INTRO TO OOP

- Object-oriented programming is a style of cod
allows developers to group similar tasks into cl
- PHP treats objects in the same way as referenc
handles, meaning that each variable contains a
reference rather than a copy of the entire objec

OBJECT ORIENTED CONCEPTS

- **Class** – This is a programmer-defined data type, which includes loca
as local data. You can think of a class as a template for making many
same kind (or class) of object.
- **Object** – An individual instance of the data structure defined by a cla
class once and then make many objects that belong to it. Objects ar
instance.
- **Member Variable** – These are the variables defined inside a class. T
invisible to the outside of the class and can be accessed via member
variables are called attribute of the object once an object is created.
- **Member function** – These are the function defined inside a class an
access object data

OBJECT ORIENTED CONCEPTS

- **Parent class** – A class that is inherited from by another class. This is also called a bas
- **Child Class** – A class that inherits from another class. This is also called a subclass or
- **Polymorphism** – This is an object oriented concept where same function can be use purposes. For example function name will remain same but it make take different nu and can do different task.
- **Overloading** – a type of polymorphism in which some or all of operators have differ depending on the types of their arguments. Similarly functions can also be overload implementation.
- **Data Abstraction** – Any representation of data in which the implementation details (abstracted).
- **Encapsulation** – refers to a concept where we encapsulate all the data and member form an object.

STRUCTURING CLASSES

```php
<?php

class MyClass
{
    // Class properties and methods go here
}

$obj = new MyClass;

var_dump($obj); // output: object(MyClass)#1 (0) { }
```

DEFINING CLASS PROPERTIES

```php
<?php

class MyClass
{
    public $prop1 = "I'm a class property!";
}

$obj = new MyClass;

echo $obj->prop1; // Output: I'm a class property!
```

DEFINING CLASS METHODS

```php
<?php

class MyClass
{
    public $prop1 = "I'm a class property!";

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1;
    }
}

$obj = new MyClass;

echo $obj->getProperty(); // Get the property value, outputs: I'm a class pr
```

OBJECTS

```php
<?php

class MyClass
{
    public $prop1 = "I'm a class property!";

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1;
    }
}

// Create two objects
$obj = new MyClass;
$obj2 = new MyClass;

// Get the value of $prop1 from both objects
echo $obj->getProperty();
echo $obj2->getProperty();

// Set new values for both objects
```

MAGIC METHODS IN PHP OOP

- To make the use of objects easier, PHP also pro number of magic methods, or special methods called when certain common actions occur wit
- This allows developers to perform a number o tasks with relative ease.
- The function names `__construct()`, `__dest` `__call()`, `__callStatic()`, `__get()`, `__se`

CONSTRUCTOR & DESCTRUCTOR

```php
<?php

class MyClass
{
    public $prop1 = "I'm a class property!";

    public function __construct()
    {
        echo 'The class "', __CLASS__, '" was initiated!';
    }

    public function __destruct()
    {
        echo 'The class "', __CLASS__, '" was destroyed.';
    }

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1;
    }
}

// Create a new object
$obj = new MyClass;
```

CONVERTING TO STRING

```
<?php

// Output the object as a string
echo $obj; // outputs: Catchable fatal error: Object of class MyClass could not be conv


class MyClass
{
    // ...

    public function __toString()
    {
        echo "Using the toString method: ";
        return $this->getProperty();
    }

    // ...
}

// Create a new object
$obj = new MyClass;

// Output the object as a string
echo $obj; // outputs: Using the toString method: I'm a class property!
```

USING CLASS INHERITANCE

```
<?php

class MyOtherClass extends MyClass
{
    public function newMethod()
    {
        echo "From a new method in " . __CLASS__;
    }
}

// Create a new object
$newobj = new MyOtherClass;

// Output the object as a string
echo $newobj->newMethod(); // outputs: From a new method in MyOtherCla

// Use a method from the parent class
echo $newobj->getProperty(); // outputs: I'm a class property!
```

OVERWRITING INHERITED PROPERTIES AND METHOD

```php
<?php

class MyOtherClass extends MyClass
{
    public function __construct()
    {
        echo "A new constructor in " . __CLASS__;
    }

    public function newMethod()
    {
        echo "From a new method in " . __CLASS__;
    }
}

// Create a new object
$newobj = new MyOtherClass; // outputs: A new constructor in MyOtherCl

// Output the object as a string
```

PRESERVING ORIGINAL METHOD FUNCTIONALITY WHILE OVERWR

```php
<?php

class MyOtherClass extends MyClass
{
    public function __construct()
    {
        parent::__construct(); // Call the parent class's constructor using 
operator (::)
        echo "A new constructor in " . __CLASS__;
    }

    public function newMethod()
    {
        echo "From a new method in " . __CLASS__;
    }
}

// Create a new object
$newobj = new MyOtherClass;

// Output the object as a string
```

ASSIGNING THE VISIBILITY OF PROPERTIES AND MET

- For added control over objects, methods and p are assigned visibility.
- This controls how and from where properties a can be accessed.
- There are three visibility keywords: `public`, `pr` and `private`.
- In addition to its visibility, a method or propert

PROTECTED PROPERTIES AND METHODS

```php
<?php

class MyClass
{
    // ..

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    protected function getProperty()
    {
        return $this->prop1;
    }
}

class MyOtherClass extends MyClass
{
    public function __construct()
    {
        parent::__construct();
        echo "A new constructor in " . __CLASS__;
    }

    public function newMethod()
    {
        echo "From a new method in " . __CLASS__;
```

PROTECTED PROPERTIES AND METHODS

```php
<?php

class MyOtherClass extends MyClass
{
    public function __construct()
    {
        parent::__construct();
        echo "A new constructor in " . __CLASS__ . ".<br />";
    }

    public function newMethod()
    {
        echo "From a new method in " . __CLASS__ . ".<br />";
    }

    public function callProtected()
    {
        return $this->getProperty();
    }

// Create a new object
```

PRIVATE PROPERTIES AND METHODS

```php
<?php

class MyClass
{
    // ..

    private function getProperty()
    {
        return $this->prop1;
    }
}

class MyOtherClass extends MyClass
{
    public function __construct()
    {
        parent::__construct();
        echo "A new constructor in " . __CLASS__;
    }

    public function newMethod()
    {
        echo "From a new method in " . __CLASS__;
    }

    public function callProtected()
    {
        return $this->getProperty();
```

STATIC PROPERTIES AND METHODS

```php
<?php

/*
A method or property declared static can be accessed without first instantia
simply supply the class name, scope resolution operator, and the property or
*/

class MyClass
{
    // ...

    public static $count = 0;

    // ...

    public static function plusOne()
    {
        return "The count is " . ++self::$count;
    }
}
```

ABSTRACT CLASSES AND METHODS

```php
<?php

abstract class Car {
    // Abstract classes can have properties
    protected $tankVolume;

    // Abstract classes can have non abstract methods
    public function setTankVolume($volume)
    {
        $this -> tankVolume = $volume;
    }

    // Abstract method
    abstract public function calcNumMilesOnFullTank();
}

class Honda extends Car {
    // Since we inherited abstract method, we need to define it in the child class,
    // by adding code to the method's body.
    public function calcNumMilesOnFullTank()
    {
        $miles = $this -> tankVolume*30;
        return $miles;
```

INTERFACES

```php
<?php

/* Interfaces resemble abstract classes in that they include abstract methods that the programmer must define in
from the interface. In this way, interfaces contribute to code organization because they commit the child classes
they should implement.*/
interface Car {
    public function setModel($name);
    public function getModel();
}

interface Vehicle {
    public function setHasWheels($bool);
    public function getHasWheels();
}

class miniCar implements Car, Vehicle {
    private $model;
    private $hasWheels;

    public function setModel($name)
    {
        $this -> model = $name;
    }

    public function getModel()
    {
        return $this -> model;
    }

    public function setHasWheels($bool)
```

DIFFERENCES BETWEEN ABSTRACT CLASSES AND IN

	interface		abstract
the code	- abstract methods - constants		- abstract r - const - concrete r - concrete v
access modifiers	- public	- - - -	- pu - priv - prot - sta
number of	The same class can implement		The child class ca

POLYMORPHISM

```php
<?php

/* According to the Polymorphism principle, methods in different classes that do similar things should have the

interface Shape {
    public function calcArea();
}
class Circle implements Shape {
    private $radius;

    public function __construct($radius)
    {
        $this -> radius = $radius;
    }

    public function calcArea() // calcArea calculates the area of circles
    {
        return $this -> radius * $this -> radius * pi();
    }
}
class Rectangle implements Shape {
    private $width;
    private $height;

    public function __construct($width, $height)
    {
        $this -> width = $width;
        $this -> height = $height;
    }
```

TYPE HINTING

```php
<?php

// The function can only get array as an argument.
function calcNumMilesOnFullTank(array $models)
{
    foreach ($models as $item) {
        echo $carModel = $item[0];
        echo " : ";
        echo $numberOfMiles = $item[1] * $item[2];
    }
}

calcNumMilesOnFullTank("Toyota"); // outputs: Catchable fatal error: Argument 1 passed to calcNumMilesOnF
type array, string given

$models = array(
    array('Toyota', 12, 44),
    array('BMW', 13, 41)
);
calcNumMilesOnFullTank($models);


class Car {
    protected $driver;

    // The constructor can only get Driver objects as arguments.
    public function __construct(Driver $driver)
    {
```

TYPE HINTING IN PHP7

```php
<?php

class car {
    protected $model;
    protected $hasSunRoof;
    protected $numberOfDoors;
    protected $price;

    // string type hinting
    public function setModel(string $model)
    {
        $this->model = $model;
    }

    // boolean type hinting
    public function setHasSunRoof(bool $value)
    {
        $this->hasSunRoof = $value;
    }

    // integer type hinting
    public function setNumberOfDoors(int $value)
    {
        $this->numberOfDoors = $value;
```

NAMESPACES

```php
<?php
// application library 1 - lib1.php
namespace App\Lib1;

const MYCONST = 'App\Lib1\MYCONST';

function MyFunction() {
    return __FUNCTION__;
}

class MyClass {
    static function WhoAmI() {
        return __METHOD__;
    }
}

/* ---------------------------------- */

require_once(‘lib1.php');
```

NAMESPACES - WITHIN THE SAME NAMESPACE

```
<?php

// application library 2 -- lib2.php
namespace App\Lib2;

const MYCONST = 'App\Lib2\MYCONST';

function MyFunction() {
    return __FUNCTION__;
}

class MyClass {
    static function WhoAmI() {
        return __METHOD__;
    }
}
/* ---------------------------------- */

namespace App\Lib1;

require_once('lib1.php');
```

NAMESPACES - IMPORTING

```php
<?php

use App\Lib2;

require_once('lib1.php');
require_once('lib2.php');

echo Lib2\MYCONST; // outputs: App\Lib2\MYCONST
echo Lib2\MyFunction(); // outputs: App\Lib2\MyFunction
echo Lib2\MyClass::WhoAmI(); // outputs: App\Lib2\MyClass::WhoAmI
```

NAMESPACES - ALIASES

```php
<?php

use App\Lib1 as L;
use App\Lib2\MyClass as Obj;

require_once('lib1.php');
require_once('lib2.php');

echo L\MYCONST; // outputs: App\Lib1\MYCONST
echo L\MyFunction(); // outputs: App\Lib1\MyFunction
echo L\MyClass::WhoAmI(); // outputs: App\Lib1\MyClass::WhoAmI
echo Obj::WhoAmI(); // outputs: App\Lib2\MyClass::WhoAmI
```

TRAITS

```
<?php

/* Traits are a mechanism for code reuse in single inheritance languag
intended to reduce some limitations of single inheritance by enabling
reuse sets of methods freely in several independent classes living in
hierarchies */

trait ezcReflectionReturnInfo {
    function getReturnType() { /*1*/ }
function getReturnDescription() { /*2*/ }
}

class ezcReflectionMethod extends ReflectionMethod {
    use ezcReflectionReturnInfo;
    /* ... */
}

class ezcReflectionFunction extends ReflectionFunction {
    use ezcReflectionReturnInfo;
```

TRAITS

```php
<?php

class Base
{
    public function sayHello()
    {
        echo 'Hello ';
    }
}

trait SayWorld
{
    public function sayHello()
    {
        parent::sayHello();
        echo 'World!';
    }
}

class MyHelloWorld extends Base
{
```

TRAITS

```php
<?php

trait HelloWorld {
    public function sayHello() {
        echo 'Hello World!';
    }
}

class TheWorldIsNotEnough {
    use HelloWorld;
    public function sayHello() {
        echo 'Hello Universe!';
    }
}

$o = new TheWorldIsNotEnough();
$o->sayHello(); // outputs: Hello Universe!
```

TRAITS

```php
<?php
trait Hello {
    public function sayHello() {
        echo 'Hello ';
    }
}

trait World {
    public function sayWorld() {
        echo 'World';
    }
}

class MyHelloWorld {
    use Hello, World;
    public function sayExclamationMark() {
        echo '!';
    }
}

$o = new MyHelloWorld();
$o->sayHello();
$o->sayWorld();
$o->sayExclamationMark(); // outputs: Hello World!
```

COMMENTING WITH DOCBLOCKS

```php
<?php

/**
 * A simple class
 *
 * This is the long description for this class,
 * which can span as many lines as needed. It is
 * not required, whereas the short description is
 * necessary.
 *
 * @author Denis Ristic <denis.ristic@perpetuum.hr>
 * @copyright 2017 Perpetuum Mobile
 * @license http://www.php.net/license/3_01.txt PHP License 3.01
 */
class SimpleClass
{
    /**
     * A public variable
     *
     * @var string stores data for the class
     */
    public $foo;

    /**
     * Sets $foo to a new value upon class instantiation
     *
     * @param string $val a value required for the class
     * @return void
     */
    public function __construct($val)
    {
        $this->foo = $val;
    }

    /**
     * Multiplies two integers
     *
     * Accepts a pair of integers and returns the
     * product of the two.
```

PHP OOP REFERENCES

- PHP Documentation
  - http://php.net/manual/en/language.oop5.php
- Object-Oriented PHP for Beginners
  - https://code.tutsplus.com/tutorials/object-oriented-php-for-begi
- Learn Object-oriented PHP
  - http://phpenthusiast.com/object-oriented-php-tutorials
- How to Use PHP Namespaces
  - https://www.sitepoint.com/php-53-namespaces-basics/