Scripting Languages in Web Applications

Intro to OPP in PHP

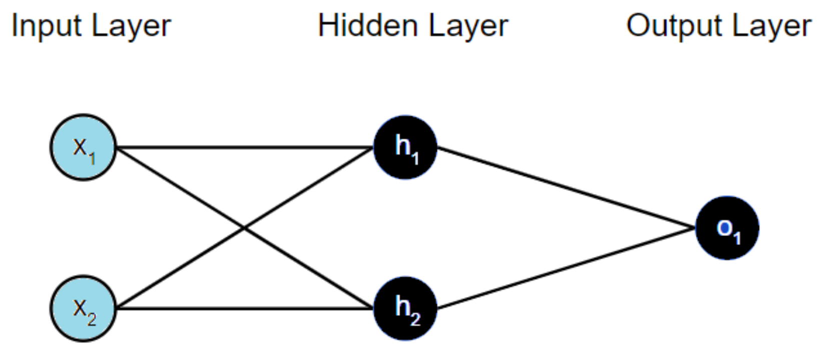
Janusz T. Starczewski

Deptartment of Computational Intelligence

Czestochowa University of Technology

OOP Case study on NN

As a case study, we analyze all object-oriented programming mechanisms enabled by PHP and create an example of an artificial neural network 2(inputs)-2(hidden neurons)-1(output neuron). The Ann neural network class is intended to contain two layers of hidden neurons and output neurons (alternatively, a list of layers). The Layer class is supposed to contain Node neurons. Any activation function. Confirm the operation of the Ann class by displaying the network's response (from the output neuron) to any single set of input signals.



Source: Janusz Starczewski

Additional source: https://www.php.net/manual/en/language.oop5.basic.php

Objects The basic idea of an object-oriented language is to enclose a set of variables and functions in a single unit and to protect both variables and functions from external interference and misuse. The mechanism that binds data and functions together is called encapsulation. Hermetic treatment of the code interior makes it easier to reuse it in many different projects. Methods are functions declared in an object that allow access to data, called properties.

A **class** is a construct or prototype from which objects are created. The components of a class are duplicated when creating each instance of the class, i.e. an object.

In PHP, the class definition looks like this:

```
<?php
class MyClass
{
 // Add property statements here
 // Add the methods here
}
?>
```

The **new** keyword is used to create an object instance, i.e. $object1 represents an object of the MyClass class.

```
<?php
$object1 = new MyClass;
?>
```

The contents of the MyClass class can be seen using the **var_dump()** function:

```
<?php
class MyClass
{
 // Add property statements here
 // Add the methods here
}
$object1 = new MyClass;
var_dump($object1);
?>
```

Result:

```
object(MyClass) #1 (0) {}
```

Setting properties

The **member variables of a class** are called **properties, attributes, or fields**.

Declaring an **access modifier** a property in a class requires the keywords public, protected, or private. Using var (PHP 4 compatible) the property will be defined as public. Importance of access modifiers:

- **public**: properties can be accessed from outside the class, from a script or from another class,
- **private**: no access is possible (from outside the class, neither by script nor from another class).
- **protected**: no access is allowed from outside the class, except for a class that inherits from the class containing the given protected property or method.

Members can be defined as public either explicitly or implicitly by default.

Once an object has been instantiated, you can access its properties using the object name and the -> operator.

```
<?php
class MyClass
{
 public $size =10;
}
$f = new MyClass;
echo $f->size;
?>
```

Result:

```
10
```

Note the single $ character in the $f->size reference.

In the example below, the display() method will print a string of characters with the specified font size and color in an HTML paragraph element using the php echo statement.

```
<?php
class MyClass
{
 public $size ="12px";
 public $color = "red";
 public $text = "laboratorium";
 public function display()
 {
 echo "<p style=font-size:".$this->size.";color:".$this->color.";>".$this->text."</p>";
 }
}
$f = new MyClass;
echo $f->display();
?>
```

Result: laboratorium

Accessing the object's components and calling display() again:

```
<?php
class MyClass
{
 public $size ="12px";
 public $color = "red";
 public $text = "laboratorium";
 public function display()
 {
 echo "<p style=font-size:".$this->size.";color:".$this->color.";>".$this->text."</p>";
 }
}
$f = new MyClass;
$f->size = "24px";
$f->color = "green";
$f->text = "nowy";
```

```
echo $f->display();
?>
```

Result:

```
nowy
```

Class constants belong not to objects, but to the class from which they are derived. They are declared with the const keyword and are not preceded by $.
When calling a class constant using the $classname::const syntax, classname can be an object or a variable with the class name. You can also access a static class constant by referencing the variable **class_name::$const_name** (since PHP 5.3.0).
The **scope resolution operator** :: in PHP allows access to static, constant and overridden properties or methods of a class.

```
<?php
  class MyClass
  {
  const constant1 = 'constant klasy';
  function displayconstant()
  {
  echo ”internal call self:” . self::constant1 . "<br>";
  }
```

```
 }
 echo ”class call:” . MyClass::constant1 . "<br>";
 $classname = "MyClass";
 echo ”call through the class name:” . $classname::constant1 . "<br>";
 $classinstantion = new MyClass();
 $classinstantion->displayconstant();
 echo ”call through the new object:” . $classinstantion::constant1 ."<br>";
 $constantname = ”constant1”;
 echo ”call through the new name:” . MyClass::$constantname ."<br>";
?>
```

A **constructor** is a built-in method that allows you to initialize the properties of an object when it is created. Constructor is not required,

```
<?php
 class MyClass
 {
 private $size;
 private $color;
 private $text;
// A three-parameter constructor declaration
 function __construct($size,$color,$text)
 {
 $this->size = $size;
 $this->color = $color;
 $this->text = $text;
 $this->display();
 }
function display()
{
echo "<p style=font-size:".$this->size.";color:".$this->color.";>".$this->text."</p>";
}
}

$f = new MyClass('10px','green','Laboratorium');
?>
```

A **destructor** is the complement of a constructor that is triggered when an object is destroyed.

```
<?php
// Define a class
class MyClass
{
 function __construct()
  {
    echo 'laboratorium'.'<br>';
    $this->nazwa = "MyClass";
  }
 function __destruct()
  {
    echo "Niszczenie " . $this->nazwa . "<br>";
  }
}
$obj = new MyClass();
?>
```

Result:

```
laboratorium
Niszczenie MyClass
```

Multiple instances of the same class

```
<?php
class MyClass
 {
 private $size;
 private $color;
 private $text;
 function __construct($size,$color,$text)
 {
 $this->size = $size;
 $this->color = $color;
 $this->text = $text;
 $this->display();
 }
 function display()
 {
```

```
echo "<p style=font-size:".$this->size.";color:".$this->color.";>".$this->text."</p>";
}
}
listNode = [];
listNode[] = new MyClass('10px','red','Node 1');
listNode[] = new MyClass('11px','green','Node 2');
listNode[] = new MyClass('12px','blue','Node 3');
?>
```

spl_autoload_register ()

enables automatic class loading when creating one PHP source file per class definition (since PHP 5.1.2).

Having in the php/classes/ class1.php and class2.php directories:

```
<?php
 spl_autoload_register(function ($classinstantion)
 {
 include 'php/classes/' . $classinstantion . '.php';
 });
 $print1 = new Class1();
 $print1 = new Class2();
?>
```

Inheritance using the **extends** word allows classes to create hierarchies and use methods and properties (that are public or protected) in child classes.

In the example subclass below, "DescendantClass" inherits all protected properties and public method from the "MyClass" class, plus the added text decoration attribute:

```
<?php
class MyClass
 {
 protected $size;
 protected $color;
 protected $text;
 function __construct($size,$color,$text)
 {
 $this->size = $size;
 $this->color = $color;
 $this->text = $text;
 $this->display();
 }
 public function display()
 {
 echo "<p style=font-size:".$this->size.";color:".$this->color.";>".$this->text."</p>";
 }
 }

class DescendantClass extends MyClass
 {
 //overriding of display method (redefinition)
 public function display()
 {
 echo "<p style=font-size:".$this->size.";color:".$this->color.";text-
decoration:underline;>".$this->text."</p>";
 }
 }

 $p = new MyClass('10px','yellow','Rodzic');
 $s = new DescendantClass('11px','green','Potomek');
?>
```

Result:

```
Rodzic
Potomek
```

Interfaces provide methods to implement, where:

1. derived classes can implement more than one interface,
2. interfaces can inherit from other interfaces,
3. all methods are public in the interface definition,
4. no name collision can occur between methods defined in different interfaces.

```
interface MyInterface
 {
   function method1 ();
   function method2 ();
 }
MyClass implements MyInterface
 {
   function method1 ()
    {
        // method1 definition
    }
   function method2 ()
```

```
    {
        // method2 definition
    }
}
```

Cloning objects is used to create a copy of an object using the __**clone** keyword, where:

1. all properties of the object have copied values,
2. properties that are references to other variables remain references,

```
<?php
class MyClass
{
 public $x;
 private $y;
 function __construct($x, $y)
 {
  $this->x = $x;
  $this->y = $y;
 }
 function __clone()
 {
  $this->x = "z";
 }
}
$a = new MyClass("laboratorium", "inf");
$b = clone $a;
var_dump($a);
echo '<br>';
var_dump($b);
?>
```

Result:

```
object (MyClass)#1 (2) {["x"] => string (3) "lab" ["y": "MyClass": private] => string (3) "inf"}
object (MyClass)#2 (2) {["x"] => string (1) "z" ["y": "MyClass": private] => string (3) "inf"}
```

Other magical methods

__get() is used to read data from unavailable properties.
__set() is run when writing data to unavailable properties.
__isset() is triggered by calling isset() or empty() on unavailable properties.
__unset() is called when unset() is used on unavailable properties.

__call() is triggered when calling unavailable methods in the object context.
__callStatic() is triggered when calling unavailable methods in a static context.
__sleep() is used to commit pending data or perform similar cleanup tasks, useful e.g. when large objects do not need to be completely saved.
__wakeup() is used to re-establish any database connections that may have been lost during serialization and perform other re-initialization tasks.

__toString() allows a class to decide how it will react when treated as a string.

__invoke() is called when the script tries to invoke an object as a function.

__set_state() is called as a static method for classes exported by var_export() since PHP 5.1.0.