Object-Oriented Programming-PHP

Object Oriented Concepts:

Before we go in detail, lets define important terms related to Object Oriented Programming.

- **Class:** This is a programmer-defined datatype, which includes 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 individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.
- **Member Variable:** These are the variables defined inside a class. This data will be invisible to the outside of the class and 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.
- **Inheritance:** When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.
- **Parent class:** A class that is inherited from by another class. This is also called a base class or super class.
- **Child Class:** A class that inherits from another class. This is also called a subclass or derived class.
- **Polymorphism:** This is an object oriented concept where same function can be used for different purposes. For example function name will remain same but it make take different number of arguments and can do different task.
- **Overloading:** a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly functions can also be overloaded with different implementation.
- **Data Abstraction:** Any representation of data in which the implementation details are hidden (abstracted).
- **Encapsulation:** refers to a concept where we encapsulate all the data and member functions together to form an object.
- **Constructor:** refers to a special type of function which will be called automatically whenever there is an object formation from a class.
- **Destructors:** refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

Defining PHP Classes:

Everything oop starts with classes. Classes are the type of objects like any other variable. The object are the instance of the class.

The general form for defining a new class in PHP is as follows:

```
<?php
Class Person
{
var $var1;
var $var2 = "constant string";
function myfunc ($arg1, $arg2)
{
[..]
}
[..]
}
?>
```

Declaration of class

Declaration of members

Declaration of methods

Here is the description of each line:

- The keyword **class**, followed by the name of the class that you want to define.
- A set of braces enclosing any number of variable declarations and function definitions.
- Variable declarations start with the special form **var**, which is followed by a conventional $ variable name; they may also have an initial assignment to a constant value.
- Function definitions look much like standalone PHP functions but are local to the class and will be used to set and access object data.

We will see the step by step process of creating a class

class Author // create with keyword class
{
}

class Author
{
var $name; // add variables if needed
.
.
.
}

```
class Author
{
var $name;
function      set_author_name($data)      //      Add      methods      if      needed
                                   //like this
{
.
.
}
.
.

}
```

Now to allocate the value $data passed to the function set_author_name() do as shown in the coding to the right:

$this keyword point to the current object.

In addition, omit the **$** in front of the property referring like this,

```
$this->name
```

```
class Author
{
var $name;
function set_author_name($data)
{
$this->name = $data;        // allocate the parameter value
}
function get_author_name()
{
return $this->name;
}
.

}
```

The above line created a new Author object named **$sujatha**

Creating objects
So far we have seen how to create class. Now we will see how to create objects for the class which is just created. To create new object **new** operator is used.
$sujatha = new Author;

```
class  Author
{
var $name;
function set_author_name($data)
{
$this->name = $data;
}
function get_author_name()
{
return $this->name;
}
}
$sujatha = new Author;
```

Create objects

All the methods and properties in the Author class are built into the **$sujatha** object. To call the $sujatha object's set_author_name method like this, using the ->operator again

```
$sujatha=new Author;
$sujatha->set_author_name("sujatha");
```

Now the phpobject.php with html coding is as shown below

```
<html>
<head>
<title>
Creating an object
</title>
</head>

<body>
<h1>Creating an object</h1>
<?php
class Author
{
var $name;
function set_author_name($data)
{
```

```
$this->name = $data;
}

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

$sujatha = new Author;
$sujatha->set_author_name("sujatha"); //

echo "The name of your friend is ", $sujatha->get_author_name(), ".";
?>
</body>
</html>
```

In the above program the value for **$name** is assigned by using the statement

$sujatha->set_author_name("sujatha");

The assigned value can be retrieved by the method get_author_name as

$sujatha->get_author_name();

This can be achieved like this also

$sujatha->name;

i.e in the echo statement can be written as

echo "The name of your friend is ", $sujatha->name, ".";

Setting access to properties and methods

In all the above examples and discussion so far seen, any code has access to the $name property of the object created. That's because by default a variable declared has public access, which means the members along with their variables (properties) are accessible from everywhere in the code. See the example below:

```
<?php
class  Author
{
var $name;
function set_author_name($data)
{
$this->name = $data;
}
function get_author_name()
{
return $this->name;
}
}
$sujatha = new Author;
$sujatha->set_author_name("sujatha");
echo "the name of your friend is ", $sujatha->name, ".";
?>
```

In the above example the object **$sujatha** is able to access the **$name** directly as it is declared as public. You can restrict access to the members of a class or object by using ***access*** modifiers.

They are:

- public : means "accessible to all"
- private: means "accessible in the same class"
- protected: means "accessible in the same class and classes derived from that class"

Public Access

Public access is the most unrestricted access to all, and it's the default. To declare explicitly use the keyword ***public*** as follows:

```
<?php
class  Author
{
public $name;   // public access for member
public function set_author_name($data) // public access for methods
{
$this->name = $data;
}
public function get_author_name() // public access for methods
{
return $this->name;
}
}
$sujatha = new Author;
$sujatha->set_author_name("sujatha");
echo "the name of your friend is ", $sujatha->name, ".";
?>
```

Now to restrict access use private and protected

Private Access

To make a class or object member private use ***private*** key word. When a member is made private it can’t be accessed outside the class or object.

```
<?php
class  Author
{
private $name;// private access for member
public function set_author_name($data)
{
$this->name = $data;
}
public function get_author_name()
{
return $this->name;
}
}
$sujatha = new Author;
$sujatha->set_author_name(“sujatha”);
echo “the name of your friend is “, $sujatha->name, “.”;
?>
```

When the above program is executed you can see the following error message:

The name of your friend is PHP Fatal error: Cannot access private property Author::$name in (path) on line …..

That means you have forced code outside the object to access the name stored in the **$sujatha** object using the **get_author_name()**accessor method, instead of directly accessing the $name property. The following code works fine as the private member is accessed through the method defined inside the class.

```
<?php
class  Author
{
private $name;
public function set_author_name($data)
{
$this->name = $data;
}
public function get_author_name()
{
return $this->name;
}
}
```

```
$sujatha = new Author;
$sujatha->set_author_name(“sujatha”);
echo “the name of your friend is “, $sujatha->get_author_name(), “.”;
?>
```

You can also make a method private as

```
<?php
class  Author
{
var $name;
function set_author_name($data)
{
$this->name = $data;
}
private function get_author_name()  //private access for method
{
return $this->name;
}
}
$sujatha = new Author;
$sujatha->set_author_name(“sujatha”);
echo “the name of your friend is “, $sujatha->get_author_name(), “.”;
?>
```

But now this code doesn’t work as you are trying to call a private method g**et_author_name()** outside the object. That means you can call it only from the code inside the object. So make changes in the above code as follows:

```
<?php
class  Author
{
var $name;
function set_author_name($data)
{
$this->name = get_author_name(); // make changes here
}
private function get_author_name()
{
return $this->name;
}
}
$sujatha = new Author;
$sujatha->set_author_name(“sujatha”);
```

```
echo “the name of your friend is “, $sujatha->get_author_name(), “.”;

?>
```

Now the program works

Constructors

A constructor is used to create as well as initialize an object at the same time. In the code

```
function set_author_name($data)
{
$this->name=$data;
}
```

We create the object using new operator and then using the above function like this we initialize the variables inside that object. But using a constructor you can initialize the object at the time of creation itself.

In PHP constructors are built with special name as __construct (double underscore followed by the keyword construct)

```
function __construct($data)
{
.
.
.
}
```

The above constructor takes an argument which can be assigned to the internal name stored in the object like this:

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

The constructor takes the data when the object is created with **new** operator. See the below example phpconstructor.php

```
<html>
<head>
<title>
Creating an object
</title>
</head>

<body>
<h1>Creating an object</h1>
<?php
class Author
{
var $name;
function __construct($data) // constructor
{
$this->name=$data;
}
function set_author_name($data)
{
$this->name = $data;
}

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

$kalki = new Person(“kalki”);//initialized through
                                   // constructor
$sujatha = new Author;            // the object is created
$sujatha->set_author_name("sujatha"); //the object is
                                       //initialized here
echo "The name of your friend is ", $kalki->get_author_name(), ".";
echo "The name of your friend is ", $sujatha->get_author_name(), ".";
?>
</body>

</html>
```

In the above program the object $kalki is created and initialized at the time of creation itself. The line in the program is given below:

$kalki = new Person(“kalki”);

In the line

$sujatha = new Author;

the object **$sujatha** is just created not initialized. It is initialized in the next statement

$sujatha->set_author_name("sujatha");

You can pass as many arguments to constructors as you need- as the constructor is set up to take those arguments.

All PHP classes come with a default constructor that takes no arguments- it's the default constructor that gets called when you execute code like this:

```
$sujatha = new Author;// constructor with default parameter – no
                      // arguments
```

$sujatha->set_author_name("sujatha");

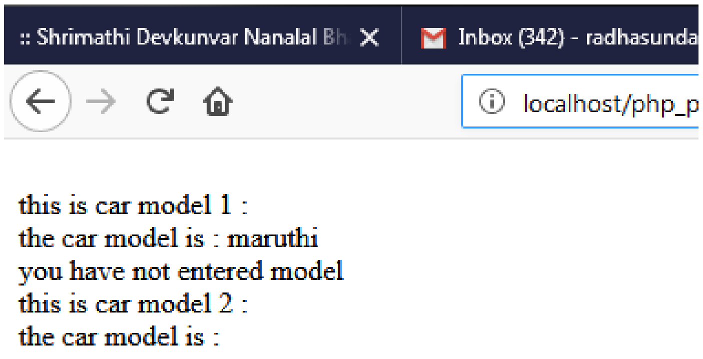
However, as soon as you create your own constructor, no matter how many arguments it takes, the default constructor is no longer accessible.

Example

```
<?php
//declare the class
class car
{
private $model;

public function __construct($model=null)
{
if($model)
{
$this->model=$model;
}
else
{
echo '<br> you have not entered model';
}
}

public function getcarmodel()
{
return '<br> the car model is : '.$this->model;
}

}
```

}

```
$car1=new car("maruthi");
echo "<br> this is car model 1 : ".$car1->getcarmodel();
$car2=new car();
echo "<br> this is car model 2 : ".$car2->getcarmodel();
?>
```



```
this is car model 1 :
the car model is : maruthi
you have not entered model
this is car model 2 :
the car model is :
```

Destructors

Besides constructors, **Destructor functions** are the opposite of constructors. They are called when the object is being destroyed (for example, when there are no more references to the object). As PHP makes sure all resources are freed at the end of each request, the importance of destructors is limited. However, they can still be useful for performing certain actions, such as flushing a resource or logging information on object destruction. There are two situations where your destructor might be called: during your script's execution when all references to an object are destroyed, or when the end of the script is reached and PHP ends the request. The latter situation is delicate because you are relying on some objects that might already have had their destructors called and are not accessible anymore. So, use it with care, and don't rely on other objects in your destructors.

Defining a destructor is as simple as adding a __destruct() method to your class:

```
Class MyClass {

function __destruct()

{

print "An object of type MyClass is being destroyed\n";

}
}

$obj = new MyClass();

$obj = NULL;
```

This script prints An object of type MyClass is being destroyed In this example, when $obj = NULL; is reached, the only handle to the object is destroyed, and therefore the destructor is called, and the object itself is destroyed. Even without the last line, the destructor would be called, but it would be at the end of the request during the execution engine's shutdown.

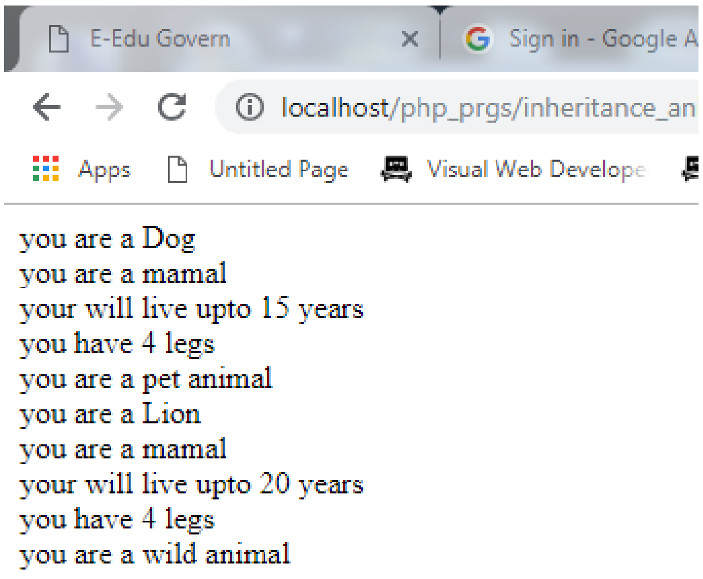
Inheritance

Inheritance is one of the most important aspects of OOP. It allows a class to inherit members from another class. From the parent class with its own methods and properties are inherited by a child class. The child class can have its own members and functions along with the public members and public functions of its parent. In the following example a class called animal is created. The members $age, $legs, $category, functions set_age_legs, get_age, get_legs are declared in animal class. These are all inherited by the child class dog. The inheritance of members and functions from parent class into child class can be done by the keyword 'extends'. In the dog (child) class the members $type, $species and functions set_type, get_type, set_species, get_species are declared. Another child class lion is also declared which inherits from animal class

```
<?php
class animal
{
var $age;
var $legs;
var $category;
function set_age_legs($age,$legs)
{
$this->age=$age;
$this->legs=$legs;
}
function get_age()
{
return $this->age;
}
function get_legs()
{
return $this->legs;
}
}

class dog extends animal
```

```
{
var $type;
var $species;
function set_type($type)
{
$this->type=$type;
}

function get_type()
{
return $this->type;
}
function set_species($species)
{
$this->species=$species;
}

function get_species()
{
return $this->species;
}
}
class lion extends animal
{
var $type;
var $species;
function set_type($type)
{
$this->type=$type;
}
function get_type()
{
return $this->type;
}
function set_species($species)
{
$this->species=$species;
}

function get_species()
{
return $this->species;
}
}
$obj1=new dog;
$obj2=new lion;
```

```
$obj1->category="pet";
$obj2->category="wild";
$obj1->set_age_legs(15,4);
$obj2->set_age_legs(20,4);
$obj1->set_species("Dog");
$obj1->set_type("mamal");
echo "you are a ". $obj1->get_species()."<br>";
echo "you are a ". $obj1->get_type()."<br>";
echo "your will live upto ". $obj1->get_age()." years <br>";
echo "you have ". $obj1->get_legs()." legs <br>";
echo "you are a ". $obj1->category." animal<br>";

$obj2->set_species("Lion");
$obj2->set_type("mamal");
echo "you are a ". $obj2->get_species()."<br>";
echo "you are a ". $obj2->get_type()."<br>";
echo "your will live upto ". $obj2->get_age()." years <br>";
echo "you have ". $obj2->get_legs()." legs <br>";
echo "you are a ". $obj2->category. " animal <br>";
?>
```



```
you are a Dog
you are a mamal
your will live upto 15 years
you have 4 legs
you are a pet animal
you are a Lion
you are a mamal
your will live upto 20 years
you have 4 legs
you are a wild animal
```

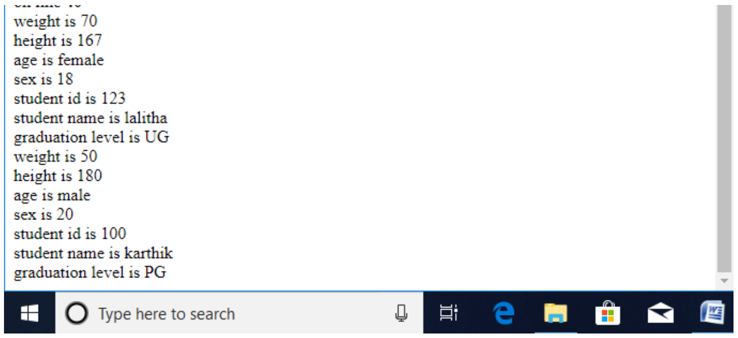
the dog is declared like a regular class. The dog class has the exact functionality as animal class along with its own functionality.

Constructors and Inheritance

How to call a base class constructor from a child class? It can be done by prefixing the keyword parent:: before construct from the child class constructor or by prefixing the name of the class before construct.

```
<?php
class Person
{
var $weight;
var $height;
var $sex;
var $age;
function __construct($w,$h,$sex,$age)           //base class constructor
{
   $this->weight=$w;
          $this->height=$h;
          $this->sex=$sex;
          $this->age=$age;
}
public function set_person($w,$h,$sex,$age)
{
          $this->weight=$w;
          $this->height=$h;
          $this->sex=$sex;
          $this->age=$age;
}
public function get_person()
{
          echo "weight is  $this->weight <br>";
          echo "height is $this->height <br>";
          echo "age is $this->age <br>";
          echo " sex is $this->sex <br>";
}
}
class Student extends Person
{
          var $id;
          var $stname;
          var $grad_level;
function __construct($w,$h,$sex,$age,$id,$stname,$grad_level)  //declaring child
                                                                 // constructor
                                                           //along with parent parameters
          {
                    parent::__construct($w,$h,$sex,$age);   //calling parent constructor
                    $this->id=$id;
```

```
                $this->stname=$stname;
                $this->grad_level=$grad_level;
        }
        public function set_student($id,$stname,$grad_level)
        {
                $this->id=$id;
                $this->stname=$stname;
                $this->grad_level=$grad_level;
        }
        public function get_student()
                {
        echo " student id  is $this->id <br>";
        echo "student name is $this->stname <br>";
        echo " graduation level is $this->grad_level<br> ";
        }
}
$obj1=new Student();
$obj2=new Student(50,180,20,"male",100,"karthik","PG");  //initializing object
$obj1->set_person(70,167,18,"female");
$obj1->get_person();
$obj1->set_student(123,"lalitha","UG");
$obj1->get_student();

?>
```



```
weight is 70
height is 167
age is female
sex is 18
student id is 123
student name is lalitha
graduation level is UG
weight is 50
height is 180
age is male
sex is 20
student id is 100
student name is karthik
graduation level is PG
```

Overriding methods

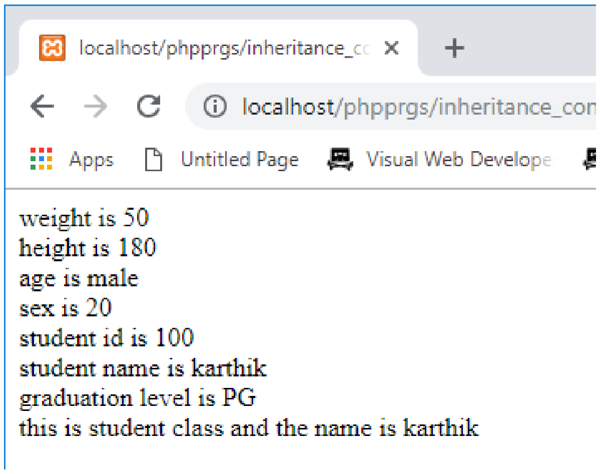
Redefining a base class method in a derived class is said to be overriding method. In the below program a function named display() is declared in both parent and child class.

When it is called from the main program, it access only the immediate class , i.e the child class method.

```
<?php
class Person
{
var $weight;
var $height;
var $sex;
var $age;
function __construct($w,$h,$sex,$age)
{
   $this->weight=$w;
         $this->height=$h;
         $this->sex=$sex;
         $this->age=$age;
}
public function set_person($w,$h,$sex,$age)
{
         $this->weight=$w;
         $this->height=$h;
         $this->sex=$sex;
         $this->age=$age;
}
public function get_person()
{
         echo "weight is  $this->weight <br>";
         echo "height is $this->height <br>";
         echo "age is $this->age <br>";
         echo " sex is $this->sex <br>";
}
public function display()        //overriding method
{
         echo "this is person class";
}
}
class Student extends Person
{
         var $id;
         var $stname;
         var $grad_level;
         function __construct($w,$h,$sex,$age,$id,$stname,$grad_level)
         {
                  parent::__construct($w,$h,$sex,$age);
                  $this->id=$id;
                  $this->stname=$stname;
```

```
                        $this->grad_level=$grad_level;
            }
            public function set_student($id,$stname,$grad_level)
            {
                        $this->id=$id;
                        $this->stname=$stname;
                        $this->grad_level=$grad_level;
            }
            public function get_student()
                        {
            echo " student id  is $this->id <br>";
            echo "student name is $this->stname <br>";
            echo " graduation level is $this->grad_level<br> ";
            }
            public function display()                    //overriding method
{
            echo "this is student class and the name is ".$this->stname;
}
}
//$obj1=new Student;
$obj2=new Student(50,180,20,"male",100,"karthik","PG");
//$obj1->set_person(70,167,18,"female");
$obj2->get_person();
//$obj1->set_student(123,"lalitha","UG");
$obj2->get_student();
$obj2->display();           //this call will call only the Student version of display() function.

?>
```

Output

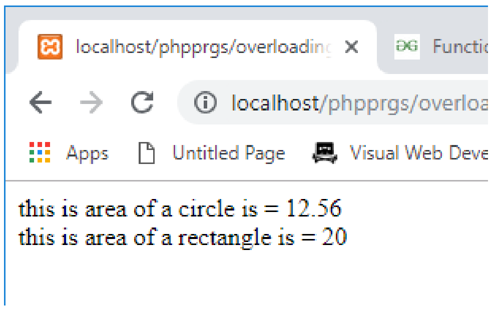

localhost/phpprgs/inheritance_co
localhost/phpprgs/inheritance_con
Apps
Untitled Page
Visual Web Develope

```
weight is 50
height is 180
age is male
sex is 20
student id is 100
student name is karthik
graduation level is PG
this is student class and the name is karthik
```

Overloading methods

Overriding is redefining a method, but overloading means creating an alternative version with a different argument list. For example to find the area of shapes, a function called area can be written with different arguments say, to find area of circle passing radius, to find area of rectangle passing length and breadth, to find area of square passing side of square and so on. The compiler will call the correct version of the function according the argument passed.

```
<?php
//php program for function overloading
class shape
{
function __call($name_of_function,$arguments)
{
if($name_of_function=='area')
{
if(count($arguments)==1)
{
return 3.14*$arguments[0]*$arguments[0];
}
if(count($arguments)==2)
{
return $arguments[0]*$arguments[1];
}
} //if ends
}  //function ends
}   //class ends
$obj1=new shape;
echo "this is area of a circle is = ";
echo($obj1->area(2));  //overloading
echo "<br>";
echo "this is area of a rectangle is = ";
echo($obj1->area(4,5));    //overloading
?>
```



```
this is area of a circle is = 12.56
this is area of a rectangle is = 20
```

Autoloading classes

When many classes are created and to be included in one file we can go for autoload function. This function is passed the names of any classes that PHP is looking for and can’t find in the current file. That means you can load the missing class using require or include like this , where the class $classname is in a file autoloadeg.php

```
function __autoload($classname)
{
        require $classname.".php";
}
```

In the below example two classes called employee.php and csc.php are written as follows;

Filename: employee.php

```
<?php
class employee
{
var $name;
var $age;
function set_info($data,$age)
{
$this->name=$data;
$this->age=$age;
}
function get_name()
{
return $this->name;
}
function get_age()
{
return $this->age;
}
}
?>
```

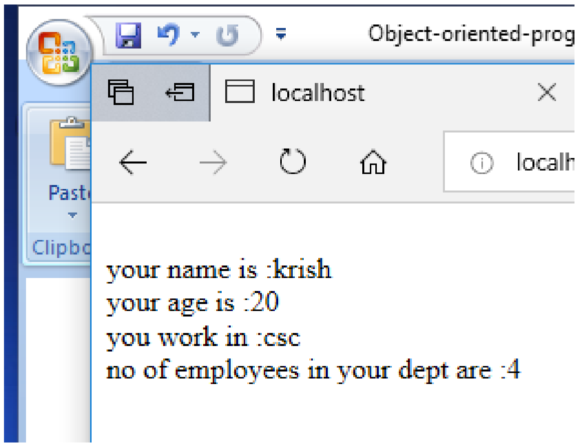
Filename : csc.php

```
<?php
class csc extends employee
{
var $dept;
var $no_of_emp;
function set_dept_info($dept,$no)
{
$this->dept=$dept;
$this->no_of_emp=$no;
}
function get_dept()
{
return $this->dept;
}
function get_no()
{
return $this->no_of_emp;
}
}
?>
```

Filename :autoloadeg.php

```
<?php
function __autoload($classname)
{
        require $classname.".php";
}
$obj1=new csc();
$obj1->set_info("krish",20);
$obj1->set_dept_info("csc",4);
echo "<br>your name is :".$obj1->get_name();
echo "<br>your age is :".$obj1->get_age();
echo "<br>you work in :".$obj1->get_dept();
echo "<br>no of employees in your dept are :".$obj1->get_no();

?>
```

Output



```
your name is :krish
your age is :20
you work in :csc
no of employees in your dept are :4
```