**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.

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.

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 {}

?>

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:

<?php

class Fruit {

public $name;

public $color;

function set\_name($name) {

$this->name = $name;

}

function get\_name() {

return $this->name;

}

}

?>

In a class, variables are called properties and functions are called methods!

**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 are created using the new keyword.

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

Example

<?php

class Fruit {

public $name;

public $color;

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();

?>

**PHP - The \_\_construct Function**

A constructor allows you to initialize an object's properties upon creation of the object.

If you create a \_\_construct() function, PHP will automatically call this function when you create an object from a class.

Notice that the construct function starts with two underscores (\_\_)!

We see in the example below, that using a constructor saves us from calling the set\_name() method which reduces the amount of code:

Example

<?php

class Fruit {

public $name;

public $color;

function \_\_construct($name) {

$this->name = $name;

}

function get\_name() {

return $this->name;

}

}

$apple = new Fruit("Apple");

echo $apple->get\_name();

?>

**PHP - The \_\_destruct Function**

A destructor is called when the object is destructed or the script is stopped or exited.

If you create a \_\_destruct() function, PHP will automatically call this function at the end of the script.

Notice that the destruct function starts with two underscores (\_\_)!

The example below has a \_\_construct() function that is automatically called when you create an object from a class, and a \_\_destruct() function that is automatically called at the end of the script:

Example

<?php

class Fruit {

public $name;

public $color;

function \_\_construct($name) {

$this->name = $name;

}

function \_\_destruct() {

echo "The fruit is {$this->name}.";

}

}

$apple = new Fruit("Apple");

?>

**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

<?php

class Fruit {

public $name;

protected $color;

private $weight;

}

$mango = new Fruit();

$mango->name = 'Mango'; // OK

$mango->color = 'Yellow'; // ERROR

$mango->weight = '300'; // ERROR

?>

**PHP - What are Abstract Classes and Methods?**

Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks.An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code.An abstract class or method is defined with the abstract keyword:

Syntax

<?php

abstract class ParentClass {

abstract public function someMethod1();

abstract public function someMethod2($name, $color);

abstract public function someMethod3() : string;

}

?>

When inheriting from an abstract class, the child class method must be defined with the same name, and the same or a less restricted access modifier. So, if the abstract method is defined as protected, the child class method must be defined as either protected or public, but not private. Also, the type and number of required arguments must be the same. However, the child classes may have optional arguments in addition.

So, when a child class is inherited from an abstract class, we have the following rules:

* The child class method must be defined with the same name and it re declares the parent abstract method
* The child class method must be defined with the same or a less restricted access modifier
* The number of required arguments must be the same. However, the child class may have optional arguments in addition

Example

<?php

abstract class Car {

public $name;

public function \_\_construct($name) {

$this->name = $name;

}

abstract public function intro() ;}

// Child class

class Audi extends Car {

public function intro() {

return "Choose German quality! I'm an $this->name!";

}

}

$audi = new audi("Audi");

echo $audi->intro();

?>

**PHP - What is Inheritance?**

Inheritance in OOP = When a class derives from another class.

The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods.

An inherited class is defined by using the extends keyword.

Let's look at an example:

Example

<?php

class Fruit {

public $name;

public $color;

public function \_\_construct($name, $color) {

$this->name = $name;

$this->color = $color;

}

public function intro() {

echo "The fruit is {$this->name} and the color is {$this->color}.";

}

}

// Strawberry is inherited from Fruit

class Strawberry extends Fruit {

public function message() {

echo "Am I a fruit or a berry? ";

}

}

$strawberry = new Strawberry("Strawberry", "red");

$strawberry->message();

$strawberry->intro();

?>

**Static Methods**

Static methods can be called directly - without creating an instance of the class first.

Static methods are declared with the static keyword:

**Syntax**

<?php

class ClassName {

public static function staticMethod() {

echo "Hello World!";

}

}

?>

To access a static method use the class name, double colon (::), and the method name:

Syntax

ClassName::staticMethod();

Let's look at an example:

Example

<?php

class greeting {

public static function welcome() {

echo "Hello World!";

}

}

// Call static method

greeting::welcome();

?>

PHP - More on Static Methods

A class can have both static and non-static methods. A static method can be accessed from a method in the same class using the self keyword and double colon (::):

Example

<?php

class greeting {

public static function welcome() {

echo "Hello World!";

}

public function \_\_construct() {

self::welcome();

}

}

new greeting();

?>

**PHP - Static Properties**

Static properties can be called directly - without creating an instance of a class.

Static properties are declared with the static keyword:

**Syntax**

<?php

class ClassName {

public static $staticProp = "W3Schools";

}

?>

To access a static property use the class name, double colon (::), and the property name:

**Syntax**

ClassName::$staticProp;

Let's look at an example:

**Example**

<?php

class pi {

public static $value = 3.14159;

}

// Get static property

echo pi::$value;

?>

A class can have both static and non-static properties. A static property can be accessed from a method in the same class using the self keyword and double colon (::):

Example

<?php

class pi {

public static $value=3.14159;

public function staticValue() {

return self::$value;

}

}

$pi = new pi();

echo $pi->staticValue();

?>

To call a static property from a child class, use the parent keyword inside the child class:

Example

<?php

class pi {

public static $value=3.14159;

}

class x extends pi {

public function xStatic() {

return parent::$value;

}

}

// Get value of static property directly via child class

echo x::$value;

// or get value of static property via xStatic() method

$x = new x();

echo $x->xStatic();

?>

PHP - What are Traits?

PHP only supports single inheritance: a child class can inherit only from one single parent.

So, what if a class needs to inherit multiple behaviors? OOP traits solve this problem.

Traits are used to declare methods that can be used in multiple classes. Traits can have methods and abstract methods that can be used in multiple classes, and the methods can have any access modifier (public, private, or protected).

Traits are declared with the trait keyword:

Syntax

<?php

trait TraitName {

// some code...

}

?>

To use a trait in a class, use the use keyword:

Syntax

<?php

class MyClass {

use TraitName;

}

?>

Let's look at an example:

Example

<?php

trait message1 {

public function msg1() {

echo "OOP is fun! ";

}

}

class Welcome {

use message1;

}

$obj = new Welcome();

$obj->msg1();

?>