PHP

What is OOP?

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-oriented programming has several advantages over procedural programming:

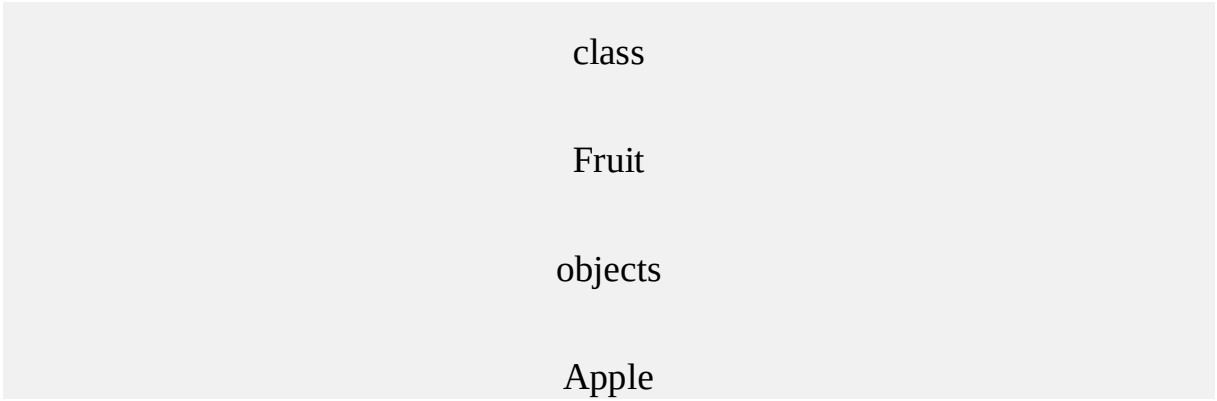
- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Tip: The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

PHP - What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

Look at the following illustration to see the difference between class and objects:



Banana

Mango

Another example:

class

Car

objects

Volvo

Audi

Toyota

So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

PHP OOP - Classes and Objects

A class is a template for objects, and an object is an instance of class.

OOP Case

Let's assume we have a class named Fruit. A Fruit can have properties like name, color, weight, etc. We can define variables like $name, $color, and $weight to hold the values of these properties.

When the individual objects (apple, banana, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

Define a Class

A class is defined by using the class keyword, followed by the name of the class and a pair of curly braces ({}). All its properties and methods go inside the braces:

Syntax

```
<?php
class Fruit {
  // code goes here...
}
?>
```

Below we declare a class named Fruit consisting of two properties ($name and $color) and two methods set_name() and get_name() for setting and getting the $name property:

Example

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

Define Objects

Classes are nothing without objects! We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values.

Objects of a class is created using the new keyword.

In the example below, $apple and $banana are instances of the class Fruit:

Example

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

In the example below, we add two more methods to class Fruit, for setting and getting the $color property:

Example

```
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
  function set_color($color) {
    $this->color = $color;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " . $apple->get_color();
?>
```

PHP - The $this Keyword

The $this keyword refers to the current object, and is only available inside methods.

Look at the following example:

Example

```
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
?>
```

So, where can we change the value of the $name property? There are two ways:

1. Inside the class (by adding a set_name() method and use $this):

Example

```
<?php
class Fruit {
  public $name;
  function set_name($name) {
    $this->name = $name;
  }
}
$apple = new Fruit();
$apple->set_name("Apple");
?>
```

2. Outside the class (by directly changing the property value):

Example

```
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
```

```
$apple->name = "Apple";
?>
```

PHP - instanceof

You can use the instanceof keyword to check if an object belongs to a specific class:

Example

```
<?php
$apple = new Fruit();
var_dump($apple instanceof Fruit);
?>
```

Object Oriented Programming in PHP

We can imagine our universe made of different objects like sun, earth, moon etc. Similarly we can imagine our car made of different objects like wheel, steering, gear etc. Same way there is object oriented programming concepts which assume everything as an object and implement a software using different objects.

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.

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.

PHP extends Keyword

Example

Inherit from a class:

```
<?php
class MyClass {
  public function hello() {
    echo "Hello World!";
  }
}

class AnotherClass extends MyClass {
}

$obj = new AnotherClass();
$obj->hello();
```

Definition and Usage

The ***extends*** keyword is used to derive a class from another class. This is called inheritance. A derived class has all of the public and protected properties of the class that it is derived from.

PHP - Access Modifiers

Properties and methods can have access modifiers which control where they can be accessed.

There are three access modifiers:

- public - the property or method can be accessed from everywhere. This is default
- protected - the property or method can be accessed within the class and by classes derived from that class
- private - the property or method can ONLY be accessed within the class

In the following example we have added three different access modifiers to the three properties. Here, if you try to set the name property it will work fine (because the name property is public). However, if you try to set the color or weight property it will result in a fatal error (because the color and weight property are protected and private):

Example

```
<?php
class Fruit {
  public $name;
  protected $color;
  private $weight;
}

$mango = new Fruit();
```

```
$mango->name = 'Mango'; // OK
$mango->color = 'Yellow'; // ERROR
$mango->weight = '300'; // ERROR
?>
```

In the next example we have added access modifiers to two methods. Here, if you try to call the set_color() or the set_weight() function it will result in a fatal error (because the two functions are considered protected and private), even if all the properties are public:

Example

```
<?php
class Fruit {
  public $name;
  public $color;
  public $weight;

  function set_name($n) {  // a public function (default)
    $this->name = $n;
  }
  protected function set_color($n) { // a protected function
    $this->color = $n;
  }
  private function set_weight($n) { // a private function
    $this->weight = $n;
  }
}

$mango = new Fruit();
$mango->set_name('Mango'); // OK
$mango->set_color('Yellow'); // ERROR
$mango->set_weight('300'); // ERROR
?>
```

PHP Directory Introduction

PHP Directory Functions

The directory functions allow you to retrieve information about directories and their contents.

Installation

The PHP directory functions are part of the PHP core. No installation is required to use these functions.

PHP Directory Functions

Function	Description
chdir()	Changes the current directory
chroot()	Changes the root directory
closedir()	Closes a directory handle
dir()	Returns an instance of the Directory class
getcwd()	Returns the current working directory
opendir()	Opens a directory handle

readdir()	Returns an entry from a directory handle
rewinddir()	Resets a directory handle
scandir()	Returns an array of files and directories of a specified directory

PHP File Open/Read/Close

In this chapter how to open, read, and close a file on the server.

PHP Open File - fopen()

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.

We will use the text file, "webdictionary.txt", during the lessons:

AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language

The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fread($myfile,filesize("webdictionary.txt"));
fclose($myfile);
?>
```

Tip: The fread() and the fclose() functions will be explained below.

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

Modes	Description
r	**Open a file for read only**. File pointer starts at the beginning of the file
w	**Open a file for write only**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	**Open a file for write only**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	**Creates a new file for write only**. Returns FALSE and an error if file already exists
r+	**Open a file for read/write**. File pointer starts at the beginning of the file

w+	**Open a file for read/write**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	**Open a file for read/write**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	**Creates a new file for read/write**. Returns FALSE and an error if file already exists

---

PHP Read File - fread()

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

fread($myfile,filesize("webdictionary.txt"));

PHP Close File - fclose()

The fclose() function is used to close an open file.

It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

```
<?php
$myfile = fopen("webdictionary.txt", "r");
// some code to be executed....
fclose($myfile);
?>
```

PHP Read Single Line - fgets()

The fgets() function is used to read a single line from a file.

The example below outputs the first line of the "webdictionary.txt" file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fgets($myfile);
fclose($myfile);
?>
```

PHP Check End-Of-File - feof()

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

The feof() function is useful for looping through data of unknown length.

The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
```

```
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

PHP Read Single Character - fgetc()

The fgetc() function is used to read a single character from a file.

The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one character until end-of-file
while(!feof($myfile)) {
  echo fgetc($myfile);
}
fclose($myfile);
?>
```

Note: After a call to the fgetc() function, the file pointer moves to the next character.

Using remote files

As long as allow_url_fopen is enabled in *php.ini*, you can use HTTP and FTP URLs with most of the functions that take a filename as a parameter. In addition, URLs can be used with the include, include_once, require and require_once statements (since PHP 5.2.0, allow_url_include must be enabled for these). See Supported Protocols and Wrappers for more information about the protocols supported by PHP.

For example, you can use this to open a file on a remote web server, parse the output for the data you want, and then use that data in a database query, or simply to output it in a style matching the rest of your website.

Example #1 Getting the title of a remote page

```
<?php
$file = fopen ("http://www.example.com/", "r");
if (!$file) {
    echo "<p>Unable to open remote file.\n";
    exit;
}
while (!feof ($file)) {
    $line = fgets ($file, 1024);
    /* This only works if the title and its tags are on one line */
    if (preg_match ("@\<title\>(.*)\</title\>@i", $line, $out)) {
        $title = $out[1];
        break;
    }
}
fclose($file);
?>
```

You can also write to files on an FTP server (provided that you have connected as a user with the correct access rights). You can only create new files using this method; if you try to overwrite a file that already exists, the fopen() call will fail.

To connect as a user other than 'anonymous', you need to specify the username (and possibly password) within the URL, such as 'ftp://user:password@ftp.example.com/path/to/file'. (You can use the same sort of syntax to access files via HTTP when they require Basic authentication.)

Example #2 Storing data on a remote server

```
<?php
$file = fopen ("ftp://ftp.example.com/incoming/outputfile", "w");
if (!$file) {
    echo "<p>Unable to open remote file for writing.\n";
    exit;
```

```
}
/* Write the data here. */
fwrite ($file, $_SERVER['HTTP_USER_AGENT'] . "\n");
fclose ($file);
?>
```

Note:
You might get the idea from the example above that you can use this technique to write to a remote log file. Unfortunately that would not work because the fopen() call will fail if the remote file already exists. To do distributed logging like that, you should take a look at syslog().

Runtime Configuration

The behavior of the filesystem functions is affected by settings in php.ini.

Name	Default	Description	Changeable
allow_url_fopen	"1"	Allows fopen()-type functions to work with URLs	PHP_INI_SYSTEM
allow_url_include	"0"	(available since PHP 5.2)	PHP_INI_SYSTEM
user_agent	NULL	Defines the user agent for PHP to send (available since PHP 4.3)	PHP_INI_ALL

default_socket_timeout	"60"	Sets the default timeout, in seconds, for socket based streams (available since PHP 4.3)	PHP_INI_ALL
from	""	Defines the email address to be used on unauthenticated FTP connections and in the From header for HTTP connections when using ftp and http wrappers	PHP_INI_ALL
auto_detect_line_endings	"0"	When set to "1", PHP will examine the data read by fgets() and file() to see if it is using Unix, MS-Dos or Mac line-ending characters	PHP_INI_ALL

		(available since PHP 4.3)	
sys_temp_dir	""	(available since PHP 5.5)	PHP_INI_SYSTEM

PHP Filesystem Functions

Function	Description
basename()	Returns the filename component of a path
chgrp()	Changes the file group
chmod()	Changes the file mode
chown()	Changes the file owner
clearstatcache()	Clears the file status cache
copy()	Copies a file

delete()	See unlink()
dirname()	Returns the directory name component of a path
disk_free_space()	Returns the free space of a filesystem or disk
disk_total_space()	Returns the total size of a filesystem or disk
diskfreespace()	Alias of disk_free_space()
fclose()	Closes an open file
feof()	Checks if the "end-of-file" (EOF) has been reached for an open file
fflush()	Flushes buffered output to an open file
fgetc()	Returns a single character from an open file
fgetcsv()	Returns a line from an open CSV file
fgets()	Returns a line from an open file

fgetss()	Deprecated from PHP 7.3. Returns a line from an open file - stripped from HTML and PHP tags
file()	Reads a file into an array
file_exists()	Checks whether or not a file or directory exists
file_get_contents()	Reads a file into a string
file_put_contents()	Writes data to a file
fileatime()	Returns the last access time of a file
filectime()	Returns the last change time of a file
filegroup()	Returns the group ID of a file
fileinode()	Returns the inode number of a file
filemtime()	Returns the last modification time of a file
fileowner()	Returns the user ID (owner) of a file

fileperms()	Returns the file's permissions
filesize()	Returns the file size
filetype()	Returns the file type
flock()	Locks or releases a file
fnmatch()	Matches a filename or string against a specified pattern
fopen()	Opens a file or URL
fpassthru()	Reads from the current position in a file - until EOF, and writes the result to the output buffer
fputcsv()	Formats a line as CSV and writes it to an open file
fputs()	Alias of fwrite()
fread()	Reads from an open file (binary-safe)

fscanf()	Parses input from an open file according to a specified format
fseek()	Seeks in an open file
fstat()	Returns information about an open file
ftell()	Returns the current position in an open file
ftruncate()	Truncates an open file to a specified length
fwrite()	Writes to an open file (binary-safe)
glob()	Returns an array of filenames / directories matching a specified pattern
is_dir()	Checks whether a file is a directory
is_executable()	Checks whether a file is executable
is_file()	Checks whether a file is a regular file

is_link()	Checks whether a file is a link
is_readable()	Checks whether a file is readable
is_uploaded_file()	Checks whether a file was uploaded via HTTP POST
is_writable()	Checks whether a file is writable
is_writeable()	Alias of is_writable()
lchgrp()	Changes the group ownership of a symbolic link
lchown()	Changes the user ownership of a symbolic link
link()	Creates a hard link
linkinfo()	Returns information about a hard link
lstat()	Returns information about a file or symbolic link
mkdir()	Creates a directory

move_uploaded_file()	Moves an uploaded file to a new location
parse_ini_file()	Parses a configuration file
parse_ini_string()	Parses a configuration string
pathinfo()	Returns information about a file path
pclose()	Closes a pipe opened by popen()
popen()	Opens a pipe
readfile()	Reads a file and writes it to the output buffer
readlink()	Returns the target of a symbolic link
realpath()	Returns the absolute pathname
realpath_cache_get()	Returns realpath cache entries
realpath_cache_size()	Returns realpath cache size

rename()	Renames a file or directory
rewind()	Rewinds a file pointer
rmdir()	Removes an empty directory
set_file_buffer()	Alias of stream_set_write_buffer(). Sets the buffer size for write operations on the given file
stat()	Returns information about a file
symlink()	Creates a symbolic link
tempnam()	Creates a unique temporary file
tmpfile()	Creates a unique temporary file
touch()	Sets access and modification time of a file
umask()	Changes file permissions for files
unlink()	Deletes a file

PHP File Create/Write

In this chapter teach ,how to create and write to a file on the server.

PHP Create File - fopen()

The fopen() function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.

If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:

Example

```
$myfile = fopen("testfile.txt", "w")
```

PHP File Permissions

If you are having errors when trying to get this code to run, check that you have granted your PHP file access to write information to the hard drive.

PHP Write to File - fwrite()

The fwrite() function is used to write to a file.

The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.

The example below writes a couple of names into a new file called "newfile.txt":

Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "John Doe\n";
fwrite($myfile, $txt);
$txt = "Jane Doe\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

Notice that we wrote to the file "newfile.txt" twice. Each time we wrote to the file we sent the string $txt that first contained "John Doe" and second contained "Jane Doe". After we finished writing, we closed the file using the fclose() function.

If we open the "newfile.txt" file it would look like this:

John Doe
Jane Doe

PHP Overwriting

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

In the example below we open our existing file "newfile.txt", and write some new data into it:

Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "Mickey Mouse\n";
fwrite($myfile, $txt);
$txt = "Minnie Mouse\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

If we now open the "newfile.txt" file, both John and Jane have vanished, and only the data we just wrote is present:

Mickey Mouse
Minnie Mouse

PHP Directory Functions

PHP Directory Introduction

The directory functions allow you to retrieve information about directories and their contents.

Installation

The PHP directory functions are part of the PHP core. No installation is required to use these functions.

PHP Directory Functions

Function	Description

chdir()	Changes the current directory
chroot()	Changes the root directory
closedir()	Closes a directory handle
dir()	Returns an instance of the Directory class
getcwd()	Returns the current working directory
opendir()	Opens a directory handle
readdir()	Returns an entry from a directory handle
rewinddir()	Resets a directory handle
scandir()	Returns an array of files and directories of a specified directory

PHP Directory operations

Some days before we have seen set of basic PHP file functions to perform file open, read, write or append operations. Similarly, PHP includes set of directory

functions to deal with the operations, like, listing directory contents, and create/ open/ delete specified directory and etc. These basic functions are listed below.

- *mkdir()*: To make new directory.
- *opendir()*: To open directory.
- *readdir()*: To read from a directory after opening it.
- closedir(): To close directory with resource-id returned while opening.
- *rmdir()*: To remove directory.

In this article, we have to discuss each of the above basic directory functions in PHP with their corresponding usage of these functions, possible parameter to be passed if any, with suitable examples.

Creating New Directory

For creating a new directory using PHP programming script, *mkdir()* function used, and, the usage of this function is as follows.

```
mkdir($directory_path,$mode,$recursive_flag,$context);
```

This function accepts four arguments as specified. Among them, the first argument is mandatory, whereas, the remaining set of arguments are optional.

- *$directory_path*: By specifying either relative and absolute path, a new directory will be created in such location if any, otherwise, will return an error indicating that there are no such locations.
- $mode: The mode parameter accepts octal values in which the accessibility of the newly created directory depends on.
- $recursive: This parameter is a flag and having values either *true* or *false*, that allow or refuse to create nested directories further.
- $context: As similar as we have with PHP *unlink()* having a stream for specifying protocols and etc.

This function will return boolean data, that is, *true* on successful execution, *false* otherwise.

Listing Directory Content in PHP

For listing the contents of a directory, we require two of the above-listed PHP directory functions, these are, *opendir()* and *readdir()*. There are two steps in directory listing using PHP program.

- Step 1: Opening directory.
- Step 2: Reading content to be listed one by one using PHP loop.

Step 1: Opening Directory Link

As its name, *opendir*() function is used to perform this step. And, it has two arguments, one is compulsory for specifying the path of the directory, and the other is optional, expecting stream context if any. The syntax will be,

```
opendir($directory_path,$context);
```

Unlike PHP *mkdir()* returning boolean value, this function will return resource data as like as fopen(), mysql_query() and etc. After receiving the resource identifier return by this function, then only we can progress with the subsequent steps to read, rewind or to close required directory with the reference of this resource id. Otherwise, PHP error will occur for indicating the user, that the resource id is not valid.

Step 2: Reading Directory Content

For performing this step, we need to call *readdir()* function recursively until the directory handle reaches the end of the directory. For that, we need to specify the resource-id returned while invoking *opendir()*, indicated as directory handle. PHP *readdir()* will return string data on each iteration of the loop, and this string will be the name of each item stored in the directory with its corresponding extension. For example,

```
$directory_handle = opendir($directory_path);

while($directory_item = readdir($directory_handle)) {

echo $directory_item . "<br>";

}
```

And, thereby executing the above code sample, we can list the content of a directory as expected.

Closing Directory Link

Once the directory link is opened to perform set of dependent operations like reading directory content, we need to close this link after completing the related functionalities required. For example,

```
$directory_handle = opendir($directory_path);

...

...

closedir($directory_handle);
```

Removing Directory

We have seen in the previous article about how to delete a file from a directory using PHP unlink(). Similarly, for removing the entire directory, PHP provides a function named as *rmdir()* which accepts the same set of arguments, as like as *mkdir()*. These are, the *$directory_path* and *$context*(Optional) as stated below.

```
rmdir($directory_path,$mode,$recursive_flag,$context);
```

But, this function will remove the directory, if and only if it is empty. For removing the non-empty directory, we need to create a user define a function that recursively calls *unlink()* function to remove each file stored in the directory to be deleted.

Example: PHP Directory Functions:

The following PHP program deals with the set of directory functions, that is, for creating a new directory, open and read for listing directory content and closing directory as follows.

```
<?php

mkdir("php_directory_functions_manual",0777);

/*Creating files into php_directory_functions_manual*/

$file_pointer1 = fopen("php_directory_functions_manual/mkdir.txt","x");
```

```
$file_pointer2 = fopen("php_directory_functions_manual/rmdir.txt","x");

fclose($file_pointer1);

fclose($file_pointer2);

$directory_handle = opendir("php_directory_functions_manual");

while($directory_item = readdir($directory_handle)) {

echo $directory_item . "<br>";

}

closedir($directory_handle);

?>
```

This program will return the list of created files by using *fopen()* function as shown below.

```
.

..

mkdir.txt

rmdir.txt
```

And then, let us have an another example, for looking into how a directory can be removed.

```
<?php

$directory_handle = opendir("php_directory_functions_manual");

while($directory_item = readdir($directory_handle)) {

@unlink("php_directory_functions_manual/".$directory_item);

}

closedir($directory_handle);
```

```
rmdir("php_directory_functions_manual");

?>
```

On iterating with a loop for reading each item stored into the directory, the *unlink()* function is invoked to wipeout the directory before attempting to delete it. And then, *rmdir()* is used by referring with the name of the directory, to remove it.

Note:

- If we invoked *opendir()* only once in a PHP program, its dependent functions, like, *readdir()*, *closedir()* and etc., will execute successfully, without specifying the directory handle resource, since, it refers recently returned resource by default. Rather, if we have multiple resource data referring various directory links, then we need to specify the directory handle appropriately.
- For getting the successful execution of PHP *unlink()* and *rmdir()*, all required permissions should be provided with the entries on which these functions are applied.