OOPS Fundamentals

(B1) What is class in Object Oriented Programming Language?

-> **In object-oriented programming, a class is a template definition of the method s and variable s in a particular kind of object.**

Thus, an object is a specific instance of a class; it contains real values instead of variables.

The class is one of the defining ideas of object-oriented programming.

A class is a way of organizing information about a type of data so a programmer can reuse elements when making multiple instances of that data type

For example,

If a programmer wanted to make three instances of Car, maybe a BMW, a Ferrari, and a Ford instance.

(B2) What is an Object in Object Oriented Programming Language?

->

In object-oriented programming (OOP), objects are the things you think about first in designing a program and they are also the units of code that are eventually derived from the process.

In between, each object is made into a generic class of object, and even more generic classes are defined so that objects can share models and reuse the class definitions in their code.

Each object is an instance of a particular class or subclass with the class's own methods or procedures and data variables.

An object is what actually runs in the computer.

In object-oriented programming (OOP), objects are the things you think about first in designing a program and they are also the units of code that are eventually derived from the process.

(B3) What Is Difference Between Class And Interface?

->

Class:

A class can be instantiated i.e.,

Objects of a class can be created.

Classes does not support multiple inheritance.

Interface:

A Interface cannot be instantiated i.e.,

Objects cannot be created.

Interface supports multiple inheritance.

An instance of a class is an object. It is also known as a class object or class instance.

As such, instantiation may be referred to as construction.

Whenever values vary from one object to another, they are called instance variables.

(B4) What Is Method Overloading in Object Oriented Programming Language?

->Overloading in PHP provides means to dynamically create properties and methods. These dynamic entities are processed via magic methods one can establish in a class for various action types.

(B5)What Is Data hiding in Object Oriented Programming Language?

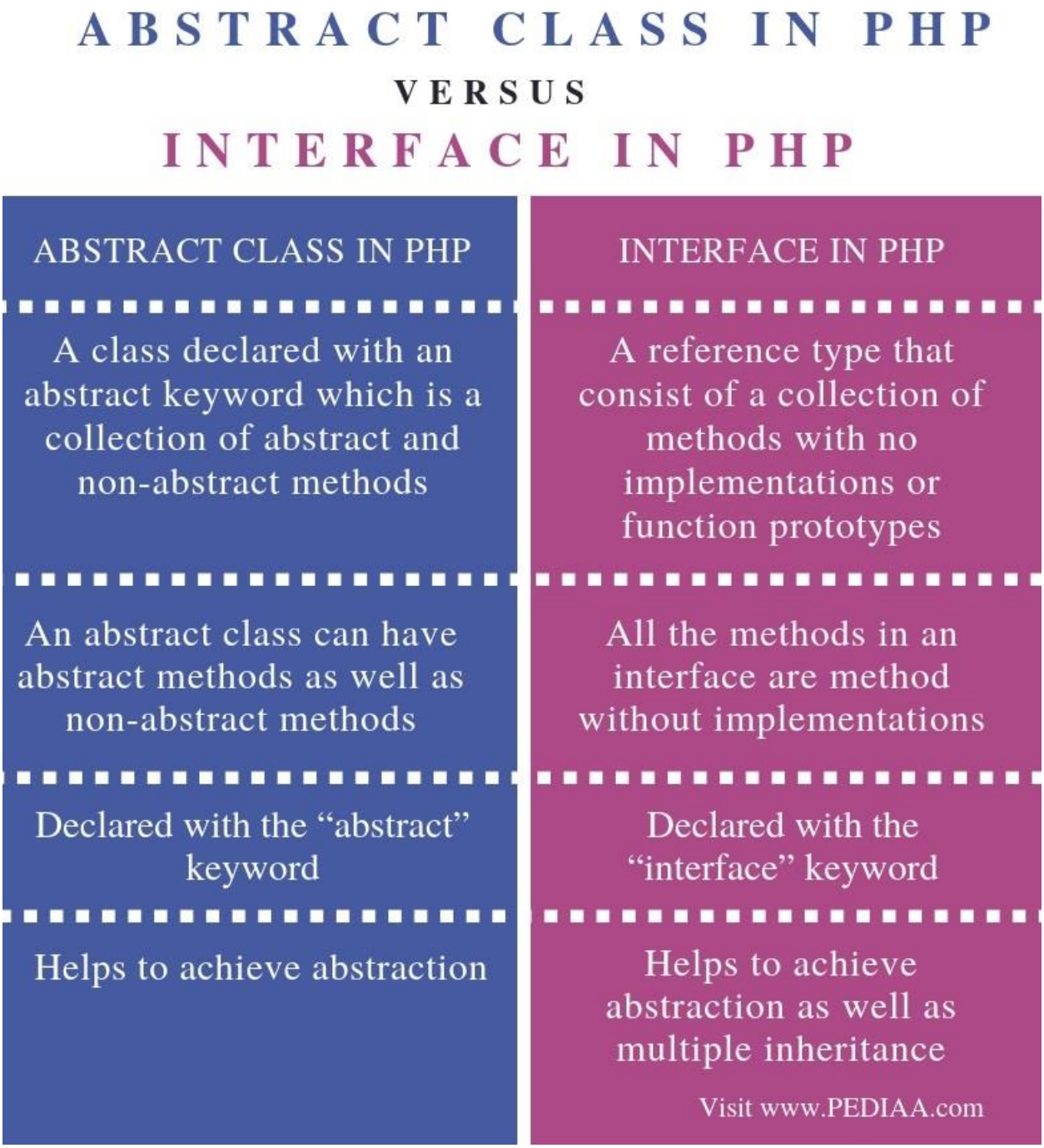
-> Data hiding is an object-oriented programming (OOP) technique specifically used to hide internal object details (i.e., data members).Data hiding guarantees exclusive data access to class members only and protects and maintains object integrity by preventing intended or unintended changes and intrusions.

(B7) What are the Virtual Function in Object Oriented Programming?

->A virtual function is a member function that you expect to be redefined in derived classes. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

(B6) What are the differences between abstract classes and interfaces?

->



(B8) What is Constructor in Object Oriented Programming?

->

In class-based, object-oriented programming, a constructor is a special type of subroutine called to create an object.

It prepares the new object for use, often accepting arguments that the constructor uses to set required member variables.

A constructor resembles an instance method, but it differs from a method in that it has no explicit return type, it is not implicitly inherited and it usually has different rules for scope modifiers.

Constructors often have the same name as the declaring class.

They have the task of initializing the object's data members and of establishing the invariant of the class, failing if the invariant is invalid.

A properly written constructor leaves the resulting object in a *valid* state.

Immutable objects must be initialized in a constructor.

Types:

Parameterized constructors:

Constructors that can take at least one argument are termed as parameterized constructors. When an object is declared in a parameterized constructor, the initial values have to be passed as arguments to the constructor function. The normal way of object declaration may not work. The constructors can be called explicitly or implicitly. The method of calling the constructor implicitly is also called the shorthand method.

Default constructors:

If the programmer does not supply a constructor for an instantiable class, Java compiler inserts a default constructor into your code on your behalf. This constructor is known as default constructor. You would not find it in your source code as it would be inserted into the code during compilation and exists in .class file. The behavior of the default constructor is language dependent.

Conversion constructors:

Conversion constructors provide a means for a compiler to implicitly create an object belonging to one class based on an object of a different type. These constructors are usually invoked implicitly to convert arguments or operands to an appropriate type, but they may also be called explicitly.

(B9) What is Abstract class in Object Oriented Programming?

-> Abstract classes and methods are when the parent class has a named method, but need its child class 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;

}

?>
```

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

Example:

```
<?php

// Parent class
abstract class Car

 {
  public $name;
  public function __construct($name)

{
    $this->name = $name;
  }
  abstract public function intro() : string;
}

// Child classes
class Audi extends Car
```

```
 {
  public function intro() : string {
    return "Choose German quality! I'm an $this->name!";
  }
}

class Volvo extends Car {
  public function intro() : string {
    return "Proud to be Swedish! I'm a $this->name!";
  }
}

class Citroen extends Car {
  public function intro() : string {
    return "French extravagance! I'm a $this->name!";
  }
}

// Create objects from the child classes
$audi = new audi("Audi");
echo $audi->intro();
echo "<br>";

$volvo = new volvo("Volvo");
echo $volvo->intro();
echo "<br>";

$citroen = new citroen("Citroen");
echo $citroen->intro();


?>
```

(B10) What is Final Keyword in Object Oriented Programming?

->

The final keyword is used to prevent a class from being inherited and to prevent inherited method from being overridden.

Example:

Prevent inheritance of a class using the final keyword:

```
<?php

final class MyClass
{

  public $name = "John";

}

class AnotherClass extends MyClass{};

?>
```

(B11) What is Pure Virtual function in Object Oriented Programming?

-> A pure virtual function or pure virtual method is a virtual function that is required to be implemented by a derived class if the derived class is not abstract.

Classes containing pure virtual methods are termed "abstract" and they cannot be instantiated directly.

A virtual function is a member function of base class which can be redefined by derived class.

A pure virtual function is a member function of base class whose only declaration is provided in base class and should be defined in derived class otherwise derived class also becomes abstract.

(B12) what are Sealed Modifier in Object Oriented Programming?

-> When applied to a class, the sealed modifier prevents other classes from inheriting from it.

In the following example, class B inherits from class A, but no class can inherit from class B.

You can also use the sealed modifier on a method or property that overrides a virtual method or property in a base class.

(B13) What is Dynamic or run time Polymorphism in oops?

-> Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. Method overriding by a subclass is termed as runtime polymorphism. JVM determines the method to be executed at runtime instead of compile time.

Example:

```
class SuperClass {
  SuperClass get(){
    System.out.println("SuperClass");
    return this;
  }
}
public class Tester extends SuperClass {
  Tester get(){
    System.out.println("SubClass");
    return this;
  }
  public static void main(String[] args) {
    SuperClass tester = new Tester();

    tester.get();
  }
}
```

(B14)What is Access Modifier in Object Oriented Programming?

->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

In the following example we have added three different access modifiers to three properties (name, color, and weight).

Here, if you try to set the name property it will work fine (because the name property is public, and can be accessed from everywhere).

However, if you try to set the color or weight property it will result in a fatal error (because the color and weight property are protected and private):

Example:

```
<? Php

class Fruit
 {

  public $name;
  protected $color;
  private $weight;

}

$mango = new Fruit();
$mango->name = 'Mango';
$mango->color = 'Yellow';
$mango->weight = '300';
?>
```

(B15)What is Friend Function in Object Oriented Programming?

-> In object-oriented programming, a **friend function**, that is a "friend" of a given class, is a function that is given the same access as methods to private and protected data.

Friend function is **best shared among a number of different classes**. Such functions can be declared either as member functions of one class or as global functions. In either case they can be set to be friends of other classes, by using a friend specifier in the class that is admitting them

A friend function is declared by the class that is granting access, so friend functions are part of the class interface, like methods. Friend functions allow alternative syntax to use objects, for instance. Friend functions have the same implications on encapsulation as methods.A similar concept is that of friend class.

This approach may be used in friendly function when a function needs to access private data in objects from two different classes. This may be accomplished in two similar ways

- a function of global or namespace scope may be declared as friend of both classes
- a member function of one class may be declared as friend of another one.

(B16) What is Overriding in Object Oriented Programming?

-> Method overriding, in object-oriented programming, is a language feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its superclasses or parent classes.

In addition to providing data-driven algorithm-determined parameters across virtual network interfaces, it also allows for a specific type of polymorphism.

The implementation in the subclass overrides the implementation in the superclass by providing a method that has same name, same parameters or signature, and same return type as the method in the parent class.

The version of a method that is executed will be determined by the object that is used to invoke it.

If an object of a parent class is used to invoke the method, then the version in the parent class will be executed,

but if an object of the subclass is used to invoke the method, then the version in the child class will be executed.

This helps in preventing problems associated with differential relay aanalytics which would otherwise rely on a framework in which method overriding might be obviated.

Some languages allow a programmer to prevent a method from being overridden.

(B17) What is the role of mutable storage class specifier?

-> The mutable storage class specifier is used only on a class data member to make it modifiable even though the member is part of an object declared as const. You cannot use the mutable specifier with names declared as static or const, or reference members.

In the following example:

```
class A
{
  public:
    A() : x(4), y(5) { };
    mutable int x;
    int y;
};

int main()
{
  const A var2;
  var2.x = 345;
  // var2.y = 2345;

}
```

The compiler would not allow the assignment var2.y = 2345 because var2 has been declared as const.

The compiler will allow the assignment var2.x = 345 because A::x has been declared as mutable.

(B18) Distinguish between shallow copy and deep copy?

->

Shallow Copy	Deep Copy
Shallow Copy stores the references of objects to the original memory address.	Deep copy stores copies of the object's value.
Shallow Copy reflects changes made to the new/copied object in the original object.	Deep copy doesn't reflect changes made to the new/copied object in the original object.
Shallow Copy stores the copy of the original object and points the references to the objects.	Deep copy stores the copy of the original object and recursively copies the objects as well.
Shallow copy is faster.	Deep copy is comparatively slower.

(B19) what is a Refrence variable in Object oriented Programming Language?

->

One of the key-points of PHP OOP that is often mentioned is that "objects are passed by references by default".

This is not completely true. This section rectifies that general thought using some examples.

A PHP reference is an alias, which allows two different variables to write to the same value.

In PHP, an object variable doesn't contain the object itself as value.

It only contains an object identifier which allows object accesses to find the actual object.

When an object is sent by argument, returned or assigned to another variable, the different variables are not aliases: they hold a copy of the identifier, which points to the same object.

Example:

```
<?php

class A

 {
```

```
    public $foo = 1;

}

$a = new A;
$b = $a;
$b->foo = 2;
echo $a->foo."\n";


$c = new A;
$d = &$c;

$d->foo = 2;
echo $c->foo."\n";


$e = new A;

function foo($obj)
 {


    $obj->foo = 2;
}

foo($e);
echo $e->foo."\n";


?>
```

(B20) What is a Copy Constructor?

->

A copy constructor is a special type of constructor which initializes all the data members of the newly created object by copying the contents of an existing object.

The compiler provides a default copy constructor.

The syntax of calling a copy constructor is as shown below:

```
class_name new_object (existing object);
```

Assume that we have a class Period and object p1 of this class has already been defined.

Then we can create new object p2 belonging to same class as

```
Period p2 (p1);
```

(B21)What is this Pointer in Object oriented Programming Language?

->

$this is a special variable in Object Oriented programming in PHP, which is reference to current object.is used to access an object member in PHP Place the value of variable $entryId into the entryId field of this object.

Example:

```
$b=2;
$a=$b;
$a=3;
Print $a;
Print $b;
```

OR

```
$b=2;
$a=&$b;
$a=3;
print $a;
print $b;
```

In the above code, because we used & operator, a reference to where $b is pointing is stored in $ a. So $ a is actually a reference to $b.

In PHP, arguments are passed by value by default. So when calling a function, when you pass in your values, they are copied by value not by reference.

[I1] Define Constructor and Destructor?

->

Constructors :

Constructors are the blueprints for object creation providing values for member functions and member variables. Once the object is initialized, the constructor is automatically called.

Destructors :

Destructors are for destroying objects and automatically called at the end of execution. In this article, we are going to learn about object-oriented concepts of constructors and destructors.

Syntax:

- __construct():

```
function __construct()
    {
    // initialize the object and its properties by assigning
    }
```

- __destruct():

```
function __destruct()
    {
    // destroying the object or clean up resources here
    }
```

[I2] How to Load Classes in Object Oriented Programming?

->

PHP load classes are used for declaring its object etc. in object oriented applications.

PHP parser loads it automatically, if it is registered With spl_autoload_register() function.

PHP parser gets the least chance to load class/interface before emitting an error.

Before using a class, you need to:

- **First, define the class in a file.**
- **Second, load it using the require, require_once, include, or include_once statement.**

Syntax:

```
spl_autoload_register(function ($class_name)
 {
  include $class_name . '.php';
});
```

Example:

```
<?php

class Contact

{
        private $email;

        public function __construct(string $email)

        {
                $this->email = $email;
        }

        public function getEmail()

        {
                return $this->email;
        }
}
```

[I3] How to Call Parent 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.

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
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();
?>
```

[I4] Are Parent Constructor Called Implicitly When Create An Object Of Class?

->

Parent constructors are not called implicitly 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.

[I5] What Happen, If Constructor Is Defined As Private Or Protected?

-> **The constructor may be made private or protected to prevent it from being called externally.**

If so, only a static method will be able to instantiate the class.

Because they are in the same class definition they have access to private methods, even if not of the same object instance.

Public, private and protected are called access modifiers.

Just like C++, PHP also have three access modifiers such as public, private and protected.

The visibility of a property, a method or a constant can be defined by prefixing the declaration with these keywords.

- **If the class member declared as public then it can be accessed everywhere.**

- **If the class members declared as protected then it can be accessed only within the class itself and by inheriting child classes.**

- **If the class members declared as private then it may only be accessed by the class that defines the member.**

[I6] Define New-style Constructor & Old-style Constructor. Check which One Will Be Called?

-> Could someone tell me how the "old-style" object constructor is different from the "new-style" constructor? I'm learning PHP OOP, and I want to know when I'm reading old syntax vs new syntax, and better understand how OOP has changed in PHP over time.

New Style

```
class aObject
{
    public $name;
```

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

Old (PHP4)

```
<?php

class MyOldClass
{
    var $foo;

    function MyOldClass($foo)
    {
        $this->foo = $foo;
    }

    function notAConstructor()
}
```

New (PHP5+)

```
<?php

class MyNewClass
{
    var $foo;

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

    public function notAConstructor()
    {
        /* ... */
    }
}
```

[I7] Create Abstract class and method?

-> Abstract classes and methods are when the parent class has a named method, but need its child class 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;
}
?>
```

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

[I8] Define 3 types of visibility of data member & member function.

-> Visibility of a class member (property, method or constant) is where it can be accessed. (Eg: inside the class, outside the class)

Properties, methods, and constants can be declared with visibility.

1. To validate and restrict data

2. To keep private things private

Types of Visibility

1. **Public** –

   Can be accessed from everywhere

2. **Private** –

   Can only be accessed within the class

3. **Protected** –

   Can be accessed by the class declared it and by the classes that inherit the above declared class.

[I9] Create a method which will never inherited?

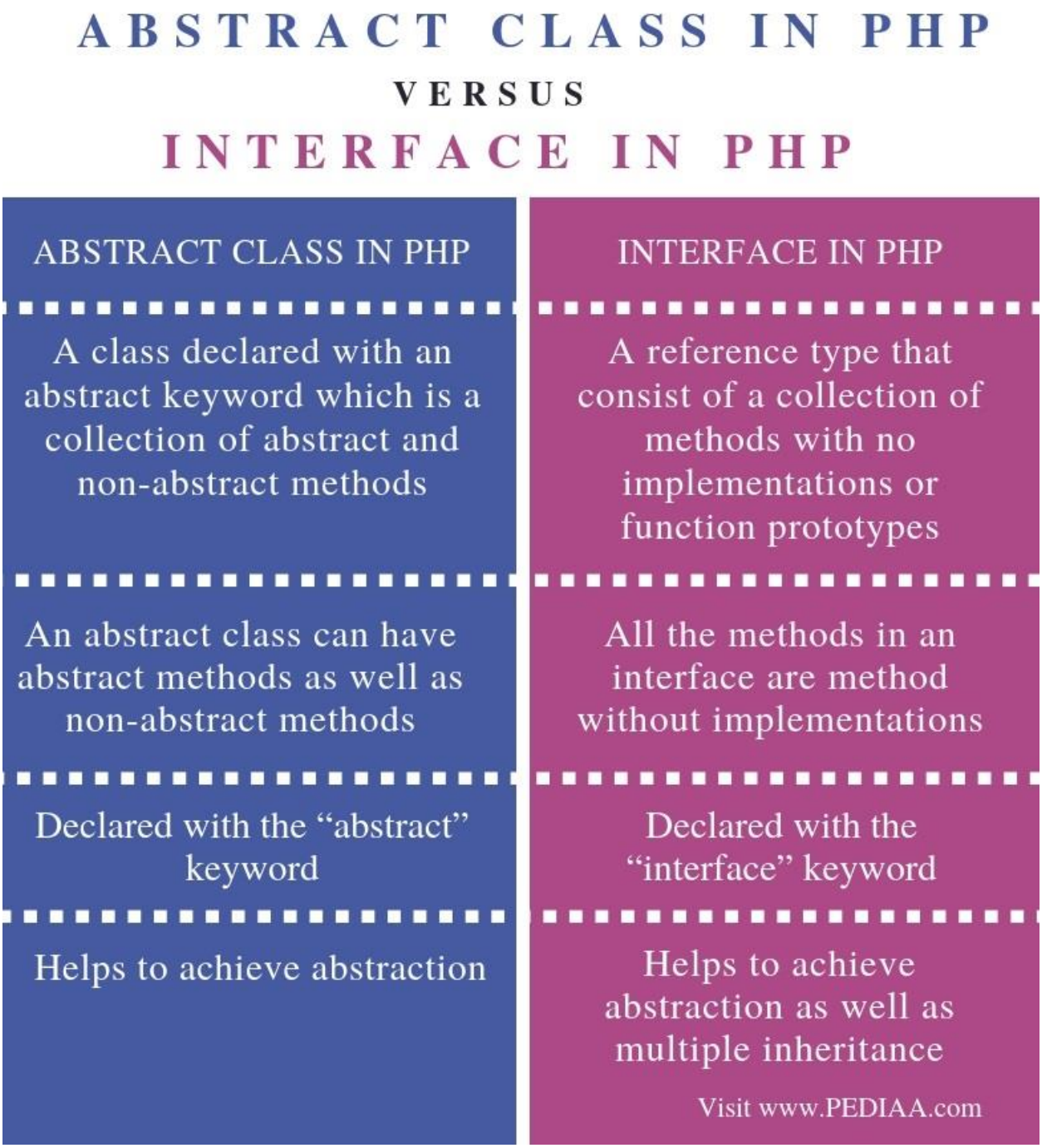
-> Inheritance in OOP = When a class derives from another class. The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods. An inherited class is defined by using the extends keyword.

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}.";
  }
}
class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}
$strawberry = new Strawberry("Strawberry", "red");
$strawberry->message();
$strawberry->intro();
?>
```

[I10] What is the difference between Abstract class and Interface?

->



[I11] Create parent class for car and child class for car_model and use car functionality in car_model class

->

```
//The parent class
class Car {

  private $model;


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

  public function hello()
  {
    return "beep! I am a <i>" . $this -> model . "</i><br />";
  }
}

class SportsCar extends Car {

}


$sportsCar1 = new SportsCar();


$sportsCar1 -> setModel('Mercedes Benz');

echo $sportsCar1 -> hello();

}
```

[I12] Override the parent's properties and methods in the child class?

->In order to prevent the method in the child class from overriding the parent's methods, we can prefix the method in the parent with the **final** keyword.

In the example given below, we declare the hello() method in the parent as final, but still try to override it in the child class.

In the previous chapter we learned that protected properties or methods can be accessed within the class and by classes derived from that class.

[I13] Prevent the child class from overriding the parent's methods?

->

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();
?>
```

[I15] Declare classes and methods as abstract?

-> When can define a class abstract using the abstract keyword. An class which is defined as abstract cannot be instantiated.

Following are some important points about abstract class and method:

1. An abstract class can have methods and properties just like any other normal class.
2. An abstract class cannot be instantiated, hence we need to create a child class which extends it, then we can create object of the child class.
3. If a class has even a single abstract method then the class should also be abstract.
4. An abstract method is just the declaration, where we provide name of the method and argument, while the body part is empty.

Example:

```
<?php
   abstract class Vehicle {
      protected $name;
       public function start() {
```

```
        echo $this->name. " - Engine start...<br/>";
    }
        public function stop() {
        echo $this->name. " - Engine stop...<br/>";
    }
      public function setName($name) {
        $this->name = $name;
    }
       abstract public function mileage() {


    }
 }
?>
```

[I16]Can we have non abstract methods inside an abstract class? Explain With Example?

- An abstract class means that hiding the implementation and showing the function definition to the user.
- An abstract class having both abstract methods and non-abstract methods.

- **For an abstract class, we are not able to create an object directly. But Indirectly we can create an object using the subclass object.**
- **A Java abstract class can have instance methods that implement a default behavior.**
- **An abstract class can extend only one class or one abstract class at a time.**
- **Declaring a class as abstract with no abstract methods means that we don't allow it to be instantiated on its own.**
- **An abstract class used in Java signifies that we can't create an object of the class directly.**

Example:

```
abstract class Car {

  protected $tankVolume;

  public function setTankVolume($volume)
  {
    $this -> tankVolume = $volume;
  }


  abstract public function calcNumMilesOnFullTank();
}
```

[I17] How to create child classes from an abstract class?

->

Since we cannot create objects from abstract classes, we need to create child classes that inherit the abstract class code.

Child classes of abstract classes are formed with the help of the extends keyword, like any other child class.

They are different in that they have to add the bodies to the abstract methods.

> *The child classes that inherit from abstract classes must add bodies to the abstract methods.*

Example:

```
class Honda extends Car

 {

  public function calcNumMilesOnFullTank()

  {
    $miles = $this -> tankVolume*30;
    return $miles;
  }
}
```

[I18] What are Magic Methods/Functions? List them in OOPS?

-> Magic methods are special methods which override PHP's default's action when certain actions are performed on an object.

The following method names are considered magical:

__construct(),

__destruct(),

__call(),

__callStatic(),

__get(),

__set(),

__isset(),

__unset(),

__sleep(),

__wakeup(),

__serialize(),

__unserialize(),

__toString(),

__invoke(),

__set_state(),

__clone(),

__debugInfo().

[I19] How can we used Virtual Function write an examples in oops?

->

A virtual function is a member function that you expect to be redefined in derived classes.

When you refer to a derived class object using a pointer or a reference to the base class.

You can call a virtual function for that object and execute the derived class's version of the function.

Virtual functions ensure that the correct function is called for an object.

Regardless of the expression used to make the function call.

Suppose a base class contains a function declared as virtual and a derived class defines the same function.

The function from the derived class is invoked for objects of the derived class.

Even if it is called using a pointer or reference to the base class.

The following example shows a base class that provides an implementation of the Print Balance function and two derived classes

[I20] How can we Used various type of Constructor write an Examples in oops?

->

- **Default Constructor:**
  It has no parameters, but the values to the default constructor can be passed dynamically.

- **Parameterized Constructor:**
  It takes the parameters, and also you can pass different values to the data members.

- **Copy Constructor:**
  It accepts the address of the other objects as a parameter.

Example:

```
<?php
class ParentClass{
        function __construct(){
                print "Parent class constructor.\n";
        }
   }
class ChildClass extends Parentclass{
        function __construct(){
                parent::__construct();
                print "Child Class constructor";
        }
   }
$obj = new ParentClass();
$obj = new ChildClass();

?>
```

[I21] Which Constructor have no Parameter write an Examples in oops?

-> Default Constructor:

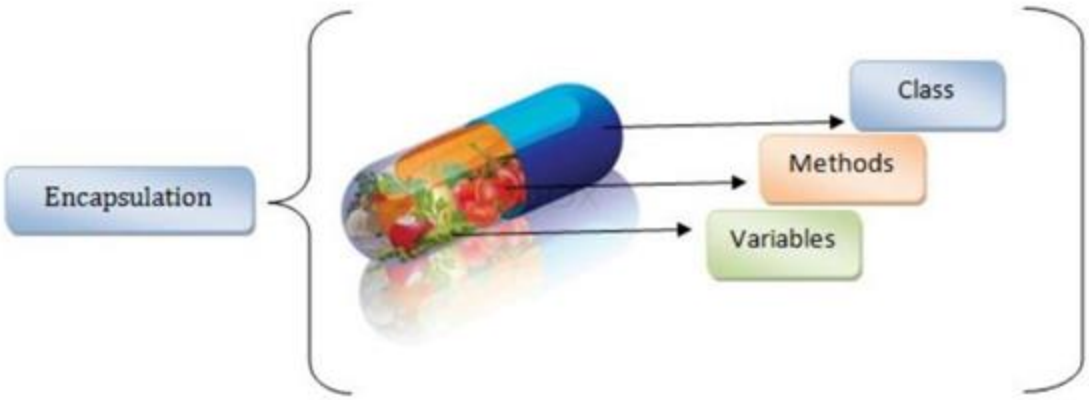
It has no parameters, but the values to the default constructor can be passed dynamically.

Example:

```
<?php

class Tree{

   function Tree()
{
      echo "It’s a User-defined Constructor of the class Tree";
}
   function __construct()
 {
      echo "Its a Pre-defined Constructor of the class Tree";
   }
}

$obj= new Tree();

?>
```

[I22] How to Secured Internal Data using Encapsulation Write a Example in oops?

->



- Encapsulation is a concept where we encapsulate all the data and member functions together to form an object.
- Wrapping up data member and method together into a single unit is called Encapsulation.
- Encapsulation also allows a class to change its internal implementation without hurting the overall functioning of the system.
- Binding the data with the code that manipulates it.
- It keeps the data and the code safe from external interference.

Example:

```
<?php
class person
{
public $name;
public $age;
function __construct($n, $a)
{
$this->name=$n;
$this->age=$a;
}
public function setAge($ag)
    {
$this->ag=$ag;
}
public function display()
{
echo  "welcome ".$this->name."<br/>";
return $this->age-$this->ag;
}
}
$person=new person("sonoo",28);
$person->setAge(1);
echo "You are ".$person->display()." years old";

?>
```

(A1) Write a Programmed to create a Class in OOPS?

-> 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 {
  // code...
}
?>
```

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;
  }
}
?>
```

(A2) Write a Programme to Create a Object in OOPS?

-> 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 is 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();
?>
```

(A3) Write a Programme to Create an Abstract Class In OOPS?

-> Abstract classes and methods are when the parent class has a named method, but need its child class 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;

}
?>
```

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

Example:

```
<?php

// Parent class
abstract class Car

 {
  public $name;
  public function __construct($name)

{
    $this->name = $name;
  }
  abstract public function intro() : string;
}

class Audi extends Car

 {
  public function intro() : string {
```

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

class Volvo extends Car {
  public function intro() : string {
    return "Proud to be Swedish! I'm a $this->name!";
  }
}

class Citroen extends Car {
  public function intro() : string {
    return "French extravagance! I'm a $this->name!";
  }
}

$audi = new audi("Audi");
echo $audi->intro();
echo "<br>";

$volvo = new volvo("Volvo");
echo $volvo->intro();
echo "<br>";

$citroen = new citroen("Citroen");
echo $citroen->intro();


?>
```

(A4)Write a Programme to Create a Encapsulation in OOPS?

-> Encapsulation is just wrapping some data in an object. The term "encapsulation" is often used interchangeably with "information hiding".

Example:

```
<?php

class App {
    private static $_user;
    public function User( ) {
        if( $this->_user == null ) {
            $this->_user = new User();
        }
        return $this->_user;
    }
}
class User {
    private $_name;
    public function __construct() {
        $this->_name = "Joseph Crawford Jr.";
    }
    public function GetName() {
        return $this->_name;
    }
}
$app = new App();
echo $app->User()->GetName();

?>
```

(A5) How to Implement Access Modifier Write a Programmed in OOPS?

-> 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

In the following example we have added three different access modifiers to three properties (name, color, and weight).

Here, if you try to set the name property it will work fine (because the name property is public, and can be accessed from everywhere).

However, if you try to set the color or weight property it will result in a fatal error (because the color and weight property are protected and private):

Example:

```
<? Php

class Fruit
 {

  public $name;
  protected $color;
  private $weight;

}

$mango = new Fruit();
$mango->name = 'Mango';
$mango->color = 'Yellow';
$mango->weight = '300';
?>
```

(A6) Write a Programmed of Copy Constructor in OOPS?

->

```
<?PHP
class Tree
{
    function Tree()
    {
        echo "Its a User-defined Constructor of the class Tree";
    }

    function __construct()
    {
        echo "Its a Pre-defined Constructor of the class Tree";
    }
}

$obj= new Tree();
?>
```

(A7) Declare and implement an interface and implement more than one interface in the same class.

->

Example:

```
<?php
interface Inf1
{
   public function fun1();
}
interface Inf2
{
   public function fun2();
}
class Sample implements Inf1, Inf2
{
   public function fun1()
   {
      printf("fun1() called<br>");
   }
   public function fun2()   {
```

```
        printf("fun2() called<br>");
    }
}
$obj = new Sample();
$obj->fun1();
$obj->fun2();
?>
```

Output:

Fun1() called

Fun2() called

Explanation:

In the above program, we created two interfaces *Inf1* and *Inf2*.

The *Inf1* interface contains the declaration of *fun1()* and the *Inf2* interface contains the declaration of *fun2()*.

After that, we created a class *Sample*.

Here, we used the ***implements*** keyword to implement an interface into the class. Then we implemented both interfaces in the *Sample* class.

At last, we created an object *$obj* of *Sample* class and called functions that will print the appropriate message on the webpage

(A8) How to implement the polymorphism principle in OOPS?

->

Abstract classes in PHP are the parent classes that have methods defined but not the code. They are defined with the help of the “abstract” keyword. You can only use them to inherit other classes. Every abstract class has one or more abstract methods. These classes require their child classes to fill the methods with the same name. Hence, different child classes use the methods, in the same way, thereby implementing polymorphism in PHP.

Example:

```
<?php

abstract class shapeExmp
{
abstract protected function calcArea();
}
class rectangleExmp extends shapeExmp
{
var $x,$y;
public function __construct($x , $y){
$this->x=$x;
$this->y=$y;
}
public function calcArea(){
$a=$this->x*$this->y;
return $a;
}
}
class circleExmp extends shapeExmp{
var $r;
public function __construct($r){
$this->r=$r;
```

```
    }
public function calcArea(){
$pi=3.142;
$a=pow($this->r,2)*$pi;
return $a;
    }
}
class squareExmp extends shapeExmp{
var $s;
public function __construct($s)
{
$this->s=$s;
    }
public function calcArea(){
$a=$this->s * $this->s;
return $a;
    }
}
$rect=new rectangleExmp(8,10);
$area=$rect->calcArea();

echo “Rectangle area= $area <br>”;
$circ=new circleExmp(5);
$area=$circ->calcArea();

echo “Cirlce area=$area<br>”;
$squ=new squareExmp(7);
$area=$squ->calcArea();

echo “Square area= $area <br>”;

?>
```

Output:

```
Rectangle area= 80
Circle area=78.55
Square area=49
```

(A9) Explain Scope resolution operator with example in OOPS?

-> The scope resolution operator also known as *Paamayim Nekudotayim* or more commonly known as the double colon is a token that allows access to static, constant, and overridden properties or methods of a class.

It is used to refer to blocks or codes in context to classes, objects, etc. An identifier is used with the scope resolution operator.

The most common example of the application of the scope resolution operator in PHP is to access the properties and methods of the class.

Example:

```
<?php
class democlass {
        const PI = 3.14;
}
echo democlass::PI;
?>
```

(A10) How to Access child class property to Parent class write an Programme in OOPS?

->

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.

In the previous chapter we learned that protected properties or methods can be accessed within the class and by classes derived from that class.

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

As your Child class is extending your Parent class, every properties and methods that are either public or protected in the Parent class will be seen by the Child class as if they were defined in the Child class and the other way arround.

When the Child class extends the Parent class, it can be seen as *"Child is a Parent"* which means the Child has the properties of the Parent, unless it redefines those another way.

The __construct() and intro() methods in the child class will override the __construct() and intro() methods in the parent class

(A11) Write a Programme of how to define Interface in OOPS?

->

Example:

```php
<?php
interface Animal {
  public function makeSound();
}
class Cat implements Animal {
  public function makeSound() {
    echo " Meow ";
  }
}
class Dog implements Animal {
  public function makeSound() {
    echo " Bark ";
  }
}
class Mouse implements Animal {
  public function makeSound() {
    echo " Squeak ";
  }
}
$cat = new Cat();
$dog = new Dog();
$mouse = new Mouse();
$animals = array($cat, $dog, $mouse);
foreach($animals as $animal) {
  $animal->makeSound();
}
?>
```

(A12) Write a Programme of how to define a Constructor in OOPS?

->

Constructors are special member functions for initial settings of newly created object instances from a class, which is the key part of the object-oriented.

Constructors are the very basic building blocks that define the future object and its nature.

That the Constructors are the blueprints for object creation providing values for member functions and member variables.

Once the object is initialized, the constructor is automatically called. Destructors are for destroying objects and automatically called at the end of execution.

In this article, we are going to learn about object-oriented concepts of constructors and destructors.

Example:

```
<?php

class Employee
{
```

```
        Public $name;
        Public $position;

        function __construct($name,$position)
        {
                $this->name=$name;
                $this->profile=$position;
        }

        function show_details()
        {
                echo $this->name." : ";
                echo "Your position is ".$this->profile."\n";
        }
}

$employee_obj= new Employee("Rakesh","developer");
$employee_obj->show_details();

$employee2= new Employee("Vikas","Manager");
$employee2->show_details();

?>
```

(A13) How to define a default constructor write a programed in OOPS?

->

Default Constructor:

It has no parameters, but the values to the default constructor can be passed dynamically.

Pre-defined Default Constructor:

By using function __construct (), you can define a constructor.

Example:

```
<?php

class Tree
{
   function Tree()
 {
       echo "It’s a User-defined Constructor of the class Tree";
   }
   function __construct()
 {
       echo "Its a Pre-defined Constructor of the class Tree";
   }
}
$obj= new Tree();

?>
```