource variables hold special handles to opened files, database connections, image canvas areas and the like, converting to a resourcemakes no sense.
- **NULL** - The special NULL value represents a variable with 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()

examples of data types

1. Integers

Integers are whole numbers, like 1, 12, and 256. The range of acceptable values varies according to the details of your platform but typically extends from -2,147,483,648 to +2,147,483,647. Specifically, the range is equivalent to the range of the long data type of your C compiler. Unfortunately, the C standard doesn't specify what range that long type should have, so on some systems you might see a different integer range.

Integer literals can be written in decimal, octal, or hexadecimal. Decimal values are represented by a sequence of digits, without leading zeros. The sequence may begin with a plus (+) or minus (-) sign. If there is no sign, positive is assumed. Examples of decimal integers include the following:

1998

-641

+33

Octal numbers consist of a leading 0 and a sequence of digits from 0 to 7. Like decimal numbers, octal numbers can be prefixed with a plus or minus. Here are some example octal values and their equivalent decimal values:

```
0755      // decimal 493
+010      // decimal 8
```

Hexadecimal values begin with 0x, followed by a sequence of digits (0-9) or letters (A-F). The letters can be upper- or lowercase but are usually written in capitals. Like decimal and octal values, you can include a sign in hexadecimal numbers:

```
0xFF        // decimal 255
0x10        // decimal 16
-0xDAD1     // decimal -56017
```

If you try to store a too-large integer in a variable, it will automatically be turned into a floating-point number.

Use the is_int( ) function (or its is_integer( ) alias) to test whether a value is an integer:

```
if (is_int($x)) {
   // $x is an integer
}
```

2. Floating-Point Numbers

Floating-point numbers (often referred to as real numbers) represent numeric values with decimal digits. Like integers, their limits depend on your machine's details. PHP floating-point numbers are equivalent to the range of the double data type of your C compiler. Usually, this allows numbers between 1.7E-308 and 1.7E+308 with 15 digits of accuracy.

PHP recognizes floating-point numbers written in two different formats. There's the one we all use every day:

```
3.14
0.017
```

-7.1

but PHP also recognizes numbers in scientific notation:
0.314E1 // $0.314*10^1$, or 3.14
17.0E-3 // $17.0*10^{-3}$, or 0.017

The read format of double is –x.y where – is optionally for negative number and x and y are sequence of number between 0 to 9.

3. Booleans

A boolean value represents a "truth value"—it says whether something is true or not. Like most programming languages, PHP defines some values as true and others as false. Truth and falseness determine the outcome of conditional code such as:
if ($alive) { ... }

In PHP, the following values are false:

- The keyword false
- The integer 0
- The floating-point value 0.0
- The empty string ("") and the string "0"
- An array with zero elements
- An object with no values or functions
- The NULL value

Any value that is not false is true, including all resource

PHP provides true and false keywords for clarity:

```
$x = 5;        // $x has a true value
$x = true;     // clearer way to write it
$y = "";       // $y has a false value
$y = false;    // clearer way to write it
```

Use the is_bool( ) function to test whether a value is a boolean:

```
if (is_bool($x)) {
    // $x is a boolean
}
```

4. NULL

There's only one value of the NULL data type. That value is available through the case-insensitive keyword NULL. The NULL value represents a variable that has no value.

```
$aleph = "beta";
$aleph = null;      // variable's value is gone
$aleph = Null;      // same
$aleph = NULL;      // same
```

Use the is_null( ) function to test whether a value is NULL—for instance, to see whether a variable has a value:

```
if (is_null($x)) {
    // $x is NULL
}
```

5. Strings

Because strings are so common in web applications, PHP includes core-level support for creating and manipulating strings. A string is a sequence of characters of arbitrary length. String literals are delimited by either single or double quotes:

```
'big dog'
"fat hog"
```

Variables are expanded within double quotes, while within single quotes they are not:

```
$name = "Guido";
echo "Hi, $name\n";
echo 'Hi, $name';
Hi, Guido
```

Hi, $name

Double quotes also support a variety of string escapes, as listed in Table.

Escape sequence	Character represented
\"	Double quotes
\n	Newline
\r	Carriage return
\t	Tab
\\	Backslash
\$	Dollar sign
\{	Left brace
\}	Right brace
\[	Left bracket
\]	Right bracket
\0 through \777	ASCII character represented by octal value
\x0 through \xFF	ASCII character represented by hex value

A single-quoted string only recognizes \\ to get a literal backslash and \' to get a literal single quote:

```
$dos_path = 'C:\\WINDOWS\\SYSTEM';
$publisher = 'Tim O\'Reilly';
echo "$dos_path $publisher\n";
C:\WINDOWS\SYSTEM Tim O'Reilly
```

To test whether two strings are equal, use the == comparison operator:

```
if ($a == $b) { echo "a and b are equal" }
```

Use the is_string( ) function to test whether a value is a string:

```
if (is_string($x)) {
    // $x is a string
}
```

Variable Interpolation :

Whenever an unescaped $ symbol appears in doubly quoted string, PHP tries to interpret what follows as a variable name and splices the current values of that variable in to string , this is calles Variable interpolation….

6. Arrays

An array holds a group of values, which you can identify by position (a number, with zero being the first position) or some identifying name (a string):

```
$person[0] = "Edison";
$person[1] = "Wankel";
$person[2] = "Crapper";
```

```
$creator['Light bulb'] = "Edison";
$creator['Rotary Engine'] = "Wankel";
$creator['Toilet'] = "Crapper";
```

The array( ) construct creates an array:

```
$person = array('Edison', 'Wankel', 'Crapper');
$creator = array('Light bulb'    => 'Edison',
                 'Rotary Engine' => 'Wankel',
                 'Toilet'        => 'Crapper');
```

There are several ways to loop across arrays, but the most common is a foreach loop:

```
<?php
foreach ($person as $name) {
  echo "Hello, $name\n";
}
foreach ($creator as $invention => $inventor) {
  echo "$inventor created the $invention\n";
}
?>
```

```
Hello, Edison
Hello, Wankel
Hello, Crapper
Edison created the Light bulb
Wankel created the Rotary Engine
Crapper created the Toilet
```

You can sort the elements of an array with the various sort functions:

```
sort($person);
// $person is now array('Crapper', 'Edison', 'Wankel')

asort($creator);
// $creator is now array('Toilet'        => 'Crapper',
//                       'Light bulb'    => 'Edison',
//                       'Rotary Engine' => 'Wankel');
```

Use the is_array( ) function to test whether a value is an array:

```
if (is_array($x)) {
    // $x is an array
}
```

There are functions for returning the number of items in the array, fetching every value in the array, and much more. Arrays are described in Chapter 5.

7. Objects

PHP supports object-oriented programming (OOP). OOP promotes clean modular design, simplifies debugging and maintenance, and assists with code reuse.

Classes are the unit of object-oriented design. A class is a definition of a structure that contains properties (variables) and methods (functions). Classes are defined with the class keyword:

```
class Person {
  var $name = '';

  function name ($newname = NULL) {
    if (! is_null($newname)) {
      $this->name = $newname;
    }
    return $this->name;
  }
}
```

Once a class is defined, any number of objects can be made from it with the new keyword, and the properties and methods can be accessed with the -> construct:

```
$ed = new Person;
$ed->name('Edison');
printf("Hello, %s\n", $ed->name);
$tc = new Person;
$tc->name('Crapper');
printf("Look out below %s\n", $tc->name);
Hello, Edison
Look out below Crapper
```

Use the is_object( ) function to test whether a value is an object:

```
if (is_object($x)) {
    // $x is an object
```

```
}
```

8. Resources

Many modules provide several functions for dealing with the outside world. For example, every database extension has at least a function to connect to the database, a function to send a query to the database, and a function to close the connection to the database. Because you can have multiple database connections open at once, the connect function gives you something by which to identify that connection when you call the query and close functions: a resource.

Resources are really integers under the surface. Their main benefit is that they're garbage collected when no longer in use. When the last reference to a resource value goes away, the extension that created the resource is called to free any memory, close any connection, etc. for that resource:

```
$res = database_connect( );   // fictitious function
database_query($res);
$res = "boo";                 // database connection automatically closed
```

The benefit of this automatic cleanup is best seen within functions, when the resource is assigned to a local variable. When the function ends, the variable's value is reclaimed by PHP:

```
function search ( ) {
  $res = database_connect( );
  $database_query($res);
}
```

11.static , global and constatnt variables in PHP

static variables

static variables in php are declared using ‘static’ keyword before variable name.

for example

```
function foo() {
    static $index = 0;
    $index++;
    echo "$index\n";
}
```

constant variable

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

Syntax

define(name, value, case-insensitive)

Parameters:

name: Specifies the name of the constant

value: Specifies the value of the constant

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

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

Example

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

All constant variables are automatically global variables.

Global variables

The scope of a variable is the context within which it is defined. For the most part all PHP variables only have a single scope. This single scope spans included and required files as well. For example:

```
<?php
$a = 1;
include 'b.inc';
?>
```

Here the $a variable will be available within the included b.inc script. However, within user-defined functions a local function scope is introduced. Any variable used inside a function is by default limited to the local function scope. For example:

```
<?php
$a = 1; /* global scope */

function test()
{
    echo $a; /* reference to local scope variable */
}

test();
?>
```

This script will not produce any output because the echo statement refers to a local version of the $a variable, and it has not been assigned a value within this scope. You may notice that this is a little bit different from the C language in that global variables in C are automatically available to functions unless specifically overridden by a local definition. This can cause some problems in that people may inadvertently change a global variable. In PHP global variables must be declared global inside a function if they are going to be used in that function.

The global keyword

First, an example use of global:

Example #1 Using global

```
<?php
$a = 1;
$b = 2;

function Sum()
```

```
{
   global $a, $b;

   $b = $a + $b;
}

Sum();
echo $b;
?>
```

The above script will output 3. By declaring $a and $b global within the function, all references to either variable will refer to the global version. There is no limit to the number of global variables that can be manipulated by a function.

A second way to access variables from the global scope is to use the special PHP-defined $GLOBALS array. The previous example can be rewritten as:

Example #2 Using $GLOBALS instead of global

```
<?php
$a = 1;
$b = 2;

function Sum()
{
   $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}

Sum();
echo $b;
?>
```

The $GLOBALS array is an associative array with the name of the global variable being the key and the contents of that variable being the value of the array element. Notice how $GLOBALS exists in any scope, this is because $GLOBALS is a superglobal. Here's an example demonstrating the power of superglobals:

Example #3 Example demonstrating superglobals and scope

```
<?php
function test_superglobal()
{
```

```
    echo $_POST['name'];
}
?>
```

PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

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

PHP Arithmetic Operators

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

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y

*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

PHP Assignment Operators

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

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

Assignment	**Same as...**	**Description**
x = y	x = y	The left operand gets set to the value of the expression on the right

x += y	x = x + y	Addition
x -= y	x = x - y	Subtraction
x *= y	x = x * y	Multiplication
x /= y	x = x / y	Division
x %= y	x = x % y	Modulus

PHP Comparison Operators

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

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y

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

<=>	Spaceship	$x <=> $y	Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7.

PHP Increment / Decrement Operators

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

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

Operator	Name	Description
++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

PHP Logical Operators

The PHP logical operators are used to combine conditional statements.

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

PHP String Operators

PHP has two operators that are specially designed for strings.

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

PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs

===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Returns true if $x is not equal to $y
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y

PHP Conditional Assignment Operators

The PHP conditional assignment operators are used to set a value depending on conditions:

Operator	Name	Example	Result
?:	Ternary	$x = *expr1* ? *expr2* : *expr3*	Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* =

			FALSE
??	Null coalescing	$x = *expr1* ?? *expr2*	Returns the value of $x. The value of $x is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*. Introduced in PHP 7

12. Control structures and looping

Conditional Statements

Conditional Statements allow you to branch the path of execution in a script based on whether a single, or multiple conditions, evaluate to true or false. Put simply, they let you test things and perform various actions based on the results.

If Statements

Take a look at the following:

```
<?php
$x=1;

if ($x == 1) print '$x is equal to 1';
?>
```

This example illustrates the simplest kind of If Statement. If Statements always begin with "if", followed by a condition surrounded in parentheses. If the condition evaluates to true, then the statement or statements immediately following the condition will be executed. In this case, had the condition been false, nothing would have occurred and you would have seen a blank browser

window when the script was run.

When you have more than one statement to be executed within a control structure, it's necessary to surround them with brackets:

```
<?php
$x=1;
if ($x == 1) {
  print '$x is equal to 1';
  $x++;
  print 'now $x is equal to 2';
}
?>
```

Keep in mind that the positioning of the elements does not affect the execution of the script. All of the example arrangements below are perfectly valid not only for If Statements, but for every form of control loop.

```
if ($x == 1) print '$x is equal to 1';

if ($x == 1)
  print '$x is equal to 1';

if ($x == 1) { print '$x is equal to 1'; }

if ($x == 1) {
  print '$x is equal to 1';
}
```

In the interests of clarity, many programmers opt to use indenting and brackets even on one-line blocks of code; however, it is ultimately a matter of personal coding preference.

You can also include multiple conditions within parentheses. For the nested statements to execute, *all* of the conditions must evaluate to true.

```
<?php
$x=1;

if ($x == 1 OR $x == 2) print '$x is equal to 1 (or maybe 2)';
?>
```

Else Statements

As the name implies, Else Statements allow you to do something *else* if the condition within an If Statement evaluated to false:

```
<?php
$x=1;

if ($x == 2) {
print '$x is equal to 2';
} else {
   print '$x is equal to 1';
}
?>
```

Else If Statements

Thus far, we have been able to respond to one condition, and do something if that condition is not true. But what about evaluating multiple conditions? You could use a series of If Statements to test each potential condition, but in some situations that is not a suitable option. Here is where Else If Statements come in.

A combination of If and Else Statements, Else If Statements are evaluated sequentially if the condition within the If Statement is false. When a condition within an Else If Statement evaluates to true, the nested statements are parsed, the script stops executing the entire If/Else if/Else Structure. The rest of the script proceeds to be parsed.

Take a look at the following example:

```
<?php
$x=1;

if ($x == 2) {
print '$x is equal to 2';
} else if ($x == 1) {
print '$x is equal to 1';
} else {
   print '$x does not equal 2 or 1';
}
?>
```

The final else statement can be left off if you do not want anything to happen if none of the If or Else If Statements are true:

```
<?php
$x=0;

if ($x == 2) {
print '$x is equal to 2';
} else if ($x == 1) {
print '$x is equal to 1';
}
?>
```

In this case, since neither the condition within the If or Else if Conditions are true, and no Else Statement was provided, nothing would be outputted to the browser.

Switches

Switches are a good alternative to If/Else if/Else Statements in situations where you want to check multiple values against a single variable or condition. This is the basic syntax:

```
<?php
$var = "yes";

switch ($var) {
  case "yes":
    print '$var is equal to yes';
break;

  case "no":
    print '$var is equal to no';
    break;
```

```
}
?>
```

After running this code snippet, much of what is here will probably make sense to you. In the first line of the switch statement, we have the identifier "switch" followed by a variable surrounded by parenthesis. Each case includes a possible value for the variable.

Switches execute a little differently than If/Else if/Else statements. Once a case value matches the value of the switch expression, every following statement is executed, *including* those following other cases.

To prevent this from happening, a break statement is used. "Break;" ends the execution of the switch statement, and lets the script continue execution; it can also be used in while or for loops.

Optionally, you may also include a special case called "default". This case works much like and Else Statement, and will execute if all the other cases are found to be false. This case should be the very last one you include.

```
<?php
$var = "yes";

switch ($var) {
  case "maybe":
    print '$var is equal to yes';
break;

  case "no":
    print '$var is equal to no';
```

```
    break;

  default:
    print 'none of the other two cases were true, so this sentance will be
printed out instead.';
}
?>
```

Similar to the Break Statement is the Exit Statement. Exit is particularly useful in situations where you run into what would be considered a "fatal error" (for example, if the user had entered a password that was incorrect) or any other time you needed to end the execution of a script before it naturally terminated.

```
<?php
$var = "yes";

switch ($var) {
  case "yes":
    print '$var is equal to yes';
    exit;
  case "no":
    print '$var is equal to no';
    break;
}
```

```
print "this will not be printed, because the script will have terminate before this line is reached";
?>
```

Unlike break, exit may be used anywhere in your scripts, inside \or outside of control structures.

The Ternary Operator

Though Technically an Operator, not a Control Structure, the Ternary Operator, represented by "?", can be used as shorthand for simple If/Else Statements. It can only be used in situations where you want execute a single expression based on whether a single condition is true or false:

```
<?php
$x = 1;
($x==1) ? (print '$x is equal to 1') : (print '$x is not equal to 1');
?>
```

The condition is contained within the first set of parentheses. If it evaluates to true, then the expression within the second set of parentheses will be performed. Otherwise, the expression in the third set will be performed:

```
(condition) ? (executes if the condition is true) : (executes if the condition is false);
```

You will see how this can be particular useful a little later on.

Control Loops

Frequently in PHP there are instances where you need to perform repetitive tasks, such as formatting data pulled from a database, sending out emails to a mailing list, or cycling through the contents of an array. Control Loops allow you to perform these tasks almost effortlessly.

While Loops

While Loops are the simplest form of loops:

```
<?php
$x=1;

while ($x <=10) {
  print "$x<br>";
  $x++;
}
?>
```

When you run this snippet, you will see the numbers 1 through 10 printed on your screen. Take a look at the code. In many ways, it's very similar to an If Statement. First is the while identifier, followed by a condition surrounded by parentheses.

The statements nested within the loop will execute as long as the condition within the parentheses evaluates to true. Since the validity of the condition is checked before the loop is executed, if the condition is false, then the statements within the loop will not be executed at all.
Do...While Loops
Do...While Loops are close cousins of While Loops:

```
<?php
$x=11;

do {
print "$x<br>";
$x++;
} while ($x <=10);?>
```

The primary difference in how these work is that the validity of the condition in a Do...While Loop is tested *after* the loop has itinerated once. This means that in the example above, $x would

be printed out one time, and then the execution of the loop would end, because $x is greater than 10.

For Loops

```
<?php
for($x=1; $x<=10; $x++) {
print "$x<br>";
}
?>
```

foreach

This simply gives an easy way to iterate over arrays. *foreach* works only on arrays, and will issue an error when you try to use it on a variable with a different data type or an uninitialized variable. There are two syntaxes; the second is a minor but useful extension of the first:

```
foreach (array_expression as $value)
   statement
foreach (array_expression as $key => $value)
   statement
```

The first form loops over the array given by *array_expression*. On each loop, the value of the current element is assigned to *$value* and the internal array pointer is advanced by one (so on the next loop, you'll be looking at the next element).

The second form does the same thing, except that the current element's key will be assigned to the variable *$key* on each loop.

```
<?php
/* foreach example  value only */
$a = array(1, 2, 3, 17);
```

```
foreach ($a as $v) {
   echo "Current value of \$a: $v.\n";
}
?>
<?

/* foreach example  key and value */
$a = array(
    "one" => 1,
    "two" => 2,
    "three" => 3,
    "seventeen" => 17
);

foreach ($a as $k => $v) {
    echo "\$a[$k] => $v.\n";
}
?>
```

break

break ends execution of the current *for, foreach, while, do-while* or *switch* structure.

```
<?php
switch ($i) {
case "apple":
    echo "i is apple";
    break;
case "bar":
```

```
    echo "i is bar";
    break;
}
?>
```

continue

continue is used within looping structures to skip the rest of the current loop iteration and continue execution at the condition evaluation and then the beginning of the next iteration.

```
<?php
  for ($i = 0; $i < 5; ++$i) {
      if ($i == 2)
          continue;
      print "$i\n";  }
?>
OUTPUT IS
0 1 3 4
```

13. PHP User Defined Functions

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

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

A function will not execute immediately when a page loads.

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

Create a User Defined Function in PHP

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

Syntax

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

Example

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

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

PHP Function Arguments

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

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

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

Example

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

familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
```

```
familyName("Borge");
?>
```

PHP Default Argument

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

Example

```
<?php declare(strict_types=1); // strict requirement
function setHeight(int $minheight = 50) {
   echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

PHP Functions - Returning values

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

Example

```
<?php declare(strict_types=1); // strict requirement
function sum(int $x, int $y) {
   $z = $x + $y;
   return $z;
}

echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
```

```
echo "2 + 4 = " . sum(2, 4);
?>
```

PHP Variable Length Argument Function

PHP supports variable length argument function. It means you can pass 0, 1 or n number of arguments in function. To do so, you need to use 3 ellipses (dots) before the argument name.

The 3 dot concept is implemented for variable length argument since PHP 5.6.

```
<?php
function add(...$numbers) {
   $sum = 0;
   foreach ($numbers as $n) {
      $sum += $n;
   }
   return $sum;
}
  echo add(1, 2, 3, 4);
?>
```

Variable functions

PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it. Among other things, this can be used to implement callbacks, function tables, and so forth.

Variable functions won't work with language constructs such
as echo, print, unset(), isset(), empty(), include, require and the like. Utilize wrapper functions to make use of any of these constructs as variable functions.

Example #1 Variable function example

```
<?php
function foo() {
    echo "In foo()<br />\n";
}

function bar($arg = '')
{
    echo "In bar(); argument was '$arg'.<br />\n";
}

// This is a wrapper function around echo
function echoit($string)
{
    echo $string;
}

$func = 'foo';
$func();        // This calls foo()

$func = 'bar';
$func('test');  // This calls bar()

$func = 'echoit';
$func('test');  // This calls echoit()
?>
```

Object methods can also be called with the variable functions syntax.

Example #2 Variable method example

```
<?php
class Foo
{
    function Variable()
    {
        $name = 'Bar';
```

```
        $this->$name(); // This calls the Bar() method
    }

    function Bar()
    {
        echo "This is Bar";
    }
}
$foo = new Foo();
$funcname = "Variable";
$foo->$funcname();  // This calls $foo->Variable()

?>
```

Writing a whole function with return statement

```
<?php

$user="one"; $pass="two:"; $host="three";

function setData($user, $pass, $host){

  $a=$user;

  return function() use ($a){

    return $a;

 } };

$ar=setData($user, $pass, $host);

print_r ($ar);

?>
```

14. PHP Global Variables - Superglobals

Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

The PHP superglobal variables are:

$GLOBALS

$_SERVER

$_REQUEST

$_POST

$_GET

$_FILES

$_ENV

$_COOKIE

$_SESSION

This chapter will explain some of the superglobals, and the rest will be explained in later chapters.

$GLOBALS

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

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

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

Example

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

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

addition();
echo $z;
?>
```

$_SERVER

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

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

Example

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

$_REQUEST

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

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_REQUEST to collect the value of the input field:

Example

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

```
<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>




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

</body>
</html>
```

$_POST

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

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to the file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_POST to collect the value of the input field:

Example

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
```

```
  <input type="submit">
</form>

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

</body>
</html>
```

$_GET

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

$_GET can also collect data sent in the URL.

Assume we have an HTML page that contains a hyperlink with parameters:

```
<html>
<body>

<a href="test_get.php?subject=PHP&web=W3schools.com">Test $GET</a>

</body>
</html>
```

When a user clicks on the link "Test $GET", the parameters "subject" and "web" are sent to "test_get.php", and you can then access their values in "test_get.php" with $_GET.

The example below shows the code in "test_get.php":

Example

```
<html>
<body>

<?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?>

</body>
</html>
```

	GET	**POST**
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Bookmarked	Can be bookmarked	Cannot be bookmarked
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed

Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	Data is visible to everyone in the URL	Data is not displayed in the URL

PHP Session

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

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

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

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

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

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

Start a PHP Session

A session is started with the **session_start() function**.

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

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

Example

```
<?php
// Start the session
```

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

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

</body>
</html>
```

Get PHP Session Variable Values

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

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

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

Example

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

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

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

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

Example

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

<?php
print_r($_SESSION);
?>

</body>
</html>
```

Destroy a PHP Session

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

Example

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

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

```
// destroy the session
session_destroy();
?>

</body>
</html>
```

What is a Cookie?

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

Create Cookies With PHP

A cookie is created with the setcookie() function.

Syntax

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

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

PHP Create/Retrieve a Cookie

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

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

Example

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
```

```
} else {
   echo "Cookie '" . $cookie_name . "' is set!<br>";
   echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Modify a Cookie Value

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

Example

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
   echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
   echo "Cookie '" . $cookie_name . "' is set!<br>";
   echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Delete a Cookie

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

Example

```
<?php
// set the expiration date to one hour ago
```

```
setcookie("user", "", time() - 3600);
?>
<html>
<body>
<?php
echo "Cookie 'user' is deleted.";
?>
</body>
</html>
```

Check if Cookies are Enabled

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

Example

```
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>

<?php
if(count($_COOKIE) > 0) {
   echo "Cookies are enabled.";
} else {
   echo "Cookies are disabled.";
}
?>

</body>
</html>
```

15. Library functions

a. Variable functions

gettype()

You can use PHP4's built-in function gettype() to test the type of any variable. If you place a variable between the parentheses of the function call, gettype() returns a string representing the relevant type

Example

```
<html>
 <head>
<title>Testing the type of a variable</title>
</head>
<body>
<?php
$testing = 5;
print gettype( $testing ); // integer
 print "<br>";
 $testing = "five";
 print gettype( $testing ); // string
 print("<br>");
 $testing = 5.0;
 print gettype( $testing ); // double
 print("<br>");
 $testing = true;
 print gettype( $testing ); // boolean
 print "<br>";
 ?>
```

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

This script produces the following:

integer

string

double

Boolean

settype()

PHP provides the function settype() to change the type of a variable. To use settype(), you must place the variable to change (and the type to change it to) between the parentheses and separated by commas

```
<html>
<head>
 <title>Listing 4.5 Changing the type of a variable with settype()</title>
 </head>
 <body>
<?php
$undecided = 3.14;
print gettype( $undecided ); // double
print " -- $undecided<br>"; // 3.14
settype( $undecided, string );
 print gettype( $undecided ); // string
 print " -- $undecided<br>"; // 3.14
 settype( $undecided, integer );
 print gettype( $undecided ); // integer
```

```
print " -- $undecided<br>"; // 3
settype( $undecided, double );
print gettype( $undecided ); // double
print " -- $undecided<br>"; // 3.0
settype( $undecided, boolean );
print gettype( $undecided ); // boolean
print " -- $undecided<br>"; // 1
?>
```

Isset()

— Determine if a variable is declared and is different than NULL

Description

isset (mixed $var [, mixed $...]) : bool

Determine if a variable is considered set, this means if a variable is declared and is different than NULL.

If a variable has been unset with the unset () function, it is no longer considered to be set.

isset() will return FALSE when checking a variable that has been assigned to NULL. Also note that a null character ("\0") is not equivalent to the PHP NULL constant.

If multiple parameters are supplied then isset() will return TRUE only if all of the parameters are considered set. Evaluation goes from left to right and stops as soon as an unset variable is encountered.

Parameters

var

The variable to be checked.

...

Another variable ...

Return Values

Returns TRUE if var exists and has any value other than NULL. FALSE otherwise.

Example isset()

```
<?php

$var = '';

// This will evaluate to TRUE so the text will be printed.
if (isset($var)) {
    echo "This var is set so I will print.";
}

// In the next examples we'll use var_dump to output
// the return value of isset().

$a = "test";
$b = "anothertest";

var_dump(isset($a));      // TRUE
var_dump(isset($a, $b)); // TRUE

unset ($a);

var_dump(isset($a));     // FALSE
var_dump(isset($a, $b)); // FALSE

$foo = NULL;
var_dump(isset($foo));   // FALSE

?>
```

Strval()

(PHP 4, PHP 5, PHP 7)

strval — Get string value of a variable

Description

strval (mixed $var) : string

Get the string value of a variable. See the documentation on string for more information on converting to string.

This function performs no formatting on the returned value. If you are looking for a way to format a numeric value as a string, please see sprintf () or number_format().

Parameters

var

The variable that is being converted to a string.

var may be any scalar type or an object that implements the __toString() method. You cannot use strval() on arrays or on objects that do not implement the __toString() method.

Return Values

The string value of var.

Examples

Example #1 strval() example using PHP 5's magic __toString() method.

```
<?php
$var_name = 11.010;
echo strval ($var_name);
?>
```

Floatval()

(PHP 4 >= 4.2.0, PHP 5, PHP 7)

floatval — Get float value of a variable

Description

floatval (mixed $var) : float

Gets the float value of var.

Parameters

var

May be any scalar type. floatval() should not be used on objects, as doing so will emit an E_NOTICE level error and return 1.

Return Values

The float value of the given variable. Empty arrays return 0, non-empty arrays return 1.

Strings will most likely return 0 although this depends on the leftmost characters of the string. The common rules of float casting apply.

Example #1 floatval()

```
<?php
$var = '122.34343The';
$float_value_of_var = floatval($var);
echo $float_value_of_var; // 122.34343
?>
```

Example #2 floatval() non-numeric leftmost characters Example

```
<?php
$var = 'The122.34343';
$float_value_of_var = floatval($var);
echo $float_value_of_var; // 0
?>
```

Intval()

(PHP 4, PHP 5, PHP 7)

intval — Get the integer value of a variable

Description

intval (mixed $var [, int $base = 10]) : int

Returns the integer value of var, using the specified base for the conversion (the default is base 10). intval() should not be used on objects, as doing so will emit an E_NOTICE level error and return 1.

Parameters

var

The scalar value being converted to an integer

base

The base for the conversion

Note:

If base is 0, the base used is determined by the format of var:

if string includes a "0x" (or "0X") prefix, the base is taken as 16 (hex); otherwise,

if string starts with "0", the base is taken as 8 (octal); otherwise,

the base is taken as 10 (decimal).

Return Values

The integer value of var on success, or 0 on failure. Empty arrays return 0, non-empty arrays return 1.

The maximum value depends on the system. 32 bit systems have a maximum signed integer range of -2147483648 to 2147483647. So for example on such a system, intval('1000000000000') will return 2147483647. The maximum signed integer value for 64 bit systems is 9223372036854775807.

Strings will most likely return 0 although this depends on the leftmost characters of the string. The common rules of integer casting apply.

Example #1 intval()

The following examples are based on a 32 bit system.

```
<?php
echo intval(42);                // 42
echo intval(4.2);               // 4
echo intval('42');              // 42
echo intval('+42');             // 42
echo intval('-42');             // -42
echo intval(042);               // 34
```

```
echo intval('042');                  // 42
echo intval(1e10);                   // 1410065408
echo intval('1e10');                 // 1
echo intval(0x1A);                   // 26
echo intval(42000000);               // 42000000
echo intval(420000000000000000000);  // 0
echo intval('420000000000000000000'); // 2147483647
echo intval(42, 8);                  // 42
echo intval('42', 8);                // 34
echo intval(array());                // 0
echo intval(array('foo', 'bar'));    // 1
?>
```

print_r()

(PHP 4, PHP 5, PHP 7)

print_r — Prints human-readable information about a variable

Description

print_r (mixed $expression [, bool $return = FALSE]) : mixed

print_r() displays information about a variable in a way that's readable by humans.

print_r(), var_dump() and var_export() will also show protected and private properties of objects. Static class members will not be shown.

Parameters

expression

The expression to be printed.

return

If you would like to capture the output of print_r(), use the return parameter. When this parameter is set to TRUE, print_r() will return the information rather than print it.

Return Values

If given a string, integer or float, the value itself will be printed. If given an array, values will be presented in a format that shows keys and elements. Similar notation is used for objects.

When the return parameter is TRUE, this function will return a string. Otherwise, the return value is TRUE.

Note:

When the return parameter is used, this function uses internal output buffering so it cannot be used inside an ob_start() callback function.

Examples

```
<pre>
<?php
$a = array ('a' => 'apple', 'b' => 'banana', 'c' => array ('x', 'y', 'z'));
print_r ($a);
?>
</pre>
```

The above example will output:

```
<pre>

Array

(

    [a] => apple

    [b] => banana

    [c] => Array

        (

            [0] => x

            [1] => y

            [2] => z

        )

)

</pre>
```

b. String Functions

Chr()

Return characters from different ASCII values:

Definition and Usage

The chr() function returns a character from the specified ASCII value

Syntax

chr(ascii)

Parameter	**Description**
ascii	Required. An ASCII value

Example

```
<?php
echo chr(52) . "<br>"; // Decimal value
echo chr(052) . "<br>"; // Octal value
echo chr(0x52) . "<br>"; // Hex value
?>
```

Output

4
*
R

The ASCII value can be specified in decimal, octal, or hex values. Octal values are defined by a leading 0, while hex values are defined by a leading 0x.

ord() Function

Definition and Usage

The ord() function returns the ASCII value of the first character of a string.

Syntax

ord(string)

Parameter	Description
string	Required. The string to get an ASCII value from

Example

Return the ASCII value of "h":

```
<?php
echo ord("h")."<br>";
echo ord("hello")."<br>";
?>
```

104
104

strtolower()

Definition and Usage

The strtolower() function converts a string to lowercase.

Syntax

strtolower(string)

Parameter	Description
string	Required. Specifies the string to convert

Return Value: Returns the the lowercased string

Example

Convert all characters to lowercase:

```
<?php
echo strtolower("Hello WORLD.");
?>
```

Output

hello world.

strtoupper() Function

Definition and Usage

The strtoupper() function converts a string to uppercase.

Syntax

strtoupper(string)

Parameter	Description
string	Required. Specifies the string to convert

Example

Convert all characters to uppercase:

```
<?php
echo strtoupper("Hello WORLD!");
?>

HELLO WORLD!
```

substr() Function

Definition and Usage

The substr() function returns a part of a string.

Syntax

substr(string,start,length)

Parameter	Description
string	Required. Specifies the string to return a part of
start	Required. Specifies where to start in the string A positive number - Start at a specified position in the string A negative number - Start at a specified position from the end of the string 0 - Start at the first character in string

length	Optional. Specifies the length of the returned string. Default is to the end of the string. A positive number - The length to be returned from the start parameter Negative number - The length to be returned from the end of the string If the length parameter is 0, NULL, or FALSE - it return an empty string
Return Value:	Returns the extracted part of a string, or FALSE on failure, or an empty string

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
echo substr("Hello world",6);
?>
</body>
</html>
```

Output

World

strcmp() Function

Definition and Usage

The strcmp() function compares two strings.

Note: The strcmp() function is binary-safe and case-sensitive.

Syntax

strcmp(string1,string2)

Parameter	**Description**
string1	Required. Specifies the first string to compare
string2	Required. Specifies the second string to compare

Return Value:	This function returns: 0 - if the two strings are equal <0 - if string1 is less than string2 >0 - if string1 is greater than string2

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
echo strcmp("Hello world!","Hello world!");
?>
<p>If this function returns 0, the two strings are equal.</p>
</body>
</html>
```

```
0
```

If this function returns 0, the two strings are equal.

strcasecmp() Function

Definition and Usage

The strcasecmp() function compares two strings.

Tip: The strcasecmp() function is binary-safe and case-insensitive.

Tip: This function is similar to the strncasecmp() function, with the difference that you can specify the number of characters from each string to be used in the comparison with strncasecmp().

Syntax

strcasecmp(string1,string2)

Parameter	Description
string1	Required. Specifies the first string to compare
string2	Required. Specifies the second string to compare

Return Value:	This function returns: 0 - if the two strings are equal <0 - if string1 is less than string2 >0 - if string1 is greater than string2

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
echo strcasecmp("Hello world!","HELLO WORLD!");
?>
<p>If this function returns 0, the two strings are equal.</p>
</body>
</html>
0
```

If this function returns 0, the two strings are equal.

stristr() Function

Definition and Usage

The stristr() function searches for the first occurrence of a string inside another string.

Syntax

stristr(string,search,before_search)

Parameter Description

string Required. Specifies the string to search

search Required. Specifies the string to search for. If this parameter is a number, it will search for the character matching the ASCII value of the number

```
<!DOCTYPE html>
<html>
<body>
<?php
echo stristr("Hello world!",111);
?>
</body>
</html>
```

Output

o world!

substr() Function

Definition and Usage

The substr() function returns a part of a string.

Syntax

substr(string,start,length)

Parameter	Description
string	Required. Specifies the string to return a part of
start	Required. Specifies where to start in the string

A positive number -	Start at a specified position in the string
A negative number -	Start at a specified position from the end of the string
0 -	Start at the first character in string

length	Optional. Specifies the length of the returned string. Default is to the end of the string.
A positive number -	The length to be returned from the start parameter
Negative number -	The length to be returned from the end of the string

If the length parameter is 0, NULL, or FALSE - it return an empty string

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


<?php
echo substr("Hello world",6);
?>


</body>
</html>
output
world
```

strlen() Function

Definition and Usage

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

Syntax

strlen(string)

Parameter	**Description**
string	Required. Specifies the string to check

Return Value: Returns the length of a string (in bytes) on success, and 0 if the string is empty

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

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

</body>
</html>
```

Output

5

strpos() Function

Definition and Usage

The strpos() function finds the position of the first occurrence of a string inside another string.

Note: The strpos() function is case-sensitive.

Syntax

strpos(string,find,start)

Parameter	**Description**
string	Required. Specifies the string to search
find	Required. Specifies the string to find
start	Optional. Specifies where to begin the search. If start is a negative number, it counts from the end of the string.

Return Value: Returns the position of the first occurrence of a string inside another string, or FALSE if the string is not found. Note: String positions start at 0, and not 1.

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
echo strpos("I love php, I love php too!","php");
?>
</body>
</html>
```

output

7

strrpos() Function

Definition and Usage

The strrpos() function finds the position of the last occurrence of a string inside another string.

Note: The strrpos() function is case-sensitive.

Syntax

strrpos(string,find,start)

Parameter	**Description**
string	Required. Specifies the string to search
find	Required. Specifies the string to find
start	Optional. Specifies where to begin the search

Return Value: Returns the position of the last occurrence of a string inside another string, or FALSE if the string is not found. Note: String positions start at 0, and not 1.

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

<?php
echo strrpos("I love php, I love php too!","php");
?>

</body>
</html>
19
```

stripos() Function

Definition and Usage

The stripos() function finds the position of the first occurrence of a string inside another string.

The stripos() function is case-insensitive.

Syntax

stripos(string,find,start)

Parameter	**Description**
string	Required. Specifies the string to find
start	Optional. Specifies where to begin the search

Return Value: Returns the position of the first occurrence of a string inside another string, or FALSE if the string is not found. Note: String positions start at 0, and not 1.

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

<?php
echo stripos("I love php, I love php too!","PHP");
?>
</body>
```

```
</html>
```

Output

19

strrev() Function

Definition and Usage

The strrev() function reverses a string.

Syntax

strrev(string)

Parameter	**Description**
string	Required. Specifies the string to reverse

Return Value: **Returns the reversed string**

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

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

</body>
</html>
```

```
!dlroW olleH
```

strripos() Function

Definition and Usage

The strripos() function finds the position of the last occurrence of a string inside another string.

Syntax

strripos(string,find,start)

Parameter	**Description**
string	Required. Specifies the string to search
find	Required. Specifies the string to find
start	Optional. Specifies where to begin the search

Return Value: Returns the position of the last occurrence of a string inside another string, or FALSE if the string is not found. Note: String positions start at 0, and not 1.

The start parameter was added in PHP 5.0

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

<?php
echo strripos("I love php, I love php too!","PHP");
```

```
?>


</body>

</html>

<!DOCTYPE html>

<html>

<body>


<?php

echo strripos("I love php, I love php too!","PHP");

?>


</body>

</html>

19
```

str_replace() Function

Definition and Usage

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

This function works by the following rules:

If the string to be searched is an array, it returns an array

If the string to be searched is an array, find and replace is performed with every array element

If both find and replace are arrays, and replace has fewer elements than find, an empty string will be used as replace

If find is an array and replace is a string, the replace string will be used for every find value

This function is case-sensitive. Use the str_ireplace() function to perform a case-insensitive search.

Syntax

str_replace(find,replace,string,count)

Parameter	**Description**
find	Required. Specifies the value to find
replace	Required. Specifies the value to replace the value in find
string	Required. Specifies the string to be searched
count	Optional. A variable that counts the number of replacements

Return Value: Returns a string or an array with the replaced values

Before PHP 4.3.3, this function experienced trouble when using arrays as both find and replace parameters, which caused empty find indexes to be skipped without advancing the internal pointer on the replace array. Newer versions will not have this problem.

As of PHP 4.0.5, most of the parameters can now be an array

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

```
<p>Search the string "Hello World!", find the value "world" and replace it with "Peter":</p>

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

</body>
</html>
```

Search the string "Hello World!", find the value "world" and replace it with "Peter":

Hello Peter!

echo() Function

Definition and Usage

The echo() function outputs one or more strings.

The echo() function is not actually a function, so you are not required to use parentheses with it. However, if you want to pass more than one parameter to echo(), using parentheses will generate a parse error.

The echo() function is slightly faster than print().

The echo() function also has a shortcut syntax. Prior to PHP 5.4.0, this syntax only works with the short_open_tag configuration setting enabled.

Syntax

echo(strings)

Parameter Description

strings Required. One or more strings to be sent to the output

```
<!DOCTYPE html>
<html>
<body>
<?php
echo "Hello world!";
?>
</body>
</html>
OUTPUT
Hello world!
```

print() Function

Definition and Usage

The print() function outputs one or more strings.

The print() function is not actually a function, so you are not required to use parentheses with it.

The print() function is slightly slower than echo().

Syntax

print(strings)

Parameter	Description
strings	Required. One or more strings to be sent to the output

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

<?php
print "Hello world!";
?>
OUTPUT
</body>
</html>
Hello world!
```

trim() Function

Definition and Usage

The trim() function removes whitespace and other predefined characters from both sides of a string.

Syntax

trim(string,charlist)

Parameter	Description

string	Required. Specifies the string to check
charlist	Optional. Specifies which characters to remove from the string. If omitted, all of the following characters are removed: "\0" - NULL "\t" - tab "\n" - new line "\x0B" - vertical tab "\r" - carriage return " " - ordinary white space

EXAMPLE

```
<!DOCTYPE html>
<html>
<body>
<?php
$str = "Hello World!";
echo $str . "<br>";
echo trim($str,"Hed!");
?>

</body>
</html>
```

OUTPUT

Hello World!
llo Worl

ltrim() Function

Definition and Usage

The ltrim() function removes whitespace or other predefined characters from the left side of a string.

Syntax

ltrim(string,charlist)

Parameter	Description
string	Required. Specifies the string to check
charlist	Optional. Specifies which characters to remove from the string. If omitted, all of the following characters are removed: "\0" - NULL "\t" - tab "\n" - new line "\x0B" - vertical tab "\r" - carriage return " " - ordinary white space

example

```
<!DOCTYPE html>
<html>
<body>
<?php
$str = "Hello World!";
echo $str . "<br>";
```

```
echo ltrim($str,"Hello");

?>

</body>

</html>
```

Output

Hello World!
World!

rtrim() Function

Definition and Usage

The rtrim() function removes whitespace or other predefined characters from the right side of a string.

Syntax

rtrim(string,charlist)

Parameter	Description
string	Required. Specifies the string to check
charlist	Optional. Specifies which characters to remove from the string. If omitted, all of the following characters are removed: "\0" - NULL "\t" - tab "\n" - new line "\x0B" - vertical tab "\r" - carriage return " " - ordinary white space

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


<?php
$str = "Hello World!";
echo $str . "<br>";
echo rtrim($str,"World!");
?>


</body>
</html>
```

Example

```
<!DOCTYPE html>
<html>
<body>
?php
echo strlen("Hello");
?>
</body>
</html>
output
```

5

c. math functions

abs()

The abs() function returns the absolute (positive) value of a number.

Syntax

abs(number);

Parameter	**Description**
number	Required. Specifies a number. If the number is of type float, the return type is also float, otherwise it is integer

Return Value: The absolute value of the number

Return Type: Float / Integer

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
echo(abs(6.7) . "<br>");
echo(abs(-6.7) . "<br>");
echo(abs(-3) . "<br>");
```

```
echo(abs(3));

?>

</body>

</html>
```

Out put

6.7
6.7
3
3

ceil() Function

Definition and Usage

The ceil() function rounds a number UP to the nearest integer, if necessary.

Syntax

ceil(number);

Parameter	**Description**
number	Required. Specifies the value to round up

Return Value: The value rounded up to the nearest integer

Return Type: Float

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
echo(ceil(0.60) . "<br>");
echo(ceil(0.40) . "<br>");
echo(ceil(5) . "<br>");
echo(ceil(5.1) . "<br>");
echo(ceil(-5.1) . "<br>");
echo(ceil(-5.9));
?>

</body>
</html>
```

1
1
5
6
-5
-5

floor() Function

Definition and Usage

The floor() function rounds a number DOWN to the nearest integer, if necessary.

Syntax

floor(number);

Parameter	**Description**
number	Required. Specifies the value to round down

Return Value: The value rounded down to the nearest integer

Return Type: Float

Example

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

<?php
echo(floor(0.60) . "<br>");
echo(floor(0.40) . "<br>");
echo(floor(5) . "<br>");
echo(floor(5.1) . "<br>");
echo(floor(-5.1) . "<br>");
echo(floor(-5.9));
?>
```

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

Output

0
0
5
5
-6
-6

round() Function

Definition and Usage

The round() function rounds a floating-point number.

Example

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

<?php
echo(round(0.60) . "<br>");
echo(round(0.50) . "<br>");
echo(round(0.49) . "<br>");
echo(round(-4.40) . "<br>");
echo(round(-4.60));
?>
```

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

Output

1
1
0
-4
-5

fmod() Function

Definition and Usage

The fmod() function returns the remainder (modulo) of x/y.

Syntax

fmod(x,y);

Parameter	**Description**
x	Required. Specifies the dividend
y	Required. Specifies the divisor

Return Value: The floating point remainder of x/y

Return Type: Float

Example

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

```
<?php
$x = 7;
$y = 2;
$result = fmod($x,$y);
echo $result;
// $result equals 1, because 2 * 3 + 1 = 7
?>

</body>
</html>
```

Output

1

min() Function

Definition and Usage

The min() function returns the lowest value in an array, or the lowest value of several specified values.

Syntax

min(array_values);

or

min(value1,value2,...);

Parameter	**Description**
array_values	Required. Specifies an array containing the values
value1,value2,...	Required. Specifies the values to compare (must be at least two values)

Return Value: The numerically lowest value

Return Type: Mixed

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
echo(min(2,4,6,8,10) . "<br>");
echo(min(22,14,68,18,15) . "<br>");
echo(min(array(4,6,8,10)) . "<br>");
echo(min(array(44,16,81,12)));
?>
</body>
</html>
```

Output

2
14

4
12

max() Function

Definition and Usage

The max() function returns the highest value in an array, or the highest value of several specified values.

Syntax

max(array_values);

or

max(value1,value2,...);.

Parameter	**Description**
array_values	Required. Specifies an array containing the values
value1,value2,...	Required. Specifies the values to compare (must be at least two values)

Return Value: The numerically highest value

Return Type: Mixed

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

```
<?php
echo(max(2,4,6,8,10) . "<br>");
echo(max(22,14,68,18,15) . "<br>");
echo(max(array(4,6,8,10)) . "<br>");
echo(max(array(44,16,81,12)));
?>

</body>
</html>
```

10
68
10
81

pow() Function

Definition and Usage

The pow() function returns x raised to the power of y.

Syntax

pow(x,y);

Parameter	**Description**
x	Required. Specifies the base to use
y	Required. Specifies the exponent

Return Value: x raised to the power of y

Return Type: Integer if possible, Float otherwise

Example

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

<?php
echo(pow(2,4) . "<br>");
echo(pow(-2,4) . "<br>");
echo(pow(-2,-4) . "<br>");
echo(pow(-2,-3.2));
?>

</body>
</html>
```

16
16
0.0625
NAN

sqrt() Function

Definition and Usage

The sqrt() function returns the square root of a number.

Syntax

sqrt(number);

Parameter	**Description**
number	Required. Specifies a number

Return Value: The square root of number, or NAN for negative numbers

Return Type: Float

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
echo(sqrt(0) . "<br>");
echo(sqrt(1) . "<br>");
echo(sqrt(9) . "<br>");
echo(sqrt(0.64) . "<br>");
echo(sqrt(-9));
?>
</body>
</html>
```

Output

0
1
3
0.8
NAN

rand() Function

Definition and Usage

The rand() function generates a random integer.

If you want a random integer between 10 and 100 (inclusive), use rand (10,100).

Syntax

rand();

or

rand(min,max);

Parameter	**Description**
min	Optional. Specifies the lowest number to be returned. Default is 0
max	Optional. Specifies the highest number to be returned. Default is getrandmax()

Return Value: A random integer between min (or 0) and max (or getrandmax() inclusive)

Return Type: Integer

Example

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

<?php
echo(rand() . "<br>");
echo(rand() . "<br>");
echo(rand(10,100));
```

```
?>


</body>

</html>
```

output

1554955389
961018293
32

d. Date/Time Functions

date() Function

Definition and Usage

The date() function formats a local date and time, and returns the formatted date string.

Syntax

date(format, timestamp)

Parameter	**Description**
format	Required. Specifies the format of the outputted date string. The following characters can be used:

d - The day of the month (from 01 to 31)

D - A textual representation of a day (three letters)

j - The day of the month without leading zeros (1 to 31)

l (lowercase 'L') - A full textual representation of a day

N - The ISO-8601 numeric representation of a day (1 for Monday, 7 for Sunday)

S - The English ordinal suffix for the day of the month (2 characters st, nd, rd or th. Works well with j)

w - A numeric representation of the day (0 for Sunday, 6 for Saturday)

z - The day of the year (from 0 through 365)

W - The ISO-8601 week number of year (weeks starting on Monday)

F - A full textual representation of a month (January through December)

m - A numeric representation of a month (from 01 to 12)

M - A short textual representation of a month (three letters)

n - A numeric representation of a month, without leading zeros (1 to 12)

t - The number of days in the given month

L - Whether it's a leap year (1 if it is a leap year, 0 otherwise)

o - The ISO-8601 year number

Y - A four digit representation of a year

y - A two digit representation of a year

a - Lowercase am or pm

A - Uppercase AM or PM

B - Swatch Internet time (000 to 999)

g - 12-hour format of an hour (1 to 12)

G - 24-hour format of an hour (0 to 23)

h - 12-hour format of an hour (01 to 12)

H - 24-hour format of an hour (00 to 23)

i - Minutes with leading zeros (00 to 59)

s - Seconds, with leading zeros (00 to 59)

u - Microseconds (added in PHP 5.2.2)

e - The timezone identifier (Examples: UTC, GMT, Atlantic/Azores)

I (capital i) - Whether the date is in daylights savings time (1 if Daylight Savings Time, 0 otherwise)

O - Difference to Greenwich time (GMT) in hours (Example: +0100)

P - Difference to Greenwich time (GMT) in hours:minutes (added in PHP 5.1.3)

T - Timezone abbreviations (Examples: EST, MDT)

Z - Timezone offset in seconds. The offset for timezones west of UTC is negative (-43200 to 50400)

c - The ISO-8601 date (e.g. 2013-05-05T16:34:42+00:00)

r - The RFC 2822 formatted date (e.g. Fri, 12 Apr 2013 12:01:05 +0200)

U - The seconds since the Unix Epoch (January 1 1970 00:00:00 GMT)

Return Value: Returns a formatted date string on success. FALSE on failure + an E_WARNING

Example

```
<!DOCTYPE html>

<html>

<body>


<?php

// Prints the day

echo date("l") . "<br>";

// Prints the day, date, month, year, time, AM or PM

echo date("l jS \of F Y h:i:s A") . "<br>";

// Prints October 3, 1975 was on a Friday

echo "Oct 3,1975 was on a ".date("l", mktime(0,0,0,10,3,1975)) . "<br>";

// Use a constant in the format parameter

echo date(DATE_RFC822) . "<br>";
```

```
// prints something like: 1975-10-03T00:00:00+00:00

echo date(DATE_ATOM,mktime(0,0,0,10,3,1975));

?>

</body>

</html>
```

Output

```
Saturday
Saturday 24th of August 2019 05:42:30 AM
Oct 3,1975 was on a Friday
Sat, 24 Aug 19 05:42:30 +0000
1975-10-03T00:00:00+00:00
```

getdate() Function

Definition and Usage

The getdate() function returns date/time information of a timestamp or the current local date/time.

Syntax

getdate(timestamp)

Parameter	**Description**
timestamp	Optional. Specifies an integer Unix timestamp. Default is the current local time (time())

Return Value: Returns an associative array with information related to the timestamp:

[seconds] - seconds

[minutes] - minutes

[hours] - hours

[mday] - day of the month

[wday] - day of the week (0=Sunday, 1=Monday,...)

[mon] - month

[year] - year

[yday] - day of the year

[weekday] - name of the weekday

[month] - name of the month

[0] - seconds since Unix Epoch

Example

<!DOCTYPE html>

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

```
<?php
// Print the array from getdate()
print_r(getdate());
echo "<br><br>";
// Return date/time info of a timestamp; then format the output
$mydate=getdate(date("U"));
echo "$mydate[weekday], $mydate[month] $mydate[mday], $mydate[year]";
?>
</body>
```

```
</html>
```

Output

Array ([seconds] => 28 [minutes] => 53 [hours] => 5 [mday] => 24 [wday] => 6 [mon] => 8 [year] => 2019 [yday] => 235 [weekday] => Saturday [month] => August [0] => 1566626008)

Saturday, August 24, 2019

setDate() Function

date_date_set

DateTime::setDate -- date_date_set — Sets the date

Description

Object oriented style

public DateTime::setDate (int $year , int $month , int $day) : DateTime

Procedural style

date_date_set (DateTime $object , int $year , int $month , int $day) : DateTime

Resets the current date of the DateTime object to a different date.

Parameters

object

Procedural style only: A DateTime object returned by date_create(). The function modifies this object.

year

Year of the date.

month

Month of the date.

day

Day of the date.

Return Values

Returns the DateTime object for method chaining or FALSE on failure.

Examples

Example #1 DateTime::setDate() example

Object oriented style

```
<?php
$date = new DateTime();
$date->setDate(2001, 2, 3);
echo $date->format('Y-m-d');
?>
```

Procedural style

```
<?php
$date = date_create();
date_date_set($date, 2001, 2, 3);
echo date_format($date, 'Y-m-d');
?>
```

The above examples will output:

2001-02-03

checkdate() Function

Definition and Usage

The checkdate() function is used to validate a Gregorian date.

Syntax

checkdate(month, day, year)

Parameter	**Description**
month	Required. Specifies the month as a number between 1 and 12
day	Required. Specifies the day as a number between 1 and 31
year	Required. Specifies the year as a number between 1 and 32767

Return Value: TRUE if the date is valid. FALSE otherwise

```
<!DOCTYPE html>
<html>
<body>
Example
<?php
var_dump(checkdate(12,31,-400));
echo "<br>";
var_dump(checkdate(2,29,2003));
echo "<br>";
```

```
var_dump(checkdate(2,29,2004));

?>


</body>

</html>
```

Output

bool(false)
bool(false)
bool(true)

time() Function

Definition and Usage

The time() function returns the current time in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

Syntax

time()

Return Value: Returns an integer containing the current time as a Unix timestamp

Example

```
<!DOCTYPE html>

<html>

<body>


<?php
```

```
$t=time();

echo($t . "<br>");

echo(date("Y-m-d",$t));

?>


</body>

</html>
```

Output

1566626420
2019-08-24

mktime() Function

Definition and Usage

The gmmktime() function returns the Unix timestamp for a date.

Tip: This function is identical to gmmktime() except the passed parameters represents a date (not a GMT date).

Syntax

mktime(hour, minute, second, month, day, year, is_dst)

Parameter	Description
hour	Optional. Specifies the hour
minute	Optional. Specifies the minute
second	Optional. Specifies the second

month Optional. Specifies the month

day Optional. Specifies the day

year Optional. Specifies the year

is_dst Optional. Set this parameter to 1 if the time is during daylight savings time (DST), 0 if it is not, or -1 (the default) if it is unknown. If it's unknown, PHP tries to find out itself (which may cause unexpected results). Note: This parameter is removed in PHP 7.0. The new timezone handling features should be used instead

Exmple

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

<?php
// Prints: October 3, 1975 was on a Friday
echo "Oct 3, 1975 was on a ".date("l", mktime(0,0,0,10,3,1975)) . "<br><br>";

//The mktime() function is useful for doing date arithmetic and validation.
//It will automatically calculate the correct value for out-of-range input:
echo date("M-d-Y",mktime(0,0,0,12,36,2001)) . "<br>";
echo date("M-d-Y",mktime(0,0,0,14,1,2001)) . "<br>";
echo date("M-d-Y",mktime(0,0,0,1,1,2001)) . "<br>";
echo date("M-d-Y",mktime(0,0,0,1,1,99)) . "<br>";
?>

</body>
</html>
```

Oct 3, 1975 was on a Friday

Jan-05-2002
Feb-01-2002
Jan-01-2001
Jan-01-1999

Return Value: Returns an integer Unix timestamp. FALSE on error

e. Array Functions

array() Function

Definition and Usage

The array() function is used to create an array.

In PHP, there are three types of arrays:

Indexed arrays - Arrays with numeric index

Associative arrays - Arrays with named keys

Multidimensional arrays - Arrays containing one or more arrays

Syntax

Syntax for indexed arrays:

array(value1, value2, value3, etc.)

Syntax for associative arrays:

array(key=>value,key=>value,key=>value,etc.)

Parameter	Description
key	Specifies the key (numeric or string)
value	Specifies the value

Example

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

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

</body>
</html>
```

Output

I like Volvo, BMW and Toyota.

Return Value: Returns an array of the parameters

Asosiative Array

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

There are two ways to create an associative array:

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

or:

$age['Peter'] = "35";

$age['Ben'] = "37";

$age['Joe'] = "43";

The named keys can then be used in a script:

Example

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

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

Output

Peter is 35 years old.

count() Function

Definition and Usage

The count() function returns the number of elements in an array.

Syntax

count(array, mode)

Parameter	**Description**
array	Required. Specifies the array
mode	Optional. Specifies the mode. Possible values:

0 - Default. Does not count all elements of multidimensional arrays

1 - Counts the array recursively (counts all the elements of multidimensional arrays)

Example

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

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

output

3

list() Function

Definition and Usage

The list() function is used to assign values to a list of variables in one operation.

This function only works on numerical arrays.

Syntax

list(var1, var2, ...)

Parameter	**Description**
var1	Required. The first variable to assign a value to
var2,...	Optional. More variables to assign values to

Return Value: Returns the assigned array

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

```
<body>
<?php
$my_array = array("Dog","Cat","Horse");
list($a, $b, $c) = $my_array;
echo "I have several animals, a $a, a $b and a $c.";
?>
</body>
</html>
```

Output

I have several animals, a Dog, a Cat and a Horse.

in_array() Function

Definition and Usage

The in_array() function searches an array for a specific value.

If the search parameter is a string and the type parameter is set to TRUE, the search is case-sensitive.

Syntax

in_array(search, array, type)

Parameter	**Description**
search	Required. Specifies the what to search for
array	Required. Specifies the array to search

type	Optional. If this parameter is set to TRUE, the in_array() function searches for the search-string and specific type in the array.

```
<!DOCTYPE html>
<html>
<body>
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");


if (in_array("Glenn", $people))
  {
  echo "Match found";
  }
else
  {
  echo "Match not found";
  }
?>
</body>
</html>
```

Output

Match found

Return Value: Returns TRUE if the value is found in the array, or FALSE otherwise

current() Function

Definition and Usage

The current() function returns the value of the current element in an array.

Every array has an internal pointer to its "current" element, which is initialized to the first element inserted into the array.

Tip: This function does not move the arrays internal pointer.

Syntax

current(array)

Parameter	**Description**
array	Required. Specifies the array to use

Example

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

<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");

echo current($people) . "<br>";
?>

</body>
</html>
```

Output

Peter

next() Function

Definition and Usage

The next() function moves the internal pointer to, and outputs, the next element in the array.

Syntax

next(array)

Parameter	**Description**
array	Required. Specifies the array to use

Example

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

```
<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");

echo current($people) . "<br>";
echo next($people);
?>

</body>
```

```
</html>
```

Output

Peter
Joe

Return Value: Returns the value of the last element in the array on success, or FALSE if the array is empty

prev() Function

Definition and Usage

The prev() function moves the internal pointer to, and outputs, the previous element in the array.

Syntax

prev(array)

Parameter	**Description**
array	Required. Specifies the array to use

Example

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

<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
echo current($people) . "<br>";
```

```
echo next($people) . "<br>";

echo prev($people);

?>

</body>

</html>
```

output

Peter
Joe
Peter

Return Value: Returns the value of the previous element in the array on success, or FALSE if there are no more elements

reset() Function

Definition and Usage

The reset() function moves the internal pointer to the first element of the array.

Syntax

reset(array)

Parameter	**Description**
array	Required. Specifies the array to use

Example

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

<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
echo current($people) . "<br>";
echo next($people) . "<br>";
echo reset($people);
?>
</body>
</html>
```

Output

Peter
Joe
Peter

Return Value: Returns the value of the first element in the array on success, or FALSE if the array is empty

each() Function

Definition and Usage

The each() function returns the current element key and value, and moves the internal pointer forward.

The each() function is deprecated in PHP 7.2.

This element key and value is returned in an array with four elements. Two elements (1 and Value) for the element value, and two elements (0 and Key) for the element key.

Syntax

each(array)

Parameter	**Description**
array	Required. Specifies the array to use

Return Value: Returns the current element key and value. This element key and value is returned in an array with four elements. Two elements (1 and Value) for the element value, and two elements (0 and Key) for the element key. This function returns FALSE if there are no more array elements

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

<?php
$people = array("Peter", "Joe", "Glenn", "Cleveland");
print_r (each($people));
?>

</body>
</html>
```

Output

Array ([1] => Peter [value] => Peter [0] => 0 [key] => 0)

sort() Function

Definition and Usage

The sort() function sorts an indexed array in ascending order.

Syntax

sort(array, sorttype)

Parameter	**Description**
array	Required. Specifies the array to sort
sorttype	Optional. Specifies how to compare the array elements/items. Possible values: 0 = SORT_REGULAR - Default. Compare items normally (don't change types) 1 = SORT_NUMERIC - Compare items numerically 2 = SORT_STRING - Compare items as strings 3 = SORT_LOCALE_STRING - Compare items as strings, based on current locale 4 = SORT_NATURAL - Compare items as strings using natural ordering 5 = SORT_FLAG_CASE –

Example

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

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

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

Output

BMW
Toyota
Volvo

rsort() Function

Definition and Usage

The rsort() function sorts an indexed array in descending order.

Syntax

rsort(array, sorttype)

Parameter	Description
array	Required. Specifies the array to sort
sorttype	Optional. Specifies how to compare the array elements/items. Possible values: 0 = SORT_REGULAR - Default. Compare items normally (don't change types) 1 = SORT_NUMERIC - Compare items numerically 2 = SORT_STRING - Compare items as strings 3 = SORT_LOCALE_STRING - Compare items as strings, based on current locale 4 = SORT_NATURAL - Compare items as strings using natural ordering 5 = SORT_FLAG_CASE –

Example

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

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

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

```
 echo "<br>";
 }
?>


</body>
</html>
```

Output

Volvo
Toyota
BMW

arsort() Function

Definition and Usage

The arsort() function sorts an associative array in descending order, according to the value.

Syntax

arsort(array, sorttype)

Parameter	**Description**
array	Required. Specifies the array to sort
sorttype	Optional. Specifies how to compare the array elements/items. Possible values: 0 = SORT_REGULAR - Default. Compare items normally (don't change types) 1 = SORT_NUMERIC - Compare items numerically

2 = SORT_STRING - Compare items as strings

3 = SORT_LOCALE_STRING - Compare items as strings, based on current locale

4 = SORT_NATURAL - Compare items as strings using natural ordering

5 = SORT_FLAG_CASE –

Example

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

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

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

</body>
```

```
</html>
```

Output

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

Return Value: TRUE on success. FALSE on failure

array_merge() Function

Definition and Usage

The array_merge() function merges one or more arrays into one array.

You can assign one array to the function, or as many as you like.

If two or more array elements have the same key, the last one overrides the others.

If you assign only one array to the array_merge() function, and the keys are integers, the function returns a new array with integer keys starting at 0 and increases by 1 for each value (See example below).

The difference between this function and the array_merge_recursive() function is when two or more array elements have the same key. Instead of override the keys, the array_merge_recursive() function makes the value as an array.

Syntax

array_merge(array1, array2, array3, ...)

Parameter	**Description**
array1	Required. Specifies an array
array2	Optional. Specifies an array
array3,...	Optional. Specifies an array

Return Value: Returns the merged array

Example

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

<?php
$a1=array("red","green");
$a2=array("blue","yellow");
print_r(array_merge($a1,$a2));
?>

</body>
</html>
```

Output

Array ([0] => red [1] => green [2] => blue [3] => yellow)

array_reverse() Function

Definition and Usage

The array_reverse() function returns an array in the reverse order.

Syntax

array_reverse(array, preserve)

Parameter	**Description**
array	Required. Specifies an array
preserve	Optional. Specifies if the function should preserve the keys of the array or not. Possible values: true - false

Exampe

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

<?php
$a=array("a"=>"Volvo","b"=>"BMW","c"=>"Toyota");
```

```
print_r(array_reverse($a));
?>


</body>
</html>
```

Output

Array ([c] => Toyota [b] => BMW [a] => Volvo)

f. Filesystem Functions

move_uploaded_file() Function

Definition and Usage

The move_uploaded_file() function moves an uploaded file to a new destination.

This function only works on files uploaded via PHP's HTTP POST upload mechanism.

If the destination file already exists, it will be overwritten.

Syntax

move_uploaded_file(file, dest)

Parameter	**Description**
file	Required. Specifies the filename of the uploaded file
dest	Required. Specifies the new location for the file

Return Value: TRUE on success, FALSE on failure

Example

Uploading multiple files

```
<?php
$uploads_dir = '/uploads';
foreach ($_FILES["pictures"]["error"] as $key => $error) {
    if ($error == UPLOAD_ERR_OK) {
        $tmp_name = $_FILES["pictures"]["tmp_name"][$key];
        // basename() may prevent filesystem traversal attacks;
        // further validation/sanitation of the filename may be appropriate
        $name = basename($_FILES["pictures"]["name"][$key]);
        move_uploaded_file($tmp_name, "$uploads_dir/$name");
    }
}
?>
```

fopen() Function

Definition and Usage

The fopen() function opens a file or URL.

When writing to a text file, be sure to use the correct line-ending character! Unix systems use \n, Windows systems use \r\n, and Macintosh systems use \r as the line ending character. Windows offers a translation flag ('t') which will translate \n to \r\n when working with the file. You can also use 'b' to force binary mode. To use these flags, specify either 'b' or 't' as the last character of the mode parameter.

Syntax

fopen(filename, mode, include_path, context)

Parameter Description

filename Required. Specifies the file or URL to open

mode Required. Specifies the type of access you require to the file/stream.

Possible values:

"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 truncates the file; or creates a new file if it doesn't exist. Place file pointer at the beginning of the file

"w+" - Read/Write. Opens and truncates the file; or creates a new file if it doesn't exist. Place file pointer at the beginning of the file

"a" - Write only. Opens and writes to the end of the file or creates a new file if it doesn't exist

"a+" - Read/Write. 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

"c" - Write only. Opens the file; or creates a new file if it doesn't exist. Place file pointer at the beginning of the file

"c+" - Read/Write. Opens the file; or creates a new file if it doesn't exist. Place file pointer at the beginning of the file

"e" - Only available in PHP compiled on POSIX.1-2008 conform systems.

nclude_path Optional. Set this parameter to '1' if you want to search for the file in the include_path (in php.ini) as well

contextOptional. Specifies the context of the file handle. Context is a set of options that can modify the behavior of a stream

example

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

<?php
$file = fopen("test.txt","r");
//Output lines until EOF is reached
while(! feof($file)) {
  $line = fgets($file);
  echo $line. "<br>";
}

fclose($file);
?>

</body>
</html>
```

Output

Hello, this is a test file.
There are three lines here.
This is the last line.

Return Value: TRUE on success, FALSE and an error on failure. You can hide the error by adding an "@" in front of the function name.

fwrite() Function

Definition and Usage

The fwrite() writes to an open file.

The function will stop at the end of the file (EOF) or when it reaches the specified length, whichever comes first.

Syntax

fwrite(file, string, length)

Parameter	Description
file	Required. Specifies the open file to write to
string	Required. Specifies the string to write to the open file
length	Optional. Specifies the maximum number of bytes to write

Example

```
<?php
$file = fopen("test.txt","w");
echo fwrite($file,"Hello World. Testing!");
fclose($file);
?>
```

Output

Characters writtem in file. 21 characters

fread() Function

Definition and Usage

The fread() reads from an open file.

The function will stop at the end of the file or when it reaches the specified length, whichever comes first.

Syntax

fread(file, length)

Parameter	Description
file	Required. Specifies the open file to read from
length	Required. Specifies the maximum number of bytes to read

Example

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

<?php
$file = fopen("test.txt","r");
$data=fread($file,"10");
echo $data;

fclose($file);
?>

</body>
</html>
```

Output

file data will be in $data variable and it will be printed by echo statement.

fclose() Function

Definition and Usage

The fclose() function closes an open file.

The file must have been opened by fopen() or fsockopen().

Syntax

fclose(file)

Parameter	Description
file	Required. Specifies the file to close

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
$file = fopen("test.txt","r");
fclose($file);
?>

</body>
</html>
```

Return Value: TRUE on success, FALSE on failure

feof() Function

Definition and Usage

The feof() function checks if the "end-of-file" (EOF) has been reached for an open file.

This function is useful for looping through data of unknown length.

Syntax

feof(file)

Parameter	Description
file	Required. Specifies an open file to check

Example

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

<?php
$file = fopen("test.txt","r");
```

```
//Output lines until EOF is reached
while(! feof($file)) {
  $line = fgets($file);
  echo $line. "<br>";
}

fclose($file);
?>

</body>
</html>
```

Output

Hello, this is a test file.
There are three lines here.
This is the last line.

file_exists() Function

Definition and Usage

The file_exists() function checks whether a file or directory exists.

The result of this function is cached. Use clearstatcache() to clear the cache.

Syntax

file_exists(path)

Parameter	**Description**
path	Required. Specifies the path to the file or directory to check

Result Size: 945 x 771

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

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

</body>
</html>
```

Output

1(true)-file exists

Return Value: TRUE if the file or directory exists, FALSE on failure

fgetc() Function

Definition and Usage

The fgetc() function returns a single character from an open file.

This function is slow and should not be used on large files. If you need to read one character at a time from a large file, use fgets() to read data one line at a time and then process the line one single character at a time with fgetc().

Syntax

fgetc(file)

Parameter	**Description**
file	Required. Specifies the open file to return a single character from

Example

```
<!DOCTYPE html>
```

```
<html>
<body>

<?php
$file = fopen("test.txt","r");
echo fgetc($file);
fclose($file);
?>

</body>
</html>
```

Output

H

fgets() Function

Definition and Usage

The fgets() function returns a line from an open file.

Syntax

fgets(file, length)

Parameter	**Description**
file	Required. Specifies the open file to return a line from
length	Optional. Specifies the number of bytes to read. Reading stops when length-1 bytes have been reached, or when a new line occurs, or on EOF. If no length is specified, it reads until end of the line

Example

Result Size: 945 x 820

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

<?php
$file = fopen("test.txt","r");
echo fgets($file);
fclose($file);
?>

</body>
</html>
```

Output

Hello, this is a test file.

fseek() Function

Definition and Usage

The fseek() function seeks in an open file.

This function moves the file pointer from its current position to a new position, forward or backward, specified by the number of bytes.

You can find the current position by using ftell()!

Syntax

fseek(file, offset, whence)

Parameter	**Description**
file	Required. Specifies the open file to seek in

offset	Required. Specifies the new position (measured in bytes from the beginning of the file)
whence	Optional. Possible values: SEEK_SET - Set position equal to offset. Default SEEK_CUR - Set position to current location plus offset SEEK_END - Set position to EOF plus offset (to move to a position before EOF, the offset must be a negative value)

Example

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

<?php
$file = fopen("test.txt","r");
// Read first line
echo fgets($file);
// Move back to beginning of file
fseek($file,0);
fclose($file);
?>

</body>
</html>
```

Output

Hello, this is a test file.

ftell() Function

Definition and Usage

The ftell() function returns the current position of the read/write pointer in an open file.

Syntax

ftell(file)

Parameter	Description
file	Required. Specifies the open file to check

example

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

<?php
$file = fopen("test.txt","r");

// Print current position
echo ftell($file);

// Change current position
fseek($file,"15");

// Print current position again
echo "<br>" . ftell($file);

fclose($file);
?>

</body>
</html>
```

Output

0
15

is_uploaded_file() Function

Definition and Usage

The is_uploaded_file() function checks whether the specified file is uploaded via HTTP POST.

Syntax

is_uploaded_file(file)

Parameter	**Description**
file	Required. Specifies the path to the file to check

Example

```
<?php
$file = "test.txt";
if(is_uploaded_file($file)) {
  echo ("$file is uploaded via HTTP POST");
} else {
  echo ("$file is not uploaded via HTTP POST");
}
?>
```

Output

test.txt is not uploaded via HTTP POST

g. Miscellaneous Functions

Include() and require()

The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

PHP include and require Statements

It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement.

The include and require statements are identical, except upon failure:

require will produce a fatal error (E_COMPILE_ERROR) and stop the script

include will only produce a warning (E_WARNING) and the script will continue

So, if you want the execution to go on and show users the output, even if the include file is missing, use the include statement. Otherwise, in case of FrameWork, CMS, or a complex PHP application coding, always use the require statement to include a key file to the flow of execution. This will help avoid compromising your application's security and integrity, just in-case one key file is accidentally missing.

Including files saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

Syntax

include 'filename';

or

require 'filename';

Example

```
<html>
<body>

<div class="menu">
<?php include 'menu.php';?>
```

```
</div>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>

</body>
</html>
```

include vs. require

The require statement is also used to include a file into the PHP code.

However, there is one big difference between include and require; when a file is included with the include statement and PHP cannot find it, the script will continue to execute:

Example

```
<html>
<body>
<h1>Welcome to my home page!</h1>
<?php include 'noFileExists.php';
echo "I have a $color $car.";
?>
</body>
</html>
```

If we do the same example using the require statement, the echo statement will not be executed because the script execution dies after the require statement returned a fatal error:

Example

```
<html>
<body>
<h1>Welcome to my home page!</h1>
<?php require 'noFileExists.php';
echo "I have a $color $car.";
?>
</body>
</html>
```

require_once(), include_once()

Description

require_once()

this function can be used to include a php file in another one, when you may need to include the called file more than once. If it is found that the file has already been included, calling script is going to ignore further inclusions.

If a.php is a php script calling b.php with require_once() statement, and does not find b.php, a.php stops executes causing a fatal error.

Syntax:

require_once('name of the calling file with path');

Example:

```
<?php
echo "today is:".date("Y-m-d");
?>
```

The above file is abc.php

The above file abc.php, is included twice with require_once() statement in the following file y.php. But from the output you will get that the second instance of inclusion is ignored, since require_once() statement ignores all the similar inclusions after the first one.

```
<?php
require_once('abc.php');
require_once('abc.php');
?>
```

If a calling script does not find a called script with the require_once statement, it halts the execution of the calling script.

include_once()

Description

The include_once() statement can be used to include a php file in another one, when you may need to include the called file more than once. If it is found that the file has already been included, calling script is going to ignore further inclusions.

If a.php is a php script calling b.php with include_once() statement, and does not find b.php, a.php executes with a warning, excluding the part of the code written within b.php.

Syntax:

include_once('name of the called file with path');

Example:

```
<?php
echo "today is:".date("Y-m-d");
?>
```

The above file is abc.php

The above file abc.php, is included twice with include_once() statement in the following file y.php. But from the output you will get that the second instance of inclusion is ignored, since include_once() statement ignores all the similar inclusions after the first one.

```
<?php
include_once('abc.php');
include_once('abc.php');
?>
```

header() Function

PHP headers can perform certain things, some of them are listed below:

- Tell browser not to cache the pages.
- Content-Type declaration
- Page Redirection

age Redirecting using header()

You can redirect your user to some other page.

```
<?php
  header("Location: http://www.example.com/");
?>
```

Please note that Location starts with capital L, some browsers might not redirect if small l is used.

Even though you redirected the user successfully, yet this is not the proper way to do it. The above command does not generate the 301 response which means the target page will lose a hit count and SEO ranking. To avoid that, you have to add an additional header.

```
<?php
  header("HTTP/1.1 301 Moved Permanently");
  header("Location: http://www.example.com/");
?>
```

Furthermore, you can add some redirection interval by using the following code:

```
<?php
  header("Refresh: 5; url=http://www.example.com"); //will redirect after 5 seconds
?>
```

Do not cache pages using header()

You can prevent the browser to cache pages by using the following code:

```
<?php
  //Date in the past, tells the browser that the cache has expired
  header("Expires: Mon, 20 Feb 2005 20:12:03 GMT");

  /* The following tell the browser that the last modification is right not so it must load the page again */  header("Last-Modified: ". gmdate("D, d M Y H:i:s"). "GMT");

  //HTTP/1.0
  header("Pragma: no-cache");
?>
```

The above code will let the browser load the page fresh from the server every time it is called. The reason is simple; we tell the browser that the content just expired and it was just last modified too. Furthermore, we also tell it Pragma: no-cache to make sure it gets a fresh server copy each time.

Content Types using header()

Other than HTML, you can also generate different types of data, e.g., you can parse an image, zip, XML, JSON, etc. If you do that then, you need to tell to the browser that content type is something else.

```
<?php
  //Browser will deal page as PDF
  header ( "Content-type: application/pdf" );

  //myPDF.pdf will called
  header ( "Content-Disposition: attachment; filename=myPDF.pdf' " );
```

```
?>
```

Example

Let the user be prompted to save a generated PDF file (Content-Disposition header is used to supply a recommended filename and force the browser to display the save dialog box):

```
<html>
<body>
<?php
header("Content-type:application/pdf");
// It will be called downloaded.pdf
header("Content-Disposition:attachment;filename='downloaded.pdf'");
// The PDF source is in original.pdf
readfile("original.pdf");
?>
</body>
</html>
```

Example

Send three HTTP headers to prevent page caching:

```
<?php
// Date in the past
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
header("Cache-Control: no-cache");
header("Pragma: no-cache");
?>

<html>
<body>
```

die() Function

Definition and Usage

The die() function is an alias of the exit() function.

Syntax

die(message)

Parameter	Description
message	Required. A message or status number to print before terminating the script. A status number will not be written to the output, just used as the exit status.

Example

```
<?php
$site = "https://www.w3schools.com/";
fopen($site,"r")
or die("Unable to connect to $site");
?>
```

Output

to connect to https://www.w3schools.com/ and program execution stops

16. Form Handling passing data from html form to php file

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

PHP - A Simple HTML Form

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

Example

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

Output

Name:
E-mail:
Submit

Php file that receives data from form

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

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

Welcome.php

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

The output could be something like this:

Welcome John
Your email address is john.doe@example.com

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

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

Php file which receives form data

welcome_get.php

```
<html>
  <body>

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

  </body>
</html>
```

What is the Difference Between GET and POST Method in PHP?

GET vs POST Method in PHP	
GET is a method that sends information by appending them to the page request.	POST is a method that transfers information via HTTP header.
URL	
The form information is visible in the URL	The form information is not visible in the URL
Information Amount	
Limited amount of information is sent. It is less than 1500 characters.	Unlimited amount of information is sent.
Usage	
Helps to send non-sensitive data	Helps to send sensitive data (passwords), binary data (word documents, images)and uploading files
Security	
Not very secure.	More secure.
Bookmarking the Page	
Possible to bookmark the page	Not possible to bookmark the page
Form data type constraints	
Only ASCII characters are permitted.	No constraints, even binary data is permitted.

Summary – GET vs POST Method in PHP

This article discussed two important methods of form handling in PHP. They are GET and POST methods. Generally speaking, developers prefer POST method for sending data than using the GET method. The key difference Between GET and POST method in PHP is that GET method

sends the information by appending them to the page request while POST method sends information via HTTP header.

17. mysqli Functions

a. mysqli_connect()

Definition and Usage

The mysqli_connect() function opens a new connection to the MySQL server.

Syntax

mysqli_connect(host, username, password, dbname, port, socket)

Parameter	Description
Host	Optional. Specifies a host name or an IP address
username	Optional. Specifies the MySQL username
password	Optional. Specifies the MySQL password
dbname	Optional, Specifies the default database to be used
port	Optional. Specifies the port number to attempt to connect to the MySQL server
socket	Optional. Specifies the socket or named pipe to be used

Return Value: Returns an object representing the connection to the MySQL server

Example

Open a new connection to the MySQL server:

```
<?php
$con = mysqli_connect("localhost","root","","ksv");
        // Check connection
        if (mysqli_connect_errno())
                {
                        echo "Failed to connect to MySQL: " . mysqli_connect_error();
                }
?>
```

mysqli_connect_errno() Function

Return an error code from the last connection error, if any:

```
<?php
$con=mysqli_connect("localhost","wrong_user","my_password","my_db");
// Check connection
        if (!$con)
        {
                die("Connection error: " . mysqli_connect_errno());
         }
?>
```

Definition and Usage

The mysqli_connect_errno() function returns the error code from the last connection error, if any.

Syntax

mysqli_connect_errno();

Return Value: Returns an error code value. Zero if no error occurred

mysqli_connect_error() Function

Return an error description from the last connection error, if any:

```
<?php
$con=mysqli_connect("localhost","wrong_user","my_password","my_db");
        // Check connection
        if (!$con)
                {
                        die("Connection error: " . mysqli_connect_error());
                }
?>
```

Definition and Usage

The mysqli_connect_error() function returns the error description from the last connection error, if any.

Syntax

mysqli_connect_error();

Return Value: Returns a string that describes the error. NULL if no error occurred

mysqli_connect_errno() Function

Return an error code from the last connection error, if any:

```
<?php
$con=mysqli_connect("localhost","wrong_user","my_password","my_db");
        // Check connection
```

```
        if (!$con)
                {
                        die("Connection error: " . mysqli_connect_errno());
                }
?>
```

Definition and Usage

The mysqli_connect_errno() function returns the error code from the last connection error, if any.

Syntax

mysqli_connect_errno();

Return Value: Returns an error code value. Zero if no error occurred

mysqli_query() Function

Definition and Usage

The mysqli_query() function performs a query against the database.

Syntax

mysqli_query(connection,query,resultmode);

Parameter **Description**

connection	Required. Specifies the MySQL connection to use
query Required.	Specifies the query string
resultmode	
Optional.	A constant. Either:

MYSQLI_USE_RESULT (Use this if we have to retrieve large amount of data)

MYSQLI_STORE_RESULT (This is default)

Return Value: For successful SELECT, SHOW, DESCRIBE, or EXPLAIN queries it will return a mysqli_result object. For other successful queries it will return TRUE. FALSE on failure

Example

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
      if (mysqli_connect_errno())
          {
                    echo "Failed to connect to MySQL: " . mysqli_connect_error();
          }

// Perform queries


mysqli_query($con,"INSERT INTO Persons (FirstName,LastName,Age)
VALUES ('Glenn','Quagmire',33)");


$result=mysqli_query($con,"SELECT * FROM Persons");

mysqli_close($con);
?>
```

mysqli_affected_rows() Function

Definition and Usage

The mysqli_affected_rows() function returns the number of affected rows in the previous SELECT, INSERT, UPDATE, REPLACE, or DELETE query.

Syntax

mysqli_affected_rows(connection)

Parameter	**Description**
Connection	Required. Specifies the MySQL connection to use
Return Value:	The number of rows affected. -1 indicates that the query returned an error

Example

Return number of affected rows from different queries:

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
    if (mysqli_connect_errno())
        {
        echo "Failed to connect to MySQL: " . mysqli_connect_error();
         }

// Perform queries and print out affected rows.


mysqli_query($con,"SELECT * FROM Persons");
echo "Affected rows: " . mysqli_affected_rows($con);

mysqli_query($con,"DELETE FROM Persons WHERE Age>32");
echo "Affected rows: " . mysqli_affected_rows($con);
```

```
mysqli_close($con);
?>
```

mysqli_fetch_array() Function

Definition and Usage

The mysqli_fetch_array() function fetches a result row as an associative array, a numeric array, or both.

Note: Fieldnames returned from this function are case-sensitive.

Syntax

mysqli_fetch_array(result,resulttype);

Parameter	**Description**
result	Required. Specifies a result set identifier returned by mysqli_query(), mysqli_store_result() or mysqli_use_result()
resulttype	Optional. Specifies what type of array that should be produced. Can be one of the following values: MYSQLI_ASSOC MYSQLI_NUM MYSQLI_BOTH
Return Value:	Returns an array of strings that corresponds to the fetched row. NULL if there are no more rows in result-set

Example

Fetch a result row as a numeric array and as an associative array:

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
   if (mysqli_connect_errno())
         {
                  echo "Failed to connect to MySQL: " . mysqli_connect_error();
         }

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";
$result=mysqli_query($con,$sql);

// Numeric array
$row=mysqli_fetch_array($result,MYSQLI_NUM);
printf ("%s (%s)\n",$row[0],$row[1]);

// Associative array
$row=mysqli_fetch_array($result,MYSQLI_ASSOC);
printf ("%s (%s)\n",$row["Lastname"],$row["Age"]);

// Free result set
mysqli_free_result($result);

mysqli_close($con);
?>
```

mysqli_fetch_assoc() Function

Definition and Usage

The mysqli_fetch_assoc() function fetches a result row as an associative array.

Note: Fieldnames returned from this function are case-sensitive.

Syntax

mysqli_fetch_assoc(result);

Parameter **Description**

result	Required. Specifies a result set identifier returned by mysqli_query(), mysqli_store_result() or mysqli_use_result()

Return Value: Returns an associative array of strings representing the fetched row. NULL if there are no more rows in result-set

Example

Fetch a result row as an associative array:

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
  if (mysqli_connect_errno())
            {
           echo "Failed to connect to MySQL: " . mysqli_connect_error();
            }

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";
$result=mysqli_query($con,$sql);

// Associative array
$row=mysqli_fetch_assoc($result);
printf ("%s (%s)\n",$row["Lastname"],$row["Age"]);

// Free result set
mysqli_free_result($result);

mysqli_close($con);
?>
```

mysqli_fetch_row() Function

Definition and Usage

The mysqli_fetch_row() function fetches one row from a result-set and returns it as an enumerated array.

Syntax

mysqli_fetch_row(result);

Parameter	Description
result	Required. Specifies a result set identifier returned by mysqli_query(), mysqli_store_result() or mysqli_use_result()

Return Value: Returns an array of strings that corresponds to the fetched row. NULL if there are no more rows in result set

Example

Fetch rows from a result-set:

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
  if (mysqli_connect_errno())
          {
              echo "Failed to connect to MySQL: " . mysqli_connect_error();
           }

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";

  if ($result=mysqli_query($con,$sql))
          {
                  // Fetch one and one row
                          while ($row=mysqli_fetch_row($result))
                                  {
                                          printf ("%s (%s)\n",$row[0],$row[1]);
                                  }
             // Free result set
              mysqli_free_result($result);
        }

mysqli_close($con);
```

```
?>
```

mysqli_fetch_object() Function

Definition and Usage

The mysqli_fetch_object() function returns the current row of a result set, as an object.

Note: Fieldnames returned from this function are case-sensitive.

Syntax

mysqli_fetch_object(result,classname,params);

Parameter Description

result Required. Specifies a result set identifier returned by mysqli_query(), mysqli_store_result() or mysqli_use_result()

classname Optional. Specifies the name of the class to instantiate, set the properties of, and return

params Optional. Specifies an array of parameters to pass to the constructor for classname objects

Return Value: Returns an object with string properties for the fetched row. NULL if there are no more rows in the result set

Example

Return the current row of a result set, then print each field's value:

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
  if (mysqli_connect_errno())
```

```
        {
                echo "Failed to connect to MySQL: " . mysqli_connect_error();
        }

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";

  if ($result=mysqli_query($con,$sql))
        {
                while ($obj=mysqli_fetch_object($result))
                        {
                                printf("%s (%s)\n",$obj->Lastname,$obj->Age);
                        }
        // Free result set
        mysqli_free_result($result);
         }

mysqli_close($con);
?>
```

Create a MySQL Database Using MySQLi

The CREATE DATABASE statement is used to create a database in MySQL.

The following examples create a database named "myDB":

Example (MySQLi Procedural)- create database using php file

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername, $username, $password);
```

```
// Check connection
  if (!$conn)

        {
                die("Connection failed: " . mysqli_connect_error());
        }

// Create database
$sql = "CREATE DATABASE myDB";
   if (mysqli_query($conn, $sql))

        {
        echo "Database created successfully";
        }

   else

        {
                 echo "Error creating database: " . mysqli_error($conn);
        }

mysqli_close($conn);
?>
```

Example (MySQLi Object-oriented) – create database using php file

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
    if ($conn->connect_error)

        {
                die("Connection failed: " . $conn->connect_error);
        }

// Create database
$sql = "CREATE DATABASE myDB";
       if ($conn->query($sql) === TRUE)
```

```
        {
          echo "Database created successfully";
        }

       else

        {
                    echo "Error creating database: " . $conn->error;
        }
$conn->close();
?>
```

The following PDO example create a database named "myDBPDO":

Example (PDO)- create database using php file

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
try {
   $conn = new PDO("mysql:host=$servername", $username, $password);
   // set the PDO error mode to exception
   $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   $sql = "CREATE DATABASE myDBPDO";
   // use exec() because no results are returned
   $conn->exec($sql);
   echo "Database created successfully<br>";
   }
catch(PDOException $e)
   {
   echo $sql . "<br>" . $e->getMessage();
   }
$conn = null;
?>
```

Create MySQL Tables

A database table has its own unique name and consists of columns and rows.

Create a MySQL Table Using MySQLi

The CREATE TABLE statement is used to create a table in MySQL.

We will create a table named "**MyGuests**", with five **columns: "id", "firstname", "lastname", "email" and "reg_date":**

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)
```

Notes on the table above:

The data type specifies what type of data the column can hold. For a complete reference of all the available data types, go

After the data type, you can specify other optional attributes for each column:

NOT NULL - Each row must contain a value for that column, null values are not allowed
DEFAULT value - Set a default value that is added when no other value is passed
UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
PRIMARY KEY - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO_INCREMENT

Each table should have a primary key column (in this case: the "id" column). Its value must be unique for each record in the table.

The following examples shows how to create the table in PHP:

Example (MySQLi Procedural)-create table using php file

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
```

```
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
  if (!$conn)

        {
              die("Connection failed: " . mysqli_connect_error());
        }

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)";

     if (mysqli_query($conn, $sql))

        {
              echo "Table MyGuests created successfully";


        }

   else

        {
              echo "Error creating table: " . mysqli_error($conn);
        }

mysqli_close($conn);
?>
```

Example (MySQLi Object-oriented)-create table using php file

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
    if ($conn->connect_error)

        {
            die("Connection failed: " . $conn->connect_error);
        }

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
)";

    if ($conn->query($sql) === TRUE)

        {
            echo "Table MyGuests created successfully";
         }

         else

        {
            echo "Error creating table: " . $conn->error;
        }

$conn->close();
?>
```

Example (PDO)-create table using php file

```
<?php
$servername = "localhost";
$username = "username";
```

```
$password = "password";
$dbname = "myDBPDO";

try {
   $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
   // set the PDO error mode to exception
   $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

   // sql to create table
   $sql = "CREATE TABLE MyGuests (
   id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
   firstname VARCHAR(30) NOT NULL,
   lastname VARCHAR(30) NOT NULL,
   email VARCHAR(50),
   reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
   )";

   // use exec() because no results are returned
   $conn->exec($sql);
   echo "Table MyGuests created successfully";
   }
catch(PDOException $e)
   {
   echo $sql . "<br>" . $e->getMessage();
   }

$conn = null;
?>
```

Insert Data Into MySQL

After a database and a table have been created, we can start adding data in them.

Here are some syntax rules to follow:

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted

The INSERT INTO statement is used to add new records to a MySQL table:

INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)

In the previous chapter we created an empty table named "MyGuests" with five columns: "id", "firstname", "lastname", "email" and "reg_date". Now, let us fill the table with data.

If a column is AUTO_INCREMENT (like the "id" column) or TIMESTAMP with default update of current_timesamp (like the "reg_date" column), it is no need to be specified in the SQL query; MySQL will automatically add the value.

The following examples add a new record to the "MyGuests" table:

Form register.html/.php

Example

```
<html>
<head><title>register please</title>
        <h1>Register Please</h1>
</head>
  <body>
     <form name=f1 action='signup.php' method='get'>
          Enter First Name<input type=text name='firstname'><br>
          Enter Last Name<input type=text name='lastname'><br>
          Enter Email <input type=text name='email'><br>
          <input type=submit value='register'>
    </form>
  </body>
</html>
```

-signup.php

which is mentioned in value of attribute of 'action' in form tag of register.html file

(signup.php MySQLi Procedural)- Get variable from form and insert in database

```
<?php


$fn=$_GET['firstname'];
$ln=$_GET['lastname'];
```

```
$em=$_GET[‘email’];


$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES (‘$fn’, ‘$ln’, ‘$em’)";

if (mysqli_query($conn, $sql)) {
   echo "New record created successfully";
} else {
   echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

The following examples add a new record to the "MyGuests" table:

signup.php MySQLi Object-oriented- Get variable from form and insert in database

```
<?php

$fn=$_GET[‘firstname’];
$ln=$_GET[‘lastname’];
$em=$_GET[‘email’];


$servername = "localhost";
$username = "username";
```

```
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
    if ($conn->connect_error)

    {
        die("Connection failed: " . $conn->connect_error);
    }

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('$fn', '$ln', '$em')";

   if ($conn->query($sql) === TRUE)

       {
        echo "New record created successfully";


      }

  else

    {
        echo "Error: " . $sql . "<br>" . $conn->error;
     }

$conn->close();
?>
```

signup.php Example (PDO)- Get variable from form and insert in database

```
<?php


$fn=$_GET['firstname'];
$ln=$_GET['lastname'];
$em=$_GET['email'];
```

```
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
   $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
   // set the PDO error mode to exception
   $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
   $sql = “INSERT INTO MyGuests (firstname, lastname, email)
VALUES (‘$fn’, ‘$ln’, ‘$em’)”;
   // use exec() because no results are returned
   $conn->exec($sql);
   echo "New record created successfully";
   }
catch(PDOException $e)
   {
   echo $sql . "<br>" . $e->getMessage();
   }

$conn = null;
?>
```

Select Data From a MySQL Database

The SELECT statement is used to select data from one or more tables:

SELECT column_name(s) FROM table_name

or we can use the * character to select ALL columns from a table:

SELECT * FROM table_name

Select Data With MySQLi

The following example selects the id, firstname and lastname columns from the MyGuests table and displays it on the page:

Example (MySQLi Procedural)-fetching data from mysql to php and display it in html form

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
    if (!$conn)

{
                die("Connection failed: " . mysqli_connect_error());
        }

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);

     if (mysqli_num_rows($result) > 0)

        {
                // output data of each row
                while($row = mysqli_fetch_assoc($result))

                        {
                            echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
                        }
}

  else

        {
                echo "0 results";
         }

mysqli_close($conn);
?>
```

Example (MySQLi Object-oriented) **fetching data from mysql to php and display it in html form**

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
      if ($conn->connect_error)

        {
                die("Connection failed: " . $conn->connect_error);
        }

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

     if ($result->num_rows > 0)

        {
                echo "<table><tr><th>ID</th><th>Name</th></tr>";
                // output data of each row
                        while($row = $result->fetch_assoc())
```

```
              {
                        echo "<tr><td>".$row["id"]."</td><td>".$row["firstname"]." ".$row["lastname"]."</td></tr>";
               }
              echo "</table>";
         }

else

{
                echo "0 results";
        }
$conn->close();
?>
```

Select Data With PDO (+ Prepared Statements)- fetching data from mysql to php and display it in html form

The following example uses prepared statements.

It selects the id, firstname and lastname columns from the MyGuests table and displays it in an HTML table:

Example (PDO)- **fetching data from mysql to php and display it in html form**

```
<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th></tr>";

class TableRows extends RecursiveIteratorIterator {
    function __construct($it) {
        parent::__construct($it, self::LEAVES_ONLY);
    }

    function current() {
        return "<td style='width:150px;border:1px solid black;'>" . parent::current(). "</td>";
    }

    function beginChildren() {
        echo "<tr>";
    }

    function endChildren() {
        echo "</tr>" . "\n";
    }
}

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    $stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests");
    $stmt->execute();

    // set the resulting array to associative
    $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
    foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as $k=>$v) {
```

```
        echo $v;
    }
}
catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

Delete Data From a MySQL Table Using MySQLi

The DELETE statement is used to delete records from a table:

DELETE FROM table_name
WHERE some_column = some_value

Notice the WHERE clause in the DELETE syntax: The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30
3	Julie	Dooley	julie@example.com	2014-10-26 10:48:23

The following examples delete the record with id=3 in the "MyGuests" table:

Example (MySQLi Procedural) –delete data from mysql using php file

```
<?php


$id=$_GET[‘id’];
```

```
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
     if (!$conn)

              {
                        die("Connection failed: " . mysqli_connect_error());
           }

// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=$id";

         if (mysqli_query($conn, $sql))

           {
                    echo "Record deleted successfully";
           }

         else

          {
                    echo "Error deleting record: " . mysqli_error($conn);
          }

mysqli_close($conn);
?>
```

Example (MySQLi Object-oriented) –deleting data from mysql using php file

```
<?php
$id=$_GET[‘id’];

$servername = "localhost";
$username = "username";
```

```
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=$id";

if ($conn->query($sql) === TRUE) {
   echo "Record deleted successfully";
} else {
   echo "Error deleting record: " . $conn->error;
}

$conn->close();
?>
```

After the record is deleted, the table will look like this:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

Update Data in MySQL

The UPDATE statement is used to update existing records in a table:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Notice the WHERE clause in the UPDATE syntax: The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

The following examples update the record with id=2 in the "MyGuests" table:

Example (MySQLi Procedural) –Updating data in mysql using php file

```
<?php

$id=$_GET[‘id’];
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=$id";

if (mysqli_query($conn, $sql)) {
   echo "Record updated successfully";
} else {
   echo "Error updating record: " . mysqli_error($conn);
}
```

```
mysqli_close($conn);
?>
```

mysqli_num_rows() Function

Definition and Usage

The mysqli_num_rows() function returns the number of rows in a result set.

Syntax

mysqli_num_rows(result);

Parameter	Description
result	Required. Specifies a result set identifier returned by mysqli_query(), mysqli_store_result() or mysqli_use_result()

Return Value:	Returns the number of rows in the result set

Example

Return the number of rows in a result set:

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno())
  {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  }

$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";
```

```
if ($result=mysqli_query($con,$sql))
  {
  // Return the number of rows in result set
  $rowcount=mysqli_num_rows($result);
  printf("Result set has %d rows.\n",$rowcount);
  // Free result set
  mysqli_free_result($result);
  }

mysqli_close($con);
?>
```

mysqli_affected_rows() Function

Definition and Usage

The mysqli_affected_rows() function returns the number of affected rows in the previous SELECT, INSERT, UPDATE, REPLACE, or DELETE query.

Syntax

mysqli_affected_rows(connection)

Parameter	Description
connection	Required. Specifies the MySQL connection to use

Return Value:	The number of rows affected. -1 indicates that the query returned an error

Example

Return number of affected rows from different queries:

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
if (mysqli_connect_errno())
  {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  }

// Perform queries and print out affected rows
mysqli_query($con,"SELECT * FROM Persons");
echo "Affected rows: " . mysqli_affected_rows($con);

mysqli_query($con,"DELETE FROM Persons WHERE Age>32");
echo "Affected rows: " . mysqli_affected_rows($con);

mysqli_close($con);
?>
```

mysqli_error() Function

Definition and Usage

The mysqli_error() function returns the last error description for the most recent function call, if any.

Syntax

mysqli_error(connection);

Parameter	Description
connection	Required. Specifies the MySQL connection to use

Return Value: Returns a string with the error description. "" if no error occurred

Example

Return the last error description for the most recent function call, if any:

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno())
  {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  }

// Perform a query, check for error
if (!mysqli_query($con,"INSERT INTO Persons (FirstName) VALUES ('Glenn')"))
  {
  echo("Error description: " . mysqli_error($con));
  }

mysqli_close($con);
?>
```

mysqli_errno() Function

Definition and Usage

The mysqli_errno() function returns the last error code for the most recent function call, if any.

Syntax

mysqli_errno(connection);

Parameter **Description**

connection	Required. Specifies the MySQL connection to use

Return Value: Returns an error code value. Zero if no error occurred

Example

Return the last error code for the most recent function call, if any:

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno())
  {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  }

// Perform a query, check for error
if (!mysqli_query($con,"INSERT INTO Persons (FirstName) VALUES ('Glenn')"))
  {
  echo("Errorcode: " . mysqli_errno($con));
  }

mysqli_close($con);
?>
```

Get ID of Last Inserted Record

mysqli_insert_id Function

If we perform an INSERT or UPDATE on a table with an AUTO_INCREMENT field, we can get the ID of the last inserted/updated record immediately.

The following examples are equal to the examples from the previous page (PHP Insert Data Into MySQL), except that we have added one single line of code to retrieve the ID of the last inserted record. We also echo the last inserted ID:

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql)) {
   $last_id = mysqli_insert_id($conn);
   echo "New record created successfully. Last inserted ID is: " . $last_id;
} else {
   echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
```

```
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
    $last_id = $conn->insert_id;
    echo "New record created successfully. Last inserted ID is: " . $last_id;
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

Basics of OOP in PHP

Object Oriented Concepts

Before we go in detail, lets define important terms related to Object Oriented Programming.

Class − This is a programmer-defined data type, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.

Object − An individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.

Member Variable − These are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called attribute of the object once an object is created.

Member function − These are the function defined inside a class and are used to access object data.

Inheritance − When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.

Parent class − A class that is inherited from by another class. This is also called a base class or super class.

Child Class − A class that inherits from another class. This is also called a subclass or derived class.

Polymorphism − This is an object oriented concept where same function can be used for different purposes. For example function name will remain same but it take different number of arguments and can do different task.

Overloading − a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly functions can also be overloaded with different implementation.

Data Abstraction − Any representation of data in which the implementation details are hidden (abstracted).

Encapsulation − refers to a concept where we encapsulate all the data and member functions together to form an object.

Constructor − refers to a special type of function which will be called automatically whenever there is an object formation from a class.

Destructor − refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope

As of PHP 5, PHP includes a complete object model. Some of its features are*: visibility*, *abstract* and final classes and methods, additional *magic methods*, *interfaces*, *cloning* and *typehinting*.

Magic Methods

The function names *__construct(), __destruct(), __call(), __callStatic(), __get(), __set(), __isset(), __unset(), __sleep(), __wakeup(), __toString(),__invoke(), __set_state(), __clone() and __debugInfo()* are magical in PHP classes. You cannot have functions with these names in any of your classes unless you want the magic functionality associated with them.

Defining PHP Classes

The general form for defining a new class in PHP is as follows −

```
<?php
  class phpClass {
    var $var1;
    var $var2 = "constant string";
```

```
    function myfunc ($arg1, $arg2) {
      [..]
    }
    [..]
  }
?>
```

Here is the description of each line −

The special form class, followed by the name of the class that you want to define.

A set of braces enclosing any number of variable declarations and function definitions.

Variable declarations start with the special form var, which is followed by a conventional $ variable name; they may also have an initial assignment to a constant value.

Function definitions look much like standalone PHP functions but are local to the class and will be used to set and access object data.

Example

Here is an example which defines a class of Books type −

```
<?php
  class Books {
    /* Member variables */
    var $price;
```

```
    var $title;

    /* Member functions */
    function setPrice($par){
      $this->price = $par;
    }

    function getPrice(){
      echo $this->price ."<br/>";
    }

    function setTitle($par){
      $this->title = $par;
    }

    function getTitle(){
      echo $this->title ." <br/>";
    }
  }
?>
```

The variable $this is a special variable and it refers to the same object ie. itself.

Creating Objects in PHP

Once you defined your class, then you can create as many objects as you like of that class type. Following is an example of how to create object using new operator.

```
$physics = new Books;

$maths = new Books;

$chemistry = new Books;
```

Here we have created three objects and these objects are independent of each other and they will have their existence separately. Next we will see how to access member function and process member variables.

Calling Member Functions

After creating your objects, you will be able to call member functions related to that object. One member function will be able to process member variable of related object only.

Following example shows how to set title and prices for the three books by calling member functions.

```
$physics->setTitle( "Physics for High School" );

$chemistry->setTitle( "Advanced Chemistry" );

$maths->setTitle( "Algebra" );
```

```
$physics->setPrice( 10 );

$chemistry->setPrice( 15 );

$maths->setPrice( 7 );

Now you call another member functions to get the values set by in above example −

$physics->getTitle();

$chemistry->getTitle();

$maths->getTitle();

$physics->getPrice();

$chemistry->getPrice();

$maths->getPrice();
```

This will produce the following result −

Physics for High School

Advanced Chemistry

Algebra

10

15

7

Constructor Functions

Constructor Functions are special type of functions which are called automatically whenever an object is created. So we take full advantage of this behaviour, by initializing many things through constructor functions.

PHP provides a special function called __construct() to define a constructor. You can pass as many as arguments you like into the constructor function.

Following example will create one constructor for Books class and it will initialize price and title for the book at the time of object creation.

```
function __construct( $par1, $par2 ) {
  $this->title = $par1;
  $this->price = $par2;
}
```

Now we don't need to call set function separately to set price and title. We can initialize these two member variables at the time of object creation only. Check following example below −

```
$physics = new Books( "Physics for High School", 10 );
$maths = new Books ( "Advanced Chemistry", 15 );
```

```
$chemistry = new Books ("Algebra", 7 );


/* Get those set values */

$physics->getTitle();

$chemistry->getTitle();

$maths->getTitle();


$physics->getPrice();

$chemistry->getPrice();

$maths->getPrice();
```

This will produce the following result −

Physics for High School

Advanced Chemistry

Algebra

10

15

7

Destructor

Like a constructor function you can define a destructor function using function __destruct(). You can release all the resources with-in a destructor.

Inheritance

PHP class definitions can optionally inherit from a parent class definition by using the extends clause. The syntax is as follows −

```
class Child extends Parent {
```

```
   <definition body>
}
```

The effect of inheritance is that the child class (or subclass or derived class) has the following characteristics −

Automatically has all the member variable declarations of the parent class.

Automatically has all the same member functions as the parent, which (by default) will work the same way as those functions do in the parent.

Following example inherit Books class and adds more functionality based on the requirement.

```
class Novel extends Books {
   var $publisher;

   function setPublisher($par){
      $this->publisher = $par;
   }

   function getPublisher(){
      echo $this->publisher. "<br />";
   }
}
```

Now apart from inherited functions, class Novel keeps two additional member functions.

Function Overriding

Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.

In the following example getPrice and getTitle functions are overridden to return some values.

```
function getPrice() {
  echo $this->price . "<br/>";
  return $this->price;
}

function getTitle(){
  echo $this->title . "<br/>";
  return $this->title;
}
```

Public Members

Unless you specify otherwise, properties and methods of a class are public. That is to say, they may be accessed in three possible situations −

From outside the class in which it is declared

From within the class in which it is declared

From within another class that implements the class in which it is declared

Till now we have seen all members as public members. If you wish to limit the accessibility of the members of a class then you define class members as private or protected.

Private members

By designating a member private, you limit its accessibility to the class in which it is declared. The private member cannot be referred to from classes that inherit the class in which it is declared and cannot be accessed from outside the class.

A class member can be made private by using private keyword infront of the member.

```
class MyClass {
  private $car = "skoda";
  $driver = "SRK";

  function __construct($par) {
    // Statements here run every time
    // an instance of the class
    // is created.
  }

  function myPublicFunction() {
    return("I'm visible!");
  }

  private function myPrivateFunction() {
    return("I'm  not visible outside!");
  }
}
```

When MyClass class is inherited by another class using extends, myPublicFunction() will be visible, as will $driver. The extending class will not have any awareness of or access to myPrivateFunction and $car, because they are declared private.

Protected members

A protected property or method is accessible in the class in which it is declared, as well as in classes that extend that class. Protected members are not available outside of those two kinds of classes. A class member can be made protected by using protected keyword in front of the member.

Here is different version of MyClass −

```
class MyClass {
   protected $car = "skoda";
   $driver = "SRK";

   function __construct($par) {
      // Statements here run every time
      // an instance of the class
      // is created.
   }

   function myPublicFunction() {
      return("I'm visible!");
   }

   protected function myPrivateFunction() {
      return("I'm  visible in child class!");
   }
}
```

Interfaces

Interfaces are defined to provide a common function names to the implementers. Different implementors can implement those interfaces according to their requirements. You can say, interfaces are skeletons which are implemented by developers.

As of PHP5, it is possible to define an interface, like this −

```
interface Mail {
   public function sendMail();
}
```

Then, if another class implemented that interface, like this −

```
class Report implements Mail {
   // sendMail() Definition goes here
}
```

Constants

A constant is somewhat like a variable, in that it holds a value, but is really more like a function because a constant is immutable. Once you declare a constant, it does not change.

Declaring one constant is easy, as is done in this version of MyClass −

```
class MyClass {
   const requiredMargin = 1.7;
```

```
  function __construct($incomingValue) {
    // Statements here run every time
    // an instance of the class
    // is created.
  }
}
```

In this class, requiredMargin is a constant. It is declared with the keyword const, and under no circumstances can it be changed to anything other than 1.7. Note that the constant's name does not have a leading $, as variable names do.

Abstract Classes

An abstract class is one that cannot be instantiated, only inherited. You declare an abstract class with the keyword abstract, like this −

When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child; additionally, these methods must be defined with the same visibility.

```
abstract class MyAbstractClass {
  abstract function myAbstractFunction() {
  }
}
```

Note that function definitions inside an abstract class must also be preceded by the keyword abstract. It is not legal to have abstract function definitions inside a non-abstract class.

Static Keyword

Declaring class members or methods as static makes them accessible without needing an instantiation of the class. A member declared as static can not be accessed with an instantiated class object (though a static method can).

Try out following example −

```
<?php
  class Foo {
    public static $my_static = 'foo';
      public function staticValue() {
      return self::$my_static;
    }
  }
  print Foo::$my_static . "\n";
  $foo = new Foo();
    print $foo->staticValue() . "\n";
?>
```

Final Keyword

PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with final. If the class itself is being defined final then it cannot be extended.

Following example results in Fatal error: Cannot override final method BaseClass::moreTesting()

```
<?php
  class BaseClass {
    public function test() {
      echo "BaseClass::test() called<br>";
    }
      final public function moreTesting() {
      echo "BaseClass::moreTesting() called<br>";
    }
  }
    class ChildClass extends BaseClass {
```

```
    public function moreTesting() {
      echo "ChildClass::moreTesting() called<br>";
    }
  }
?>
```

Calling parent constructors

Instead of writing an entirely new constructor for the subclass, let's write it by calling the parent's constructor explicitly and then doing whatever is necessary in addition for instantiation of the subclass. Here's a simple example −

```
class Name {
  var $_firstName;
  var $_lastName;
```

```
    function Name($first_name, $last_name) {
    $this->_firstName = $first_name;
    $this->_lastName = $last_name;
  }


  function toString() {
    return($this->_lastName .", " .$this->_firstName);
  }
}
class NameSub1 extends Name {
  var $_middleInitial;
   function NameSub1($first_name, $middle_initial, $last_name) {
    Name::Name($first_name, $last_name);
```

```
    $this->_middleInitial = $middle_initial;
```

```
  }
    function toString() {
    return(Name::toString() . " " . $this->_middleInitial);
  }
}
```

In this example, we have a parent class (Name), which has a two-argument constructor, and a subclass (NameSub1), which has a three-argument constructor. The constructor of NameSub1 functions by calling its parent constructor explicitly using the :: syntax (passing two of its arguments along) and then setting an additional field. Similarly, NameSub1 defines its non constructor toString() function in terms of the parent function that it overrides.

A constructor can be defined with the same name as the name of a class. It is defined in above example

R eferences:

1. W3schools.com
2. Javapoint.com
3. Php.net
4. Tutorialspoint.com
5. Wordpress.org
   Thankful to so many online resources