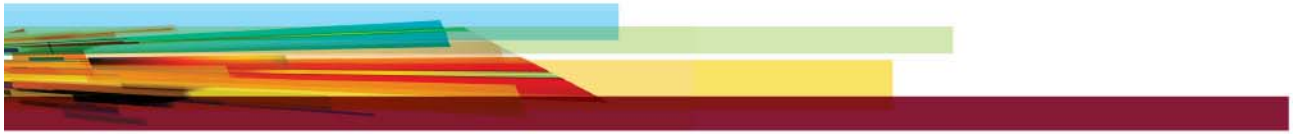
CEWP 459

PHP Programming with MySQL – Level I

Arrays and Objects



Arrays

Array Definition

- An array is a variable which can contain multiple values.
- Values can be any basic value type, or can be an object.
- An array can sometimes be referred to by an index (value)
- An array uses a key-pair structure.
- To view contents of an array use the var_dump() function.
- Arrays can hold values of mixed type.

Positional Array

- Simple Array

$array = array(“a”, “b”, “c”);

Access array element by position value.

0-Based.

echo $array[1]; Yields "b".

Array with Key Values

- Array with labels

```
$array = array(
      "rome"=>"Italian",
      "london"=>"English",
      "paris"=>"French");

echo $array["paris"];
```

Multidimentional Arrays

Simple definition: Arrays which in turn hold arrays within them.

```
$row_0 = array(1, 2, 3);
$row_1 = array(4, 5, 6);
$row_2 = array(7, 8, 9);
$multi = array($row_0, $row_1, $row_2);
```

```
$value = $multi[2][0]; // row 2, column 0. $value = 7
```

Multidimentional Array Definition

```
$cars = array
  (
     array("Volvo",80000,32000),
     array("BMW",72000,35000),
     array("Toyota",30000,12000)
  );
```

Array Definition

- Multidimensional Array

```
$families = array
(
   "Griffin"=>array("Peter", "Lois", "Megan"),
   "Quagmire"=>array("Glenn"),
   "Brown"=>array("Cleveland", "Loretta",
   "Junior")
);
```

Array Definition

- Multidimensional Array

```
$shop = array( array("rose", 1.25 , 15),
               array("daisy", 0.75 , 25),
               array("orchid", 1.15 , 7)
             );

echo $shop[0][0]." costs ".$shop[0][1]."
and you get ".$shop[0][2]."<br />";
```

array_chunk function

- $chunks = array_chunk(*array*, *size* [, *preserve_keys*]);

- Creates multidimensional array composed of new arrays based on "size" parameter.

```
$nums = range(1, 7);
$rows = array_chunk($nums, 3);
```

array_keys and array_values

- $array_of_keys = array_keys(*array*);
- $array_of_values = array_values(*array*);

```
$person = array('name' => 'Fred', 'age' => 35,
'wife' => 'Wilma');
$keys = array_keys($person); // $keys is
array('name', 'age', 'wife')
$values = array_values($person); // $values is
array('Fred', 35, 'Wilma');
```

Important Array Functions (Primary Functions)

- Count: Counts the members of an array
- Sort: Sort (shown later)

Array Functions

- Array_sum: Sums the values in an array.
- Array_walk: Performs a function for every member of the array.
- Array_reverse: Reverse order the array.
- Array_values: Returns the values of an array.
- Shuffle: Shuffles (randomizes) array elements.
- Array_merge(a,b)

Add to Array

1. Method
   - array_push(x);
2. Key
   - array('key') = x
3. Generic Add
   - array[] = x

Example

```
for ($a == 0; $a < 10; $a++)
{ $array1[$a] = "Value is $a"; }

var_dump($array1);

echo "<br><br>**********<br><br>";

$array2 = array(); //Need to define before array_push

for ($b == 0; $b < 10; $b++)
{ array_push($array2, "Value is $b"); }

var_dump($array2);
```

Example by Key

- $data[$key] = $value
- Simply add to array using the key.

Key sequential add

When started with an initial "=>" subsequent adds will be added with increasing key values.

$days = array(1 => 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday');

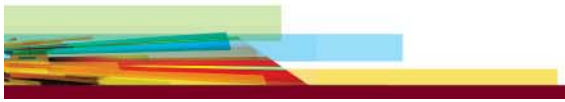
Tuesday = 2.

Add range

```
$numbers = range(2, 5);
// $numbers = array(2, 3, 4, 5);

$letters = range('a', 'z');
// $numbers holds the alphabet

$reversed_numbers = range(5, 2);
// $numbers = array(5, 4, 3, 2);
```



Exercise (Multidimensional arrays)

Create the following multidimensional array with the following parameters;

1. List of car brands (keyed)
2. Within each brand, a list of models (keyed)
3. Within each model, a list of colors available (non keyed)

- For a particular brand and model, show all the colors in a list.
- Eg: Colors for Toyota Yaris
  Green, Yellow,

array_slice

Like substr but for arrays.

Take a sub-array from an array based on offset and length.

$subset = array_slice(*array*, *offset*, *length*);

In Array

- Check if value exists in an array.
- bool in_array (mixed $needle , array $haystack [, bool $strict = FALSE])

- Useful for matching values.

In-array example

- Create a function that takes in a value, and counts the vowels and consonents in it.
- Use the following functions and technologies:
  - Arrays for looking up.
  - Substr to get each character.
    - string substr (string $string , int $start [, int $length])
  - Foreach loop.
  - A function to put this all together that returns an array with the number of vowels and consonents.

List function.

- Copy array values into variables.

```
$person = array('name' => 'Fred', 'age' => 35, 'wife' =>
'Betty');

list($n, $a, $w) = $person; // $n is 'Fred', $a is 35, $w is
'Betty'
```

List function (2)

```
$values = array('hello', 'world');
list($a, $b, $c) = $values;
```

What is $c?

Check if a member exists

- if (array_key_exists(*key*, *array*)) { ... }

Add/remove Members Using array_splice

- The array_splice( ) function can remove or insert elements in an array:
- $removed = array_splice(*array*, *start* [, *length* [, *replacement*]]);

```
$a = array(1,2,3,4,5);
$b = array_splice($a, 2,2, array(8) );
```

```
$a = ? | $b = ?
```

Extract Function

- Extracts keyed array to variables based on keys.

- $person = array('name' => 'Fred', 'age' => 35, 'wife' => 'Betty');

$name = 'Fred';

$age = 35;

$wife = 'Betty';

Compact function

- Opposite of extract.
- Creates array from variables.

```
$color = 'indigo';
$shape = 'curvy';
$floppy = 'none';
$a = compact('color', 'shape',
'floppy');
```

Traversal

```
foreach (array_expression as $value)
    statement

foreach (array_expression as $key => $value)
    statement
```

This traverses a positional or keyed array from start to finish, and places each single element from the collection into $key and $value.

Iterators

current( ) : Returns the element currently pointed at by the iterator

reset( ) : Moves the iterator to the first element in the array and returns it

next( ) : Moves the iterator to the next element in the array and returns it

prev( ) : Moves the iterator to the previous element in the array and returns it

end( ) : Moves the iterator to the last element in the array and returns it

each( ) : Returns the key and value of the current element as an array and moves the iterator to the next element in the array

key( ) : Returns the key of the current element

Example of Iterators

```
$ages = array('Person' => 'Age', 'Fred' => 35,
'Barney' => 30, 'Tigger' => 8, 'Pooh' => 40);
// start table and print heading
reset($ages);
List($c1, $c2) = each($ages);
echo("<table><tr><th>$c1</th><th>$c2</th></tr>\
n");
// print the rest of the values
while (list($c1,$c2) = each($ages)) {
echo("<tr><td>$c1</td><td>$c2</td></tr>\n");
}
// end the table
echo("</table>");
```

Exercise

Given

$ages = array('Person' => 'Age','Fred' => 35,'Barney' => 30,'Tigger' => 8,'Pooh' => 40);

- Create a FOR loop that emulates a FOREACH loop.

Why?

- FOR loops are a little faster than FOREACH dealing with huge arrays.

Array_walk function.

`array_walk(`*`array`*`, `*`function_name`*`);`

- Calls a user-defined function once per array element.

Exercise (array_walk)

- Make an array that contains this:
- “Civic” -> 24000
- “CRV” -> 32000
- “Fit” -> 17000
- Then write a function that adds a transportation fee of $500 + 5% of the cost.
- Then display the results (using that function).

Array_reduce

- $result = array_reduce(*array*, *function_name* [, *default*]);

Iteratively reduce the array to a single value using a callback function

Arguments to function are

- 1) Running total
- 2) Current working value
- 3) Initial value (if any)

Find the error

```
function addv($a, $b)
{
  $a = $a * $b;
  return $a;
}

$z = array(1,2,3,4,5);

$x=array_reduce($z,"addv");

echo $x; // 600
```

Array Sorting Function Grid

Effect	Ascending	Descending	User-defined order
Sort array by values, then reassign indexes starting with 0	`sort( )`	`rsort( )`	`usort( )`
Sort array by values	`asort( )`	`arsort( )`	`uasort( )`
Sort array by keys	`ksort( )`	`krsort( )`	`uksort( )`

Sorts

- Sort
  - Standard sort.
- Asort
  - Sort by values.
- Ksort
  - Sort by keys.
- Add R to sort for reverse sort. (arsort).

usort

- User defined sort.
- Sort based on a function that you decide the order.

- FUNCTION
- Function($a, $b)

- Function returns -1, 0, or 1 based on comparison of $a and $b.
- -1 = smaller, 0 = same, 1 = larger.

Other sorts

- Natural Sort
- Sorts based on the contents of string. (like student9, student10).
- natsort and natcasesort

Filter array.

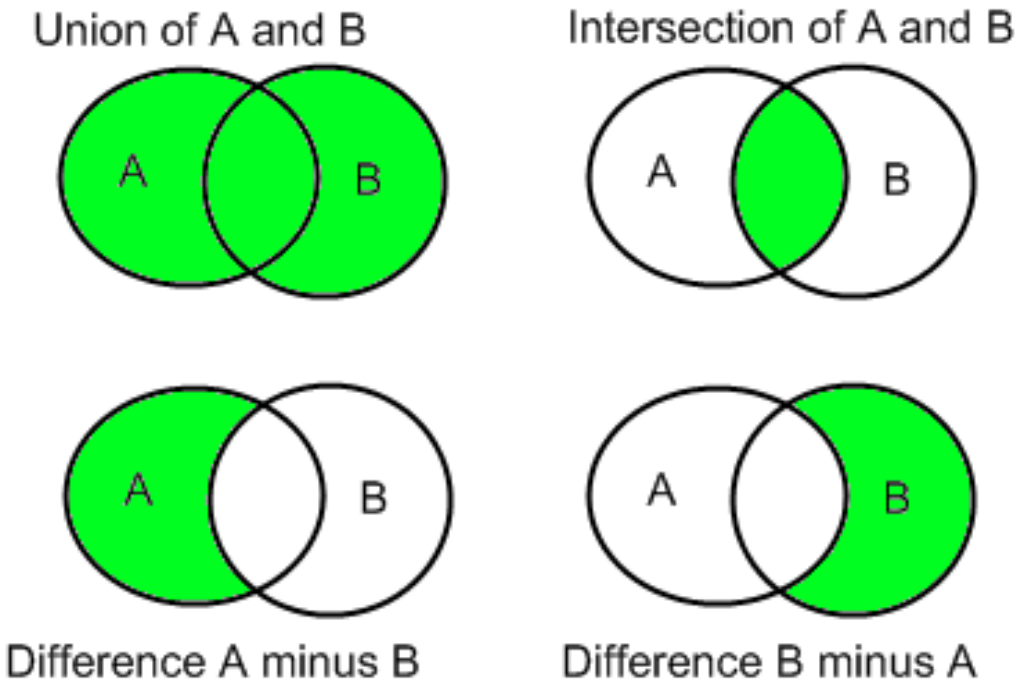
- $filtered = array_filter(*array*, *callback*);

- Callback returns 0 or 1.
- True = don't filter value.

- Create a function that filters all arrays to contain only even numbers.

Set operations

- array_merge
- Array_diff



Object Orientation

Class

- Must begin with the keyword “class”
- Class definition between curly braces { }.
- Contain properties and methods
- When referring to objects inside the class, use $this keyword.
- Template to create objects.
- Encapsulation

```
class MyClass
{
    class info goes here
}
```

Instantiate an object

- Create an object with the “new” keyword.

```
class MyClass
{
    // Class properties and methods go
here
}

$obj = new MyClass;

Optional- var_dump($obj);
```

- $object = new Class
- $object = new Person('Fred', 35);
- $class = 'Person';
- $object = new $class;

Class properties

- Define properties within the class, using the visibility desired.

```
class MyClass
{
    public $prop1 = "Some text";
    var $prop2 = "more text";
}

$obj1 = new MyClass; //create object

echo $obj1->prop1; //show value of
object's prop1.
```

Exercise 2 (Keep for future exercise)

- Create a class called “Car” and add the following attributes to it.
  - Model
  - Brand
  - Highway gas milage
- Create some car objects (2 or 3).
- Display the properties of the cars.

Variables

- Variables are defined with the "var" keyword.

```
var $t = 0;
```

Methods

- Methods are defined with function names.

- Class
- …
- function get_name() { }

Visibility

- Public
  - Accessible from anywhere (inside class, outside class, and children).
  - Attributes declared with “var” will be public by default.
- Private
  - Can only be accessed within the class.
- Protected
  - Can be accessed within the class, parents of the class and the derived classes (inherited)

Exercise 3

- Create a class like the last one
- Classname: Computer
- Set some attributes;
  - Price $1500 (Make private)
  - Memory 16 (Make public)

- Instantiate an object of the class, try to print out the price and the memory.

Class methods + GET SET

- Define properties within the class, using the visibility desired.

```
class MyClass
{
    public $prop1 = "I'm a class property";

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1 . "<br />";
    }
}
```

Getter and Setter

- Make attributes private
- Hides the data from the outside
- Encapsulation
- Security

```
public $prop1 = "I'm a class property";

    public function setProperty($newval)
    {
        $this->prop1 = $newval;
    }

    public function getProperty()
    {
        return $this->prop1 . "<br />";
    }
```

_GET _SET

Overloads

- echo $foo->bar;
- $var = $foo->bar;

$bar in the class must be inaccessible. If it is accessible it will not work.

Find the error

```
class foo {

    public $bar;
    public function __get($name) {

        echo "Get:$name";
        return $this->$name;
    }

    public function __set($name, $value) {

        echo "Set:$name to $value";
        $this->$name = $value;
    }
}
```

Exercise 4

- Use exercise 2 …
- Make your attributes private.
- Create getter and setter for each property you made.
  - Make sure the getter and setters are public.

- Make a trial run, use setter to set the properties of the car.
  - Display the properties using the getter.

Constructor

- When an object is instantiated, you can automatically call a built-in method at startup.
- This is to do some initial work when the object is created.
- Name of the class MUST BE __construct() (two underscores).

```
class MyClass
{
    public $prop1;

    public function __construct()
    {
        $this->prop1 = “test”;
    }
}
$obj = new MyClass();
echo $obj->prop1;
```

Exercise 5

- Add a constructor to your car class.
- The constructor should do 2 things.
- 1) set the value of the brand automatically to “Honda” (or whatever you like).
- 2) Display “The brand was automatically set to Honda” + BR

- As usual, make a new object, and display properties from it.

Destructor

- __destruct(); is the method name to define a destructor.
- Destructor is called when an object is destroyed.
- Unset($object) can destroy an object.
- Reminder: __CLASS__ will return the class name when echo'd
- Destructor is good when using a database manager class (it can close the connection when done with the class).

```
public function __destruct()
    {
        echo 'The class "', __CLASS__, '" was
destroyed.<br />';
    }
```

Object -> String

- Cannot echo an object (gives fatal error).
- Use the __toString() magic method.

```
MyClass …
public function __toString()
{
     echo “return string";
}
$obj = new MyClass();
echo $obj;
```

Exercise 6

- Make a toString function in your Car class that displays “This class is a template for new cars”.

- Make a new object from this class
- ECHO the object.
- echo $myNewCar

Inheritance

- Cannot echo an object (gives fatal error).
- Use the __toString() magic method.

```
Class Class1 { attr1, method1, method2 }

Class Class2 extends Class1 { attr2,
method3 }
```

RESULT

Class1 will contain attr1, method1, method2

Class2 will contain attr1, attr2, method1, method2, method3.

Exercise 7

- Create a NEW class called LeasedCar
- LeasedCar will extend Car
- In leasedCar, add a new attribute called “Lease rate” You can make it public or you can use getters/setters.

- Instantate a new $myCar object, but from LeasedCar.

```
$newLeasedCar = new LeasedCar();
```

Multiple Constructors

- PHP does not support multiple constructor types.
- We need to handle these manually if we want to emulate that behaviour.

- new self();

- Create a new instance of a class based on "self". Same type.

```
class Student
{
   public function __construct() {}

   public static function withID( $id ) {
            $instance = new self();
            $instance->loadByID( $id );
            return $instance;
   }

   public static function withRow( array $row ) {
            $instance = new self();
            $instance->fill( $row );
            return $instance;
   }

   protected function loadByID( $id ) {
            // do query
            $row = my_awesome_db_access_stuff( $id );
            $this->fill( $row );
   }

   protected function fill( array $row ) { // fill all properties from array }
}
```

Practice

1. Create a class that holds a student record.
2. Create a master class that is called student.
3. Create two classes, UndergradStudent and IndependentStudent.
4. Create a method that prints the name of the student as “Last, First”.
5. Create a method that prints the student number, prefixed with “I” for independent students and “U” for undergrad students. Assume the student number is 7 numbers and a preliminary letter.

Multiple Constructor Practice

- Create a Book class.
- Create multiple constructors that have the following signatures:
  - Book Name and ISBN
  - Book Name and Author Name

Inheritance

```
Class Class1 { attr1, method1, method2 }

Class Class2 extends Class1 { attr2,
method3 }
```

RESULT

Class1 will contain attr1, method1, method2

Class2 will contain attr1, attr2, method1, method2, method3.

Practice

1. Create a class that holds a student record.
2. Create a master class that is called student.
3. Create two classes, UndergradStudent and IndependentStudent.
4. Create a method that prints the name of the student as “Last, First”.
5. Create a method that prints the student number, prefixed with “I” for independent students and “U” for undergrad students. Assume the student number is 7 numbers and a preliminary letter.

Exercise 1 (Use previously created CAR class)

- Create a NEW class called LeasedCar
- LeasedCar will extend Car
- In leasedCar, add a new attribute called “Lease rate” You can make it public or you can use getters/setters.

- Instantate a new $myCar object, but from LeasedCar.

```
$newLeasedCar = new LeasedCar();
```

Multiple Constructors

- PHP does not support multiple constructor types.
- We need to handle these manually if we want to emulate that behaviour.

```
class Student
{
   public function __construct() {}

   public static function withID( $id ) {
            $instance = new self();
            $instance->loadByID( $id );
            return $instance;
   }

   public static function withRow( array $row ) {
            $instance = new self();
            $instance->fill( $row );
            return $instance;
   }

   protected function loadByID( $id ) {
            // do query
            $row = my_awesome_db_access_stuff( $id );
            $this->fill( $row );
   }

   protected function fill( array $row ) { // fill all properties from array }
}
```

Multiple Constructor Practice

- Create a Book class.
- Create multiple constructors that have the following signatures:
  - Book Name and ISBN
  - Book Name and Author Name

Usort with Classes

- Create a person class, with name and age as attributes.
- Create a collection of these persons.
- Sort the class by age.

Overriding

- A child class can override the parent (method and variables).

```
class Class1
{
        function func1()
        { echo "a1<br>"; }
}
class Class2 extends Class1
{
        function func1()
        {echo "a2<br>";}

        function func2()
        {
                $this->func1();
                parent::func1();
        }
}
$obj = new Class2();
$obj->func2();
```

Exercise 1 - Override

- Create the code on the previous slide and observe it's behaviour

Static Class

- A static variable or function is tied to the CLASS not to the object.
- Even if many objects are made, the static value will remain the same throughout all objects.
- A static variable is only available to a static function in a class.

```
MyClass …
public static $myStaticVariable = "hi";

echo MyClass::$myStaticVariable;
```

Accessing Static and Constant Values

- The :: operator will allow you to access static and constant variables in a class.

```
class MyClass
{
     const CONST_VALUE = 'A constant
value';
}

echo $classname::CONST_VALUE;
```

Static Class – Programming Exercise

- Create a static class that will give each object created a unique sequential object number.

Prior topics

- Array
  - Can contain objects.
  - $a = array($object1, $object2, $object3);
- FOREACH
  - Can cycle through all components of the class.

```
foreach($object as $key => $value)
{
   echo "echo $value <br>";
}
```

Exercise 2 – FOREACH / Classes

- Using your car example, do a foreach for an object you created and display all of it's attributes and methods.

Abstraction

- Classes defined as abstract may not be instantiated.
- Any class that contains at least one abstract method must be abstract.
- When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child.
- methods must be defined with the same (or a less restricted) visibility.

Abstraction Example

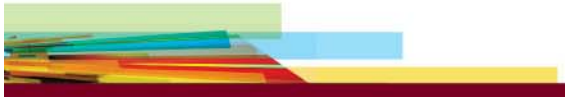
```
abstract class AbstractClass
{
    // Force Extending class to define this method
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);

    // Common method
    public function printOut() {
        print $this->getValue() . "\n";
    }
}

class ConcreteClass1 extends AbstractClass
{
    protected function getValue() {
        return "ConcreteClass1";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass1";
    }
}
```

$class1 = new ConcreteClass1;
$class1->getValue;



Exercise 3 - Abstraction

- Create an abstract class called “AbstractClass”.
- AbstractClass should have an abstract function in it called “getweight”.
- Create two classes that extend AbstractClass called “GetHeavyWeight” and “GetLightWeight”.
  - GetHeavyWeight should return a value of 1000.
  - GetLightWeight should return a value of 500.
- Try to instantiate AbstractClass and see what you get. Why doesn’t it work?

```
$test = new AbstractClass();
```

Interface

- Object interfaces allow you to create code which specifies which methods a class must implement, without having to define how these methods are handled.
- Interfaces are defined using the interface keyword
- All methods declared in an interface must be public, this is the nature of an interface.
- An interface is a DESIGN PATTERN.
- Example: I design classes for Bicycles, but I want to make sure that all bikes are defined with “showPrice” and “getInventory”. I would make an interface with these values and extend it.
- If you don’t implement the interface you will get a runtime error.

Interface Example

```
interface iBike
{
        public function showPrice($sku);
        public function getInventory($sku);
}
Class HybridBike implements iBike
{
        public function showPrice($sku)
        { return "Price for $sku is 123"; }

        public function getInventory($sku)
        { return "Inventory for $sku is 234"; }
}
$a = new HybridBike();
echo $a->showPrice(9999);
```

Exercise 4 – Interfaces

- Create an interface called iBike
- Include two functions;
  - public function showPrice($sku)
  - public function getInventory($sku);
- Create a new class called “HybridBike” that implements the functions. Return fake values like 123, 234.
- Instantiate a new HybridBike and execute a function.
- Remove one of the functions from HybridBike and see if it runs.

Exercise 4b - Multiple Inheritance

- Create interfaces that handle two functions;
- 1) ILoggable
  - Method: Log(val)

- 2) IPersistable
  - Method: Save()

Final

- Prevents child classes from overriding methods
- Prefix the definition with *final*.
- If the class itself is being defined final then it cannot be extended.

Final Example

```
class BaseClass {
   public function test() {
       echo "BaseClass::test() called\n";
   }

   final public function moreTesting() {
       echo "BaseClass::moreTesting() called\n";
   }
}

class ChildClass extends BaseClass {
   public function moreTesting() {
       echo "ChildClass::moreTesting() called\n";
   }
}
```

Exercise 5 - “Final”

- Create a class, give it a name.
- Create a method inside the class, prefix it by “final”.
- Create an extending class (that has the same method in it).

- This should give an error.

- Why do this?

Hinting

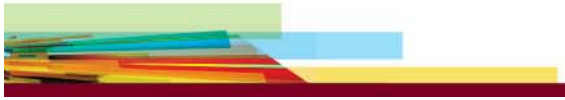
- Functions are now able to force parameters to be specific type or objects
- Specify the name of the class in the function prototype
- Can be a class name or type name.
- Can be array, interface, or class name.

Hinting Example

```
class Calculator
{
        public function average(array $array1)
        {
                return array_sum($array1) / count($array1);
        }
}

$calc = new Calculator();
$a = array(2,3,4,5);
$b = 4;

echo $calc->average($a); // Returns 3.5
echo $calc->average($b); // ERROR
```



Exercise 5 - Hinting

- Create a class like the previous example.
- Make a method (function) in the class, with 1 parameter and force it to be “array”.
- Then instantiate the class, and call the function by supplying an array.
- Try with an integer, it should fail.

Serialization

- Creates a STRING representation of an object.
- That string can be save in a text file or database.
- The string can be read somewhere else, and re-used.
- This is good for sending objects over the internet.
- Also good for saving objects in a database.
- You can serialize XML and send 1 big message over to a recipient.

Serialization 2

- A serialized string looks like this:

```
O:10:"Calculator":2:{s:4:"val1";i:2;s:4:"v
al2";i:5;}
```

Serialization Example (STORE)

```php
<?

//Use an include normally
class Calculator
{
    public $val1;
    public $val2;

    public function add()
    {
        return $this->val1 + $this->val2;
    }
}

$calc = new Calculator();       // $calc is a new object.

$calc->val1 = 2;
$calc->val2 = 5;

$a = serialize($calc);          // Serialize to $a.

setcookie('my_a_veriable',$a,time()+3600);

echo "Cookie created with serialized variable";

?>
```

Serialization Example (RETRIEVE)

```php
<?

//Use an include normally
class Calculator
{
    public $val1;
    public $val2;

    public function add()
    {
        return $this->val1 + $this->val2;
    }
}

$s = $_COOKIE['my_a_veriable'];
$a = unserialize($s);

echo $a->add();

?>
```

Exercise 6 - Serialization

- Create a class, create an object from the class.
- Serialize that object.
- Display it to the screen.

(This can now be save to a cookie, or a database or sent via web service).