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**.

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:

class
Fruit
objects
Apple
Banana
Mango

Another example:

class

Car

objects

Volvo

Audi

Toyota

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:

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
<?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 are created using the **new** keyword.

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

```
<?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();
?>
```

PHP - The $this Keyword

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

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):

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

```
echo $apple->name;
?>
```

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

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

echo $apple->name;
?>
```

PHP - instanceof

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

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

PHP OOP – Constructor

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:

```php
<?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 OOP – Destructor

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.

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:

```php
<?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 OOP - Access Modifiers

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
<?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
<?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 - What is Inheritance?

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

An inherited class is defined by using the **extends** keyword.

Let's look at an 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();
?>
```

Example Explained

The Strawberry class is inherited from the Fruit class.

This means that the Strawberry class can use the public $name and $color properties as well as the public __construct() and intro() methods from the Fruit class because of inheritance.

The Strawberry class also has its own method: message().

PHP - Inheritance and the Protected Access Modifier

PHP - Overriding Inherited Methods

Inherited methods can be overridden by redefining the methods (use the same name) in the child class.

Look at the example below. The __construct() and intro() methods in the child class (Strawberry) will override the __construct() and intro() methods in the parent class (Fruit):

```php
<?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}.";
  }
}
```

```php
class Strawberry extends Fruit {
  public $weight;
  public function __construct($name, $color, $weight) {
    $this->name = $name;
    $this->color = $color;
    $this->weight = $weight;
  }
  public function intro() {
    echo "The fruit is {$this->name}, the color is {$this->color}, and the weight is {$this->weight} gram.";
  }
}

$strawberry = new Strawberry("Strawberry", "red", 50);
$strawberry->intro();
?>
```

PHP - The final Keyword

The final keyword can be used to prevent class inheritance or to prevent method overriding.

```php
<?php
final class Fruit {
  // some code
}

// will result in error
class Strawberry extends Fruit {
  // some code
}
```

```
?>
```

PHP - Class Constants

Constants cannot be changed once it is declared.

Class constants can be useful if you need to define some constant data within a class.

A class constant is declared inside a class with the **const** keyword.

```
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
}

echo Goodbye::LEAVING_MESSAGE;
?>
```

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:

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 redeclares 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

```php
<?php

// Define an abstract class
abstract class Animal {
    protected $name;

    abstract public function makeSound(); // Abstract
method (no implementation)
```

```php
    public function setName($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }
}

// Define a concrete class that extends the abstract class
class Dog extends Animal {
    public function makeSound() {
        return "Woof!";
    }
}

// Define another concrete class that extends the
abstract class
class Cat extends Animal {
    public function makeSound() {
        return "Meow!";
    }
}

// Create objects from the concrete classes
$dog = new Dog();
$dog->setName("Buddy");
echo $dog->getName(); // Output: Buddy
echo $dog->makeSound(); // Output: Woof!

$cat = new Cat();
$cat->setName("Whiskers");
echo $cat->getName(); // Output: Whiskers
echo $cat->makeSound(); // Output: Meow!
```

```
?>
```

PHP - What are Interfaces?

Interfaces allow you to specify what methods a class should implement.

Interfaces are declared with the **interface** keyword:

PHP - Interfaces vs. Abstract Classes

Interface are similar to abstract classes. The difference between interfaces and abstract classes are:

- **Interfaces cannot have properties, while abstract classes can**
- All interface methods must be public, while abstract class methods is public or protected
- **All methods in an interface are abstract, so they cannot be implemented in code and the abstract keyword is not necessary**

PHP - Using Interfaces

To implement an interface, a class must use the implements keyword.

```
<?php
interface Animal {
  public function makeSound();
}
```

```php
class Cat implements Animal {
  public function makeSound() {
    echo "Meow";
  }
}

$animal = new Cat();
$animal->makeSound();
?>
```

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 declared with the trait keyword:**

```php
<?php
trait message1 {
public function msg1() {
    echo "OOP is fun! ";
  }
}

class Welcome {
  use message1;
}

$obj = new Welcome();
$obj->msg1();
?>
```

PHP - 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!";
  }
}
?>
```

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 (::):

```php
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }

  public function __construct() {
    self::welcome();
  }
}

new greeting();
?>
```

```php
<?php
class A {
  public static function welcome() {
    echo "Hello World!";
  }
}

class B {
  public function message() {
    A::welcome();
  }
```

```
}

$obj = new B();
echo $obj -> message();
?>
```

PHP - Static Properties

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

Static properties are declared with the **static** keyword:

```
<?php
class pi {
  public static $value = 3.14159;
}

// Get static property
echo pi::$value;
?>
```

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

```
<?php
class pi {
  public static $value=3.14159;
}
```

```php
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 Namespaces

Namespaces are qualifiers that solve two different problems:

1. They allow for better organization by grouping classes that work together to perform a task
2. They allow the same name to be used for more than one class

For example, you may have a set of classes which describe an HTML table, such as Table, Row and Cell while also having another set of classes to describe furniture, such as Table, Chair and Bed. Namespaces can be used to organize the classes into two different groups while also preventing the two classes Table and Table from being mixed up.

Declaring a Namespace

Namespaces are declared at the beginning of a file using the namespace keyword:

```php
<?php

namespace Namespace1 {
    class Class1 {
        public function sayHello() {
            echo "Hello from Namespace1\n";
        }
    }
}

namespace Namespace2 {
    class Class2 {
        public function sayHello() {
            echo "Hello from Namespace2\n";
        }
    }
}

// Using classes from the defined namespaces
$obj1 = new Namespace1\Class1();
$obj1->sayHello();

$obj2 = new Namespace2\Class2();
$obj2->sayHello();
```

PHP - What is an Iterable?

An iterable is any value which can be looped through with a foreach() loop.

PHP - Using Iterables

The iterable keyword can be used as a data type of a function argument or as the return type of a function:

```php
<?php
function printIterable(iterable $myIterable) {
  foreach($myIterable as $item) {
    echo $item;
  }
}

$arr = ["a", "b", "c"];
printIterable($arr);
?>
```