**What is oops ?**

oops stands for object oriented programming structure

oops used to create a dynamic projects

oops used to create a secured web application or website

oops is used to create a application i.e support MVC architectures

**OOPS Features :-**

1. **Inheritance**
2. **Interface**
3. **Abstract**
4. **Encapsulation**
5. **Polymorphism**
6. **Class**
7. **Object**

**CLASS:- A class is a member and its member function.**

**Class ia an instance of object.**

<?php

class test

{

    public function display()

    {

        $name="Hello php!";

        echo $name;

    }

}

$obj=new test;

$obj->display();

// Output - HEllo php!

?>

**OBJECT:- An object is instances of class. Object will be called through new keyword.**

<?php

/\*

Syntax of Object:-

class myclass

{

   ...

}

$obj=new myclass;

\*/

class item

{

    public function display()

    {

    $name="hello i am php";

    echo $name;

    }

}

$obj=new item;

$obj->display();

// Output - hello i am php

?>

**PSEUDO VARIABLE:- Pseudo variable is used to call direct inside of method or globally access any variables in the method.**

**It is used with $this variable.**

<?php

class test

{

    public $name="php is backed language!";

    public function display()

    {

        echo $this->name;

    }

}

$obj=new test;

$obj->display();

// Output - php is backed language!

?>

**INHERITANCE:- Inheritance is used to access one class property with another class it is called inheritance.**

**There are 3 types of inheritance:**

1. **Single Inheritance**

**Here in single inheritance one class property only access to its one child.**

<?php

class phone

{

    public function item()

    {

        $name="IPHONE 13 is best Smartphone";

        echo $name."<br>";

    }

}

class brand extends phone

{

    public function item1()

    {

        $name="Iphone brand name is Apple";

        echo $name."<br>";

    }

}

$obj=new brand;

$obj->item();

$obj->item1();

// Output - IPHONE 13 is best Smartphone

//          Iphone brand name is Apple

?>

1. **Multilevel Inheritance**

**Here in multilevel inheritance if a=>b=>c=>d if this chain occurs then it is multilevel inheritance.**

<?php

class phone

{

    public function item()

    {

        $name="Iphone 14 is best Smartphone";

        echo $name."<br>";

    }

}

class brand extends phone

{

    public function item1()

    {

        $name="Iphone brand name is Apple";

        echo $name."<br>";

    }

}

class purchase extends brand

{

    public function item2()

    {

        $name="I have purchased Apple iphone from flipkart";

        echo $name."<br>";

    }

}

class date extends purchase

{

    public function item3()

    {

        $name="the date i have purchased is 20th july 2022";

        echo $name."<br>";

    }

}

$obj=new date();

$obj->item();

$obj->item1();

$obj->item2();

$obj->item3();

// output - Iphone 14 is best Smartphone

//          Iphone brand name is Apple

//          I have purchased Apple iphone from flipkart

//          the date i have purchased is 20th july 2022

?>

1. **Multiple Inheritance**

**Multiple inheritance is not supported in php & java it is only supported in c++ and python.**

<?php

//Note : multiple inheritance is not supported in php and java only c++ and python will support multiple inheritance

class phone

{

    public function item()

    {

        $name="IPHONE 13 is best Smartphone";

        echo $name."<br>";

    }

}

class brand

{

    public function item1()

    {

        $name="Iphone brand name is Apple";

        echo $name."<br>";

    }

}

class purchase extends phone,brand

{

    public function itme2()

    {

        $name="i have purchased on july 2022";

        echo $name;

    }

}

$obj=new purchase;

$obj->item();

$obj->item1();

$obj->item2();

?>

**INTERFACE:-** Interfaces allow you to specify what methods a class should implement.

Interfaces make it easy to use a variety of different classes in the same way. When one or more classes use the same interface, it is referred to as "polymorphism".

Interfaces are declared with the interface keyword:

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

<?php

interface A

{

    public function item(); // declared a function

}

interface B

{

    public function item1(); // declared a function

}

interface C

{

    public function item2(); // declared a function

}

interface D

{

    public function item3(); // declared a function

}

class E implements A,B,C,D

{

    public function item()

    {

        echo "Hello i am learning PHP"."<br>";

    }

    public function item1()

    {

        echo "Hello i am learning Bootstrap"."<br>";

    }

    public function item2()

    {

        echo "Hello i am learning Javascript"."<br>";

    }

    public function item3()

    {

        echo "Hello i am learning CSS"."<br>"   ;

    }

}

$obj=new E;

$obj->item();

$obj->item1();

$obj->item2();

$obj->item3();

// Output - Hello i am learning PHP

//          Hello i am learning Bootstrap

//          Hello i am learning Javascript

//          Hello i am learning CSS

?>

**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:

<?php

trait A

{

    public function item()

    {

        $name="Hi How are you?"."<br>";

        echo $name;

    }

}

trait B

{

    public function item1()

    {

        $name="I am Fine!"."<br>";

        echo $name;

    }

}

trait C

{

    public function item2()

    {

        $name="What is your profession?"."<br>";

        echo $name;

    }

}

trait D

{

    public function item3()

    {

        $name="I am doing Job!"."<br>";

        echo $name;

    }

}

class E

{

    use A,B,C,D;

}

$obj=new E;

$obj->item();

$obj->item1();

$obj->item2();

$obj->item3();

// Output:-Hi How are you?

//         I am Fine!

//         What is your profession?

//         I am doing Job!

?>

**POLYMORPHISM:-** Polymorphism in OOPs is a concept that allows you to create classes with different functionalities in a single interface.

Polymorphism in php does not support method overloading.

1. **Method Overloading (compiletime)**

<?php

class A

{

    public function number($num1,$num2)

    {

        $num=$num1+$num2;

        echo $num;

    }

}

class B extends A

{

    public function number($num1,$num2,$num3)

    {

        $num=$num1\*$num2\*$num3;

        echo $num;

    }

}

$obj=new B;

$obj->number(2,7,9);

?>

1. **Method Overriding (runtime)**

<?php

class A

{

    public function display($num1,$num2)

     {

        $num=$num1\*$num2;

        echo $num;

     }

}

class B extends A

{

    public function display($num1,$num2)

    {

        $num=$num1+$num2;

        echo $num;

    }

}

$obj=new B;

$obj->display(50,90);

//Output:-140

**CONSTRUCTOR:-** 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.

1. **User defined constructor**

<?php

    class item

    {

        public function item()

        {

           $name="My name is patel";

           echo $name;

        }

    }

$boj=new item;

// User Defined Constructor does not support the php version 8, it gives the output in verion 7 or earlier.

// so that's why we do not use the user defined constructor.

?>

1. **Default Constructor**

<?php

class A

{

    public function \_\_construct()

    {

        $name="Hello how are you?";

        echo $name;

    }

}

$boj=new A;

//Output:- Hello how are you?

?>

1. **Parameterized Constructor**

<?php

class item

{

    public function \_\_construct($num1,$num2)

    {

        $number=$num1\*$num2;

        echo "Multiplication of two number is : ".$number;

    }

}

$obj=new item(70,50);

// Output:- Multiplication of two number is : 3500

?>