1:- What Is Polymorphism? Explain with an Example.

This word is can from Greek word poly and morphism. **Poly** means "many" and **morphism** means property which help us to assign more than one property. => **Overloading** Same method name with different signature, since PHP doesn't support method overloading concept => **Overriding** When same methods defined in parents and child class with same signature i.e know as method overriding.

oop1.php > Circle

```php
<?php

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

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

    // calcArea calculates the area of circles
    public function calcArea()
    {
        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;
    }

    // calcArea calculates the area of rectangles
    public function calcArea()
    {
        return $this->width * $this->height;
    }
}
$circ = new Circle(3);
echo 'Area of Circle with Radius 3 : ' . $circ->calcArea();
$rect = new Rectangle(3, 4);
echo '<br>Area of Rectangle with Sides 3,4 : ' . $rect->calcArea();

```

OUTPUT-

```
Area of Circle with Radius 3 : 28.274333882308
Area of Rectangle with Sides 3,4 : 12
```

2:- (a) How To Load Classes In Php? (b) How To Call Parent Constructor? Explain with an Example.

In PHP 5, we define an __autoload function which is automatically called in case we are trying to use a class/interface which hasn't been defined yet.

This is how it works in action. We will create two classes. So create Image.php file

image.php > ...

```php
<?php
    class Image {

        function __construct() {
            echo 'Class Image loaded successfully <br />';
        }

    }
    //$obj = new Image();
?>
```

Now create Test.php file

```php
<?php
class Test {

    function __construct() {
        echo 'Class Test working <br />';
    }

}
//$obj = new Test();
?>
```

Basically, we created 2 simple classes with constructors which echo some text out. Now, create a file index.php

```php
<?php
function __autoload($class_name) {
    require_once ($class_name . '.php');
}

$a = new test();
$b = new image();
```

Output:

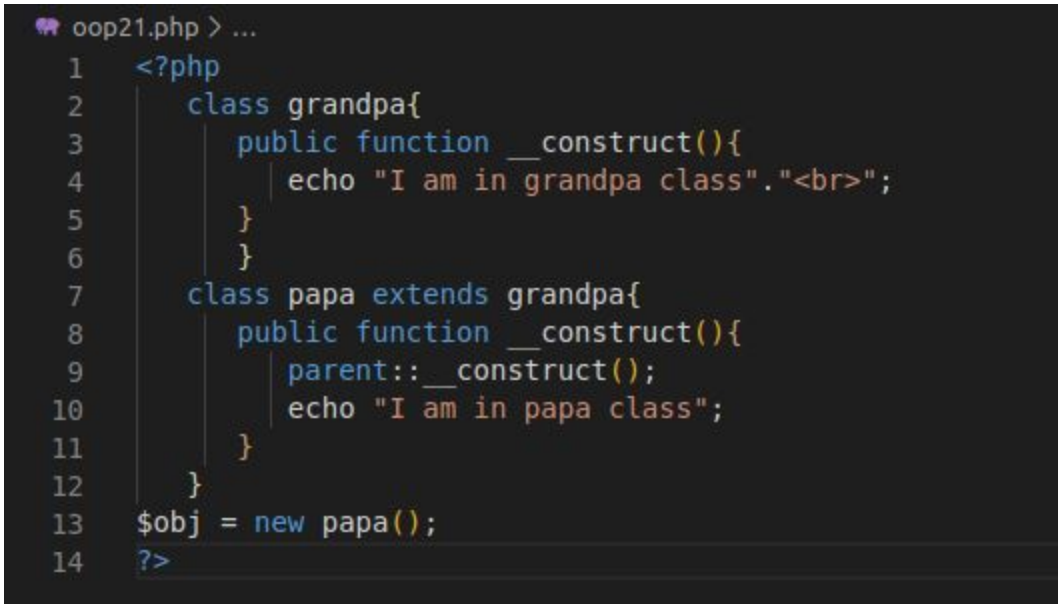
```
Class Test working
Class Image loaded successfully
```

When you run index.php in browser, everything is working fine (assuming all 3 files are in the same folder). Maybe you don’t see a point, but imagine that you have 10 or more classes and have to write require_once as many times.

(b) How To Call Parent Constructor?

- **CASE1:**

  We can't run directly the parent class constructor in child class if the child class defines a constructor. In order to run a parent constructor, a call to parent::__construct() within the child constructor is required.



```php
<?php
    class grandpa{
        public function __construct(){
            echo "I am in grandpa class"."<br>";
        }
        }
    class papa extends grandpa{
        public function __construct(){
            parent::__construct();
            echo "I am in papa class";
        }
    }
$obj = new papa();
?>
```

Output:

```
I am in grandpa class
I am in papa class
```

- **CASE2:**

  If the child does not define a constructor then it may be inherited from the parent class just like a normal class method(if it was not declared as private).

  Output:

```
I am in Grandpa class but calling it in papa class
```

3:- What is overloading and overriding in php?

- Function overloading and overriding is the OOPs feature in PHP. In function overloading, more than one function can have same method signature but different number of arguments. But in case of function overriding, more than one functions will have same method signature and number of arguments.
- **Function Overloading:** Function overloading contains same function name and that function preforms different task according to number of arguments. For example, find the area of certain shapes where radius are given then it should return area of circle if height and width are given then it should give area of rectangle and others. Like other OOP languages function overloading can not be done by native approach. In PHP function overloading is done with the help of magic function __call(). This function takes function name and arguments.

EXAMPLE

overload.php > ...

```php
<?php
// overloading in PHP

// Creating a class of type shape
class shape
{

    // __call is magic function which accepts function name and arguments

    public function __call($fn_name, $arguments)
    {

        // It will match the function name
        if ($fn_name == 'area') {

            switch (count($arguments)) {

                // If there is only one argument
                // area of circle
                case 1:
                    return 3.14 * $arguments[0];

                // IF two arguments then area is rectangel;
                case 2:
                    return $arguments[0] * $arguments[1];
            }
        }
    }
}

// Declaring a shape type object
$s = new Shape;

// Functio call
echo "Area of Circle having one argument radius : ".($s->area(2));
echo "<br>";

// calling area method for rectangel
echo "Area of Rectangle having two arguments : ".($s->area(4, 2));

```

OUTPUT:

Area of Circle having one argument radius : 6.28
Area of Rectangle having two arguments : 8

- **Function Overriding:** Function overriding is same as other OOPs programming languages. In function overriding, both parent and child classes should have same function name with and number of arguments. It is used to replace parent method in child class. The purpose of overriding is to change the behavior of parent class method. The two methods with the same name and same parameter is called overriding.

EXAMPLE:

override.php > ...

```php
<?php
// function overriding

// This is parent class
class Parents
{

    // Function father of parent class
    public function father()
    {
        echo "Parents";
    }
}

// This is child class
class Child extends Parents
{

    // Overriding father method
    public function father()
    {
        echo "<br> Child";
    }
}

// Reference type of parent
$p = new Parents;

// Reference type of child
$c = new Child;

// print Parent
$p->father();

// Print child
$c->father();
```

OUTPUT:

Parents
Child

4:- What are the magic methods in php?

- Generally, for each PHP user defined function, it contains two portions, such as function definition and function call. In some special cases, PHP functions will have function declaration. For example, PHP interfaces functions or PHP abstract functions, that we have seen with abstract access modifiers.
- In PHP, we can define some special functions that will be called automatically. Such functions require no function call to execute the code inside these functions. With this special feature, they can be referred as magic functions or magic methods.
- *__construct()/__destruct()* – We have seen enough about these two magic methods while discussing constructors and destructors which is one of the object oriented feature supported in PHP.
- *__get()/__set()* – These are magic getters and setters for getting and putting values for class properties created dynamically by PHP property overloading.
- *__isset()* – This magic method will be invoked automatically while checking whether a required overloaded property is set or not, by using the PHP isset() function.

- *__unset()* – Similarly, when we call PHP unset() function on such dynamically created properties, this magic method will automatically be invoked.
- *__call()/__callStatic()* – These two magic methods are dedicated for accessing dynamically created but invisible methods on PHP method overloading. These differ, where __call() will invoke normal PHP overloaded methods, and *__callStatic()* will invoke static methods

5:- What is STATIC keyword and what is it's use in PHP ?

Any method declared as static is accessible without the creation of an object. Static functions are associated with the class, not an instance of the class. They are permitted to access only static methods and static variables. To add a static method to the class, static keyword is used.

```
public static function test()
{
    // Method implementation
}
```

They can be invoked directly outside the class by using scope resolution operator (::) as follows:

```
MyClass::test();
```

Example:

static.php > ...

```php
<?php
/* Use static function as a counter */

class solution {

    static $count;

    public static function getCount() {
        return self::$count++;
    }
}

solution::$count = 1;

for($i = 0; $i < 5; ++$i) {
    echo 'The next value is: '.
    solution::getCount() . "<br>";
}

?>
```

OUTPUT:

The next value is: 1
The next value is: 2
The next value is: 3
The next value is: 4
The next value is: 5

6:- What is Traits in PHP? What is Conflict Resolution in Traits?

Traits are a mechanism for code reuse in single inheritance languages such as PHP. A Trait is intended to reduce some limitations of single inheritance by enabling a developer to reuse sets of methods freely in several independent classes living in different class hierarchies. The semantics of the combination of Traits and classes is defined in a way which reduces complexity, and avoids the typical problems associated with multiple inheritance and Mixins.

- **CONFLICT RESOLUTION**

If two Traits insert a method with the same name, a fatal error is produced, if the conflict is not explicitly resolved.

To resolve naming conflicts between Traits used in the same class, the *insteadof* operator needs to be used to choose exactly one of the conflicting methods.

Since this only allows one to exclude methods, the *as* operator can be used to add an alias to one of the methods. Note the *as* operator does not rename the method and it does not affect any other method either

EXAMPLE:

trait.php > ...

```php
<?php
trait oldMan
{
    public function sayHi()
    {
        echo 'OldMan says Hola!';
    }
}

trait newMan
{
    public function sayHi()
    {
        echo 'newMan says Hello!';
    }
}

trait futureMan
{
    public function sayHi()
    {
        echo "FutureMan says Hello";
    }
}
class Human
{
    use oldMan, newMan, futureMan {
        futureMan::sayHi insteadof oldMan;
        newMan::sayHi insteadof futureMan;
    }

}

$human1 = new Human();
$human1->sayHi();

```

OUTPUT:

newMan says Hello!

7:- How to merge two PHP objects?

1. **CASE1:** Convert object into data array and merge them using array_merge() function and convert this merged array back into object of class *stdClass*.

arr1.php > ...

```php
<?php
// PHP prgoram to merge two objects

class teacher {
    // Empty class
}

$objectA = new teacher();
$objectA->a = 1;
$objectA->b = 2;
$objectA->d = 3;

$objectB = new teacher();
$objectB->d = 4;
$objectB->e = 5;
$objectB->f = 6;

$obj_merged = (object) array_merge(
        (array) $objectA, (array) $objectB);

print_r($obj_merged);

?>
```

```
stdClass Object
(
    [a] => 1
    [b] => 2
    [d] => 4
    [e] => 5
    [f] => 6
)
```

2. **CASE2:** Create a new object of the original class and assign all the properties of both objects to this new object by using foreach loop. This is a simple and clean approach of merging two objects.

arr2.php > ...

```php
class Teacher {
    // Empty class
}

$objectA = new Teacher();
$objectA->a = 1;
$objectA->b = 2;
$objectA->d = 3;

$objectB = new Teacher();
$objectB->e = 4;
$objectB->f = 5;
$objectB->g = 6;

// Function to convert class of given object
function convertObjectClass($objectA,
                    $objectB, $final_class) {

    $new_object = new $final_class();

    // Initializing class properties
    foreach($objectA as $property => $value) {
        $new_object->$property = $value;
    }

    foreach($objectB as $property => $value) {
        $new_object->$property = $value;
    }

    return $new_object;
}

$obj_merged = convertObjectClass($objectA,
                        $objectB, 'Teacher');

print_r($obj_merged);

?>
```

```
Teacher Object
(
    [a] => 1
    [b] => 2
    [d] => 3
    [e] => 4
    [f] => 5
    [g] => 6
)
```

3. **CASE3:** Merge the object using array_merge() method and convert this merged array to object using convertObjectClass function. This function is used to convert object of the initial class into serialized data using serialize() method. Unserialize the serialized data into instance of the final class using unserialize() method. Using this approach obtain an object of user defined class *Geeks* rather the standard class *stdClass*.

arr3.php > ...

```php
<?php
// PHP prgoram to merge two objects

class Teacher {
    // Empty class
}

$objectA = new Teacher();
$objectA->a = 1;
$objectA->b = 2;
$objectA->d = 3;

$objectB = new Teacher();
$objectB->d = 4;
$objectB->e = 5;
$objectB->f = 6;

// Function to convert class of given object
function convertObjectClass($array, $final_class) {
    return unserialize(sprintf(
        'O:%d:"%s"%s',
        strlen($final_class),
        $final_class,
        strstr(serialize($array), ':')
    ));
}

$obj_merged = convertObjectClass(array_merge(
        (array) $objectA, (array) $objectB), 'Teacher');

print_r($obj_merged);

?>
```

```
Teacher Object
(
    [a] => 1
    [b] => 2
    [d] => 4
    [e] => 5
    [f] => 6
)
```

8:- What is instanceof keyword?

instanceof is used to determine whether a PHP variable is an instantiated object of a certain class:
It is a type operator
EXAMPLE:

inst.php

```php
<?php
interface MyInterface
{
}

class MyClass implements MyInterface
{
}

$a = new MyClass;
$b = new MyClass;
$c = 'MyClass';
$d = 'NotMyClass';

var_dump($a instanceof $b); // $b is an object of class MyClass
var_dump($a instanceof $c); // $c is a string 'MyClass'
var_dump($a instanceof $d); // $d is a string 'NotMyClass'
?>
```

OUTPUT:

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