UNIT-3
OBJECT ORIENTED CONCEPTS IN PHP

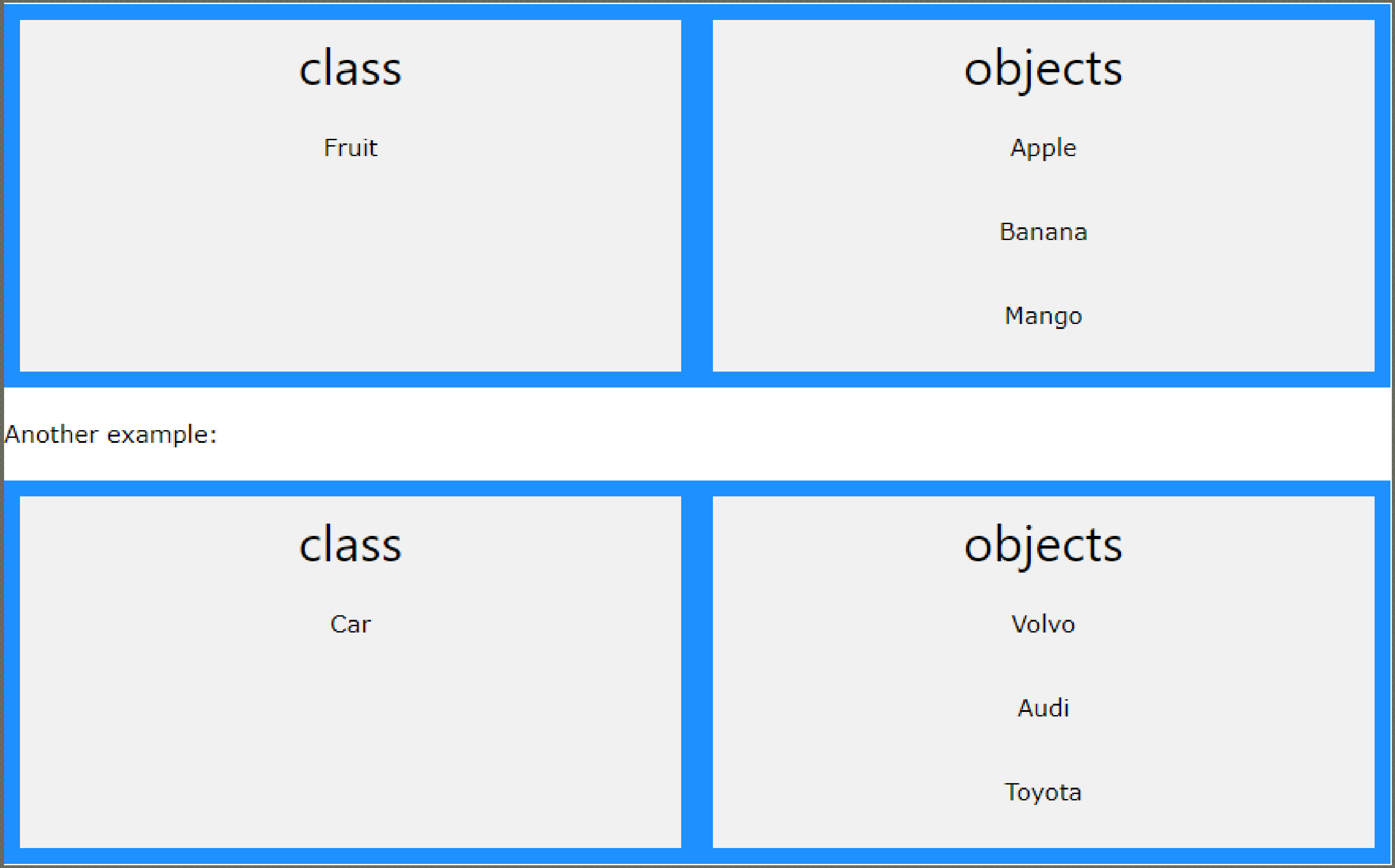
OOP

- OOP stands for Object-Oriented Programming.
- Procedural programming is about writing procedures or functions that perform operations on the data
- Object-oriented programming is about creating objects that contain both data and functions.

OOP (CNTD…)

- Object-oriented programming has several advantages over procedural programming:
  - OOP is faster and easier to execute
  - OOP provides a clear structure for the programs
  - OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
  - OOP makes it possible to create full reusable applications with less code and shorter development time

Classes and Objects



class	objects
Fruit	Apple Banana Mango

Another example:

class	objects
Car	Volvo Audi Toyota

Object Oriented Concepts

- **Class:**
  - Collection of local functions as well as local data.
  - You can think of a class as a template for making many instances of the same kind (or class) of object.
- **Object:**
  - An instance of the class.
  - You define a class once and then make many objects that belong to it.
- **Member Variable:**
  - These are the variables defined inside a class.
  - This data will be invisible to the outside of the class.
  - It can be accessed via member functions.
  - These variables are called attribute of the object once an object is created.
- **Member function:**
  - These are the function defined inside a class and are used to access object data.

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 {}.
- Syntax:

```
<?php

  class Fruit
  {
     // code goes here...
  }

  ?>
```

Define Objects

- Object is an instance of class.
- Objects of a class are created using the new keyword.
- 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.
- `<?php   $variable = new Classname(); ?>`

Example

```
<?php
class Fruit
 {
  public $name;

  // Methods
  function set_name($name)
 {
    $this->name = $name;
 }

  function get_name()
 {
    return $this->name;
 }
}
```

```
$a = new Fruit();
$a->set_name('Apple');

$b = new Fruit();
$b->set_name('Banana');

echo $a->get_name();
echo "<br>";

echo $b->get_name();
?>
```

The $this Keyword

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

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

$a = new Fruit();
$a->set_name("Apple");

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

Constructor

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 (__).

Example Constructor

```
<?php

class Fruit
{
     public $name;

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

    function get_name()
   {
      return $this->name;
    }
}
```

```
$a= new Fruit("Apple");

echo $apple->get_name();

?>
```

Destructor

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.
- Notice that the destruct function starts with two underscores (__).

Example Destructor

```
<?php

  class Fruit
  {
    public $name;

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

  function __destruct()
  {
      echo "The fruit is {$this-
  >name}.";
   }
  }

    $apple = new Fruit("Apple");

?>
```

Access Modifiers

- Properties and methods can have access modifiers which control where they can be accessed.

Modifier Name	Description
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

Inheritance

- The process of acquiring the all properties of parent class into child class is known as inheritance.
- 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.

Inheritance

```
<?php

class Base
{
   Public function intro()
    {
    echo "This is Base Clasee<br><br>";
    }
}
class Derived extends Base
{
    public function message()
    {
   echo "This is Derived class<br><br>";
    }
}
```

```
$d1 = new Derived;
$d1->message();
$d1->intro();
?>
```

Inheritance

```
<?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 "This is message function";
  }
}
```

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

Inheritance

- Inherited constructor and methods can be overridden by redefining the methods (use the same name) in the child class.
- Example:

Inheritance

```
<?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 $weight;
  public function __construct($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();
?>
```

The final Keyword

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

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

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

The final Keyword

```
<?php

class Fruit {
  final public function intro() {
    // some code
  }
}

class Strawberry extends Fruit {
  // will result in error
  public function intro() {
    // some code
  }
}
?>
```

Static Keyword

- The static keyword is used to declare properties and methods of a class as static. Static properties and methods can be used without creating an instance of the class.
- The static keyword is also used to declare variables in a function which keep their value after the function has ended.

Static Keyword

- Static properties can be called directly - without creating an instance of a class.
- Static properties are declared with the static keyword:
- Syntax:

```
<?php
  class ClassName {
    public static $variable = “Value";
  }
  ?>
```

Static Keyword

- To access a static property use the class name, double colon (::), and the property name:
- Syntax:

```
ClassName::$Variable;
```

Static Keyword

- Example 1:

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

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

Static Keyword

Example 1:

```
<?php
class student
{
      public static $number = 0;
      public function add1() {
      self::$number++;
      return self::$number;
}
}

$s2=new student();
echo $s2->add1();
?>
```

Static Keyword

- Static methods are declared with the static keyword.
- Syntax:
-
```
<?php
class ClassName {
  public static function staticMethod() {
    echo "Hello World!";
  }
}
?>
```

Static Keyword

- To access a static method use the class name, double colon (::), and the method name.
- Syntax:

```
ClassName::staticMethod();
```

Static Keyword

- Example:

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

  // Call static method
   Student::welcome();
  ?>
```

Static Keyword

- 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 (::).

Static Keyword

- Example:

```
<?php
class Student {
 public function __construct() {
    self::welcome();
  }
  public static function welcome() {
    echo "Hello World!";
  }

}

new Student ();
?>
```

Static Keyword

- Example 2

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

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

Polymorphism

- To begin with, Polymorphism is gotten from the Greek words Poly (which means many) and morphism (which meaning forms).

Method overloading

In function overloading, the class have the same function name with and number of arguments

```
<?php
class Machine
{
function doTask($var1)
{
        return $var1;
}
function DoTask($var1,$var2)
{
      return $var1 * $var1 ;
 }
}
 $task1 = new machine();
$task1->doTask(5,10);
?>
```

Method Overriding

- In function overriding, the parent and child classes have the same function name with and number of arguments

Method Overriding

Example:

```
<?php
class Base
{
function demo()
{
echo “Base class function!";
}
 }
class Derived extends Base
{
function demo()
{
echo “Derived class function!";
}
}

$ob = new Base;
 $ob->demo();
 $ob2 = new Derived;
$ob2->demo();
 ?>
```

Abstract Classes

- 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:

Abstract Classes

- Syntax:

```
<?php
  abstract class ParentClass
  {
   abstract public function someMethod1();
   abstract public function someMethod2($
  name, $color);
  }
  ?>
```

Abstract Classes

- 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.

Abstract Classes

- 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

Abstract Classes

- Example:

```
<?php

// Parent class
abstract class Car
{
  abstract public function intro();
}

// Child classes
class Audi extends Car
{
       public function intro()
       {
                   return "This is AUDI";

      }
}

class Volvo extends Car
{
      public function intro()
      {
                   return "This is VOLVO";
      }
}
class Citroen extends Car
{
      public function intro()
      {
                   return "This is CITRON";
      }
}

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

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

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

Interfaces

- 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:

Interfaces

- Syntax:
- <?php

```
        interface InterfaceName
         {
        public function someMethod1();
        public function someMethod2($name, $color);
        }
?>
```

Interfaces

- Example:

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

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

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

?>
```

Interfaces

- Example – 2
```
<?php
// Interface definition
interface Animal
{
  public function makeSound();
}

// Class definitions
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 ";
              }
  }

        // Create a list of animals
        $cat = new Cat();
        $dog = new Dog();
        $mouse = new Mouse();
        $animals = array($cat, $dog, $mouse);

        // Tell the animals to make a sound
        foreach($animals as $animal) {
          $animal->makeSound();
        }
        ?>
```

clone

- The clone keyword is used to create a copy of an object.
- If any of the properties was a reference to another variable or object, then only the reference is copied.
- Objects are always passed by reference, so if the original object has another object in its properties, the copy will point to the same object.
- This behavior can be changed by creating a __clone() method in the class.