ELSEWEDY UNIVERSITY OF TECHNOLOGY
- POLYTECHNIC OF EGYPT -

Web Programming

PhP

References

- PHP 8 Basics, 2020, Springer
  - https://link.springer.com/chapter/10.1007/978-1-4842-8082-9_2
- The Absolute Beginner's Guide to HTML and CSS, 2023, Springer
  - https://link.springer.com/chapter/10.1007/978-1-4842-9250-1_7
- W3C Tutorial
  - https://www.w3schools.com/php
  - https://www.w3schools.com/html
  - https://www.w3schools.com/js
- Additional Topics
  - JQuery: https://www.w3schools.com/jquery
  - Bootstrap 5.0: https://www.w3schools.com/bootstrap5
  - Laravel/Blade Framework 11.0: https://www.w3schools.in/laravel

PhP

PHP Arrays

```
$cars = array("Volvo", "BMW", "Toyota");
```

PHP Array Types

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

PhP

- Create Arrays
- Access Arrays
- Update Arrays
- Add Array Items
- Remove Array Items
- Sort Arrays

PhP

Array Items

Array items can be of any data type.

The most common are strings and numbers (int, float), but array items can also be objects, functions or even arrays.

```
$myArr = array("Volvo", 15, ["apples", "bananas"], myFunction);
```

PhP

```
<!DOCTYPE html>
<html>
<body>

<?php
// function example:
function myFunction() {
  echo "This text comes from a function";
}

// create array:
$myArr = array("Volvo", 15, ["apples", "bananas"], myFunction);

// calling the function from the array item:
$myArr[3]();
?>

</body>
</html>
```

Output: This text comes from a function

PhP

Array Functions

The real strength of PHP arrays are the built-in array functions, like the `count()` function for counting array items:

How many items are in the `$cars` array:

```php
$cars = array("Volvo", "BMW", "Toyota");

echo count($cars);
```

PHP Indexed Arrays

```php
$cars = array("Volvo", "BMW", "Toyota");

echo $cars[0];
```

PhP

Change Value

To change the value of an array item, use the index number:

```
$cars = array("Volvo", "BMW", "Toyota");
$cars[1] = "Ford";
```

Loop Through an Indexed Array

```
$cars = array("Volvo", "BMW", "Toyota");

foreach ($cars as $x) {

echo "$x <br>";

 }
```

PhP

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo", "BMW", "Toyota");

foreach ($cars as $x) {
  echo "$x <br>";
}
?>

</body>
</html>
```

```
Volvo
BMW
Toyota
```

PhP

Index Number   if you use the `array_push()` function to add a new item.

```
<!DOCTYPE html>
<html>
<body>
<pre>

<?php
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";

array_push($cars, "Ford");
var_dump($cars);
?>

</pre>
</body>
</html>
```

```
array(4) {
  [0]=>
  string(5) "Volvo"
  [1]=>
  string(3) "BMW"
  [2]=>
  string(6) "Toyota"
  [3]=>
  string(4) "Ford"
}
```

PhP

But if you have an array with random index numbers, like this:

```
<!DOCTYPE html>
<html>
<body>

<p>The next array item gets the index 15:</p>

<pre>
<?php
$cars[5] = "Volvo";
$cars[7] = "BMW";
$cars[14] = "Toyota";

array_push($cars, "Ford");
var_dump($cars);
?>
</pre>

</body>
</html>
```

The next array item gets the index 15:

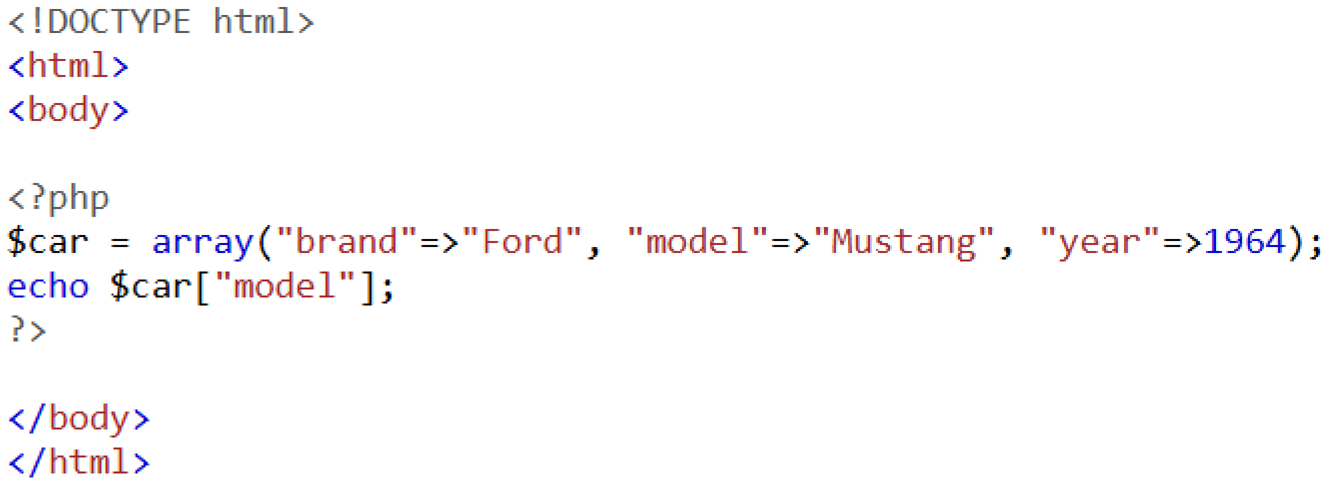
```
array(4) {
  [5]=>
  string(5) "Volvo"
  [7]=>
  string(3) "BMW"
  [14]=>
  string(6) "Toyota"
  [15]=>
  string(4) "Ford"
}
```

PhP

PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

```
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);
var_dump($car);
```



```
<!DOCTYPE html>
<html>
<body>

<?php
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);
echo $car["model"];
?>

</body>
</html>
```

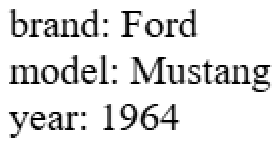
PhP

Change Value

```php
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);

$car["year"] = 2024;
```

Loop Through an Associative Array

```php
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);

foreach ($car as $x => $y) {

echo "$x: $y <br>";
}
```



```
brand: Ford
model: Mustang
year: 1964
```

PhP

PHP Create Arrays

```php
$cars = array("Volvo", "BMW", "Toyota");
```

```php
$cars = ["Volvo", "BMW", "Toyota"];
```

```php
$cars = [
"Volvo",
"BMW",
"Toyota" ];
```

Array Keys

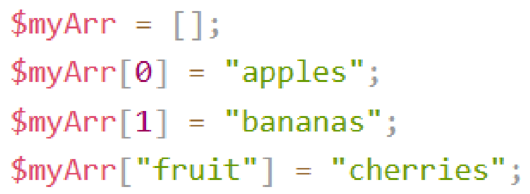
```php
$cars = [
0 => "Volvo",
1 => "BMW",
2 =>"Toyota" ];
```

```php
$myCar = [
"brand" => "Ford",
"model" => "Mustang",
"year" => 1964 ];
```

PhP

```php
$cars = [];
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] ="Toyota";
```

```php
$myCar = [];
$myCar["brand"] = "Ford";
$myCar["model"] = "Mustang";
$myCar["year"] = 1964;
```



```php
$myArr = [];
$myArr[0] = "apples";
$myArr[1] = "bananas";
$myArr["fruit"] = "cherries";
```

PhP

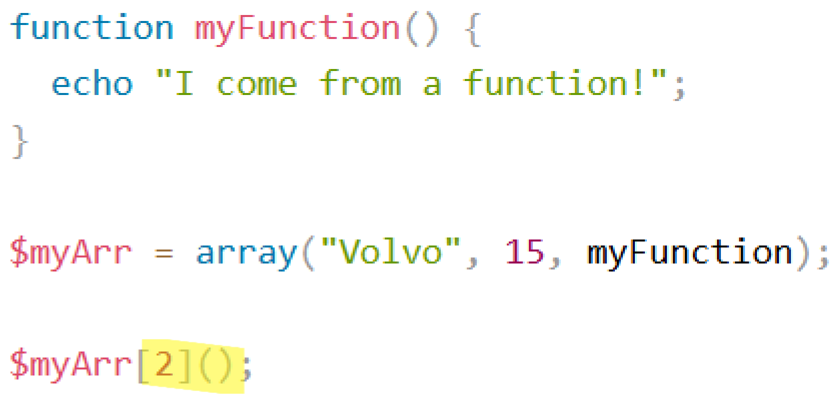
Access Array Item

```
$cars = array("Volvo", "BMW", "Toyota");
 echo $cars[2];


$cars = array("brand" => "Ford", "model" => "Mustang", "year" => 1964);
 echo $cars["year"];
```

PhP

Array items can be of any data type, including function.

To execute such a function, use the index number followed by parentheses `()`:



```php
function myFunction() {
  echo "I come from a function!";
}

$myArr = array("Volvo", 15, myFunction);

$myArr[2]();
```

I come from a function!

PhP

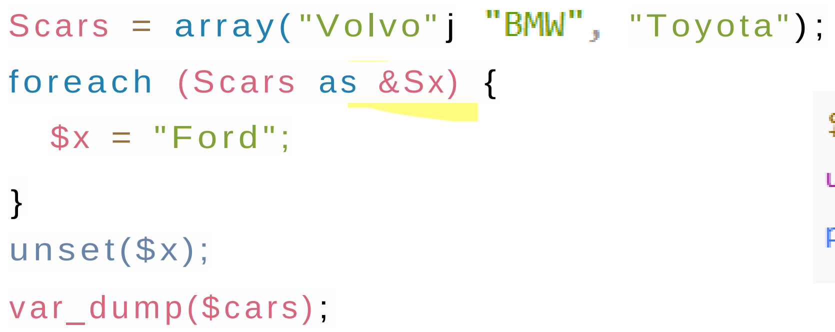
Use the key name when the function is an item in a associative array:

```
function myFunction() {
  echo "I come from a function!";
}

$myArr = array("car" => "Volvo", "age" => 15, "message" => myFunction);

$myArr["message"]();
```

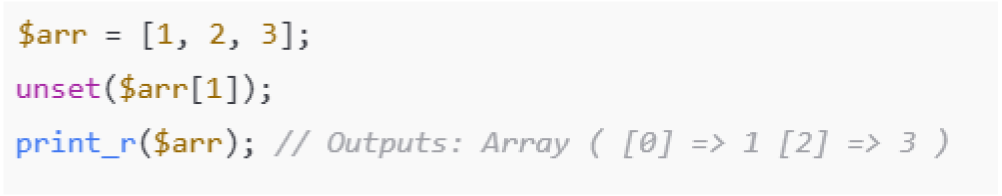
PhP

Update Array Items in a Foreach Loop

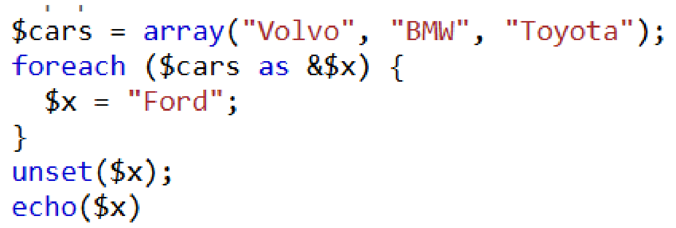
Change ALL item values to "Ford":



```
Scars = array("Volvo"j "BMW", "Toyota");
foreach (Scars as &Sx) {
  $x = "Ford";
}
unset($x);
var_dump($cars);
```



```
$arr = [1, 2, 3];
unset($arr[1]);
print_r($arr); // Outputs: Array ( [0] => 1 [2] => 3 )
```



```
$cars = array("Volvo", "BMW", "Toyota");
foreach ($cars as &$x) {
  $x = "Ford";
}
unset($x);
echo($x)
```

PhP

Remember to add the `unset()` function after the loop.
Without the `unset($x)` function, the `$x` variable will remain as a reference to the last array item.

```php
$cars = array("Volvo", "BMW", "Toyota");
foreach ($cars as &$x) {
  $x = "Ford";
}

$x = "ice cream";

var_dump($cars);
```

```
array(3) {
  [0]=>
  string(4) "Ford"
  [1]=>
  string(4) "Ford"
  [2]=>
  &string(9) "ice cream"
}
```

PhP

Add Array Item

```php
$fruits = array("Apple", "Banana", "Cherry");

$fruits[] = "Orange";
```

```php
$cars = array("brand" => "Ford", "model" => "Mustang");

$cars["color"] = "Red";
```

Add three item to the `fruits` array:

```php
$fruits = array("Apple", "Banana", "Cherry");
array_push($fruits, "Orange", "Kiwi", "Lemon");
```

```php
$cars = array("brand" => "Ford", "model" => "Mustang");
$cars += ["color" => "red", "year" => 1964];
```

PhP

Remove Array Item

To remove an existing item from an array, you can use the `array_splice()` function.

With the `array_splice()` function you specify the index (where to start) and how many items you want to delete.

```php
$cars = array("Volvo", "BMW", "Toyota");
array_splice($cars, 1, 1);
```

```
array(2) {
  [0]=>
  string(5) "Volvo"
  [1]=>
  string(6) "Toyota"
}
```

PhP

Using the unset Function

You can also use the `unset()` function to delete existing array items.

```
$cars = array("Volvo", "BMW", "Toyota");
unset($cars[1]);
```

The `unset()` function does not re-arrange the indexes, meaning that after deletion the array will no longer contain the missing indexes.

```
array(2) {
  [0]=>
  string(5) "Volvo"
  [2]=>
  string(6) "Toyota"
}
```

PhP

Remove 2 items, starting a the second item (index 1):

```
$cars = array("Volvo", "BMW", "Toyota");
array_splice($cars, 1, 2);
```

Remove the first and the second item:

```
$cars = array("Volvo", "BMW", "Toyota");
 unset($cars[0], $cars[1]);
```

Remove the "model":

```
$cars = array("brand" => "Ford", "model" => "Mustang", "year" => 1964);
unset($cars["model"]);
```

PhP

Using the array_diff Function

You can also use the `array_diff()` function to remove items from an associative array.

```
$cars = array("brand" => "Ford", "model" => "Mustang", "year" => 1964);

$newarray = array_diff($cars, ["Mustang", 1964]);
```

The `array_diff()` function takes *values* as parameters, and not *keys*.

Remove the last item:

```
$cars = array("Volvo", "BMW", "Toyota");
array_pop($cars);
```

Remove the first item:

```
$cars = array("Volvo", "BMW", "Toyota");
array_shift($cars);
```

PhP

PHP Sorting Arrays

$\longrightarrow$ small to big

- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order
- asort() - sort associative arrays in ascending order, according to the value
- ksort() - sort associative arrays in ascending order, according to the key
- arsort() - sort associative arrays in descending order, according to the value
- krsort() - sort associative arrays in descending order, according to the key

https://www.w3schools.com/php/php_arrays_sort.asp

PhP

PHP Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

PHP supports multidimensional arrays that are two, three, four, five, or more levels deep.

- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

PhP

PHP - Two-dimensional Arrays

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

```
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);
```

PhP

Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.

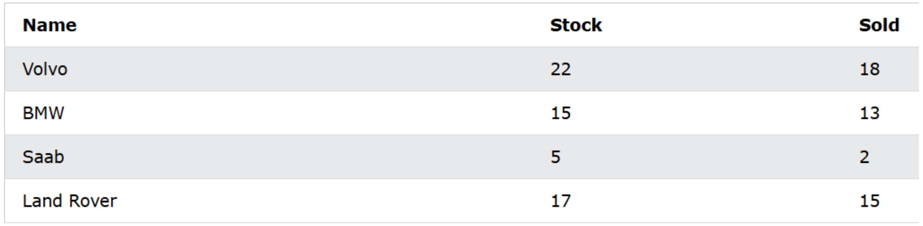
To get access to the elements of the $cars array we must point to the two indices (row and column):

```
<?php
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);

echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>
```

```
Volvo: In stock: 22, sold: 18.
BMW: In stock: 15, sold: 13.
Saab: In stock: 5, sold: 2.
Land Rover: In stock: 17, sold: 15.
```

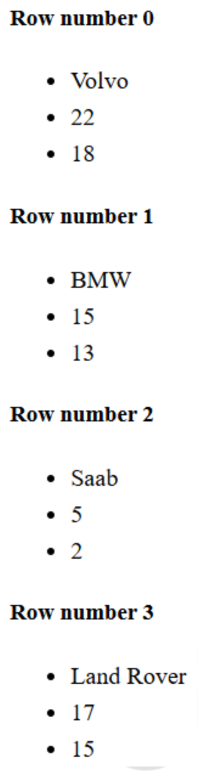
PhP

We can also put a `for` loop inside another `for` loop to get the elements of the $cars array (we still have to point to the two indices):



Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

```
for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
    for ($col = 0; $col < 3; $col++) {
      echo "<li>".$cars[$row][$col]."</li>";
    }
  echo "</ul>";
}
```



```
Row number 0

  • Volvo
  • 22
  • 18

Row number 1

  • BMW
  • 15
  • 13

Row number 2

  • Saab
  • 5
  • 2

Row number 3

  • Land Rover
  • 17
  • 15
```

```
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);

for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$cars[$row][$col]."</li>";
  }
  echo "</ul>";
}
?>

</body>
</html>
```

```
Row number 0

  • Volvo
  • 22
  • 18

Row number 1

  • BMW
  • 15
  • 13

Row number 2

  • Saab
  • 5
  • 2

Row number 3

  • Land Rover
  • 17
  • 15
```

PhP

Self Study

PHP Global Variables - Superglobals

PhP

PHP OOP

Classes and Objects

```php
<?php
class Fruit {
  // code goes here...
}
?>
```

Note: In a class, variables are called properties and functions are called methods!

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

PhP

Define Objects

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

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

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

```php
$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

PhP

PHP - The $this Keyword

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

```
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
?>
```

So, where can we change the value of the $name property? There are two ways:
1. Inside the class (by adding a set_name() method and use $this):

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

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

PhP

2. Outside the class (by directly changing the property value):

```php
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
$apple->name = "Apple";

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

PhP

PHP - instanceof

You can use the `instanceof` keyword to check if an object belongs to a specific class:

```php
<?php
$apple = new Fruit();
var_dump($apple instanceof Fruit);
?>
```

PhP

PHP - The __construct Function

If you create a __construct() function, PHP will automatically call this function when you create an object from a class.

```
<?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();
?>
```

PhP

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

  function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  function get_name() {
    return $this->name;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit("Apple", "red");
echo $apple->get_name();
echo "<br>";
echo $apple->get_color();
?>
```

PhP

PHP - 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 (__)!

The example below has a __construct() function that is automatically called when you create an object from a class, and a __destruct() function that is automatically called at the end of the script:

PhP

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

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

$apple = new Fruit ("Apple");
?>
```

PhP

PHP - 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

```php
<?php
class Fruit {
  public $name;
  protected $color;
  private $weight;
}

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

PhP

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

  function set_name($n) {  // a public function (default)
    $this->name = $n;
  }
  protected function set_color($n) { // a protected function
    $this->color = $n;
  }
  private function set_weight($n) { // a private function
    $this->weight = $n;
  }
}

$mango = new Fruit();
$mango->set_name('Mango'); // OK
$mango->set_color('Yellow'); // ERROR
$mango->set_weight('300'); // ERROR
?>
```

PhP

PHP OOP - 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.

PhP

```php
<?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();
?>
```

PhP

PHP - Inheritance and the Protected Access Modifier

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  protected function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}

// Try to call all three methods from outside class
$strawberry = new Strawberry("Strawberry", "red");  // OK. __construct() is public
$strawberry->message(); // OK. message() is public
$strawberry->intro(); // ERROR. intro() is protected
?>
```

if we try to call a `protected` method (intro()) from outside the class, we will receive an error. `public` methods will work fine!

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  protected function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
    // Call protected method from within derived class - OK
    $this -> intro();
  }
}

$strawberry = new Strawberry("Strawberry", "red"); // OK. __construct() is public
$strawberry->message(); // OK. message() is public and it calls intro() (which is protected) from within the derived class
?>
```

we see that all works fine! It is because we call the `protected` method (intro()) from inside the derived class.

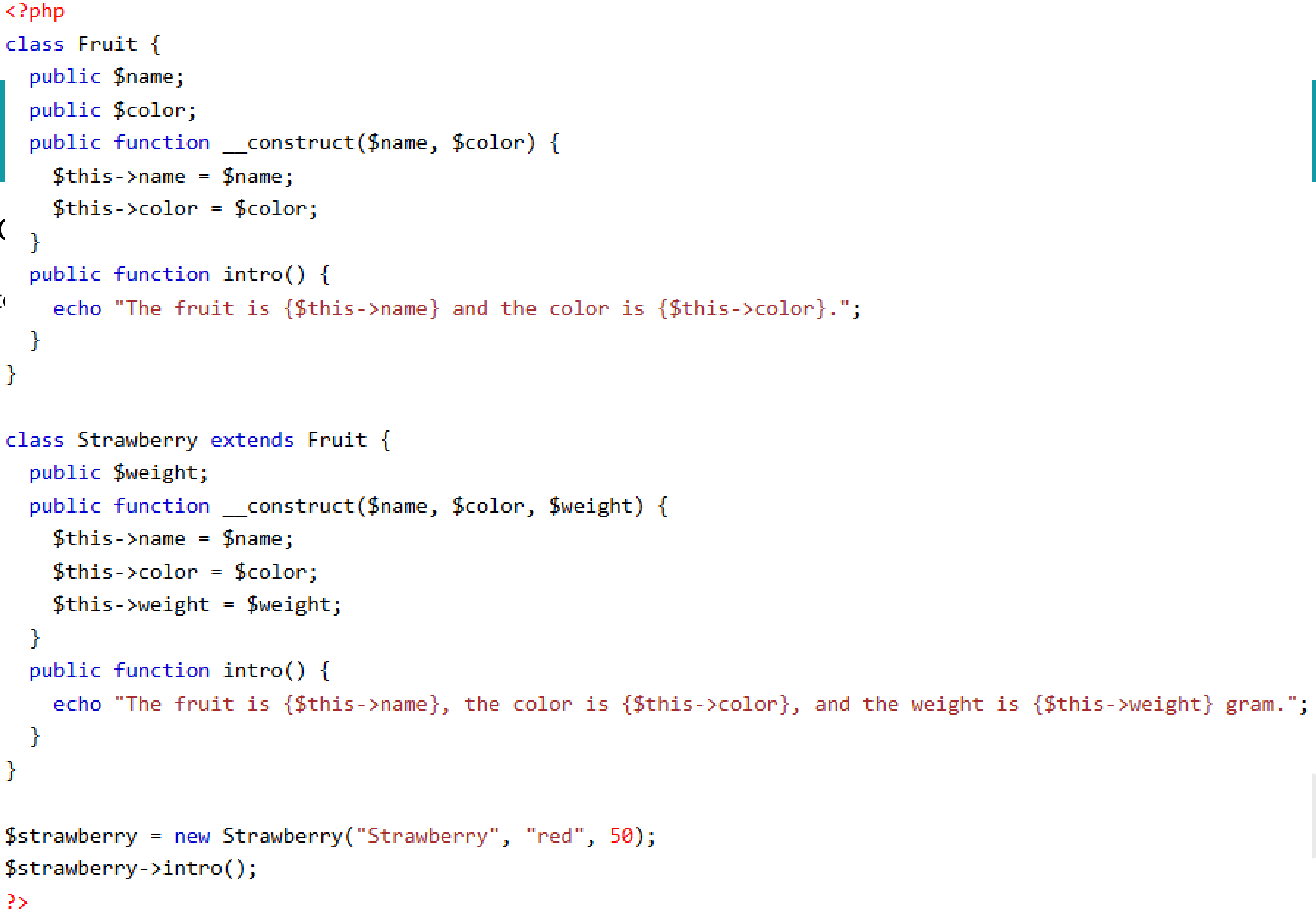
PhP

PHP - Overriding Inherited Methods

Inherited methods can be overridden by redefining the methods (use the same name) in the child class.

PHP - (

Inherit



```php
<?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();
?>
```

PhP

PHP - The final Keyword

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

The following example shows how to prevent class inheritance:

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

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

The following example shows how to prevent method overriding:

```
<?php
class Fruit {
  final public function intro() {
    // some code
  }
}

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

PhP

PHP - Class Constants

A class constant is declared inside a class with the `const` keyword.

A constant cannot be changed once it is declared.

Class constants are case-sensitive. However, it is recommended to name the constants in all uppercase letters.

We can access a constant from outside the class by using the class name followed by the scope resolution operator (`::`) followed by the constant name, like here:

```php
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
}

echo Goodbye::LEAVING_MESSAGE;
?>
```

PhP

Or, we can access a constant from inside the class by using the `self` keyword followed by the scope resolution operator (`::`) followed by the constant name, like here:

```php
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
  public function byebye() {
    echo self::LEAVING_MESSAGE;
  }
}

$goodbye = new Goodbye();
$goodbye->byebye();
?>
```

PhP

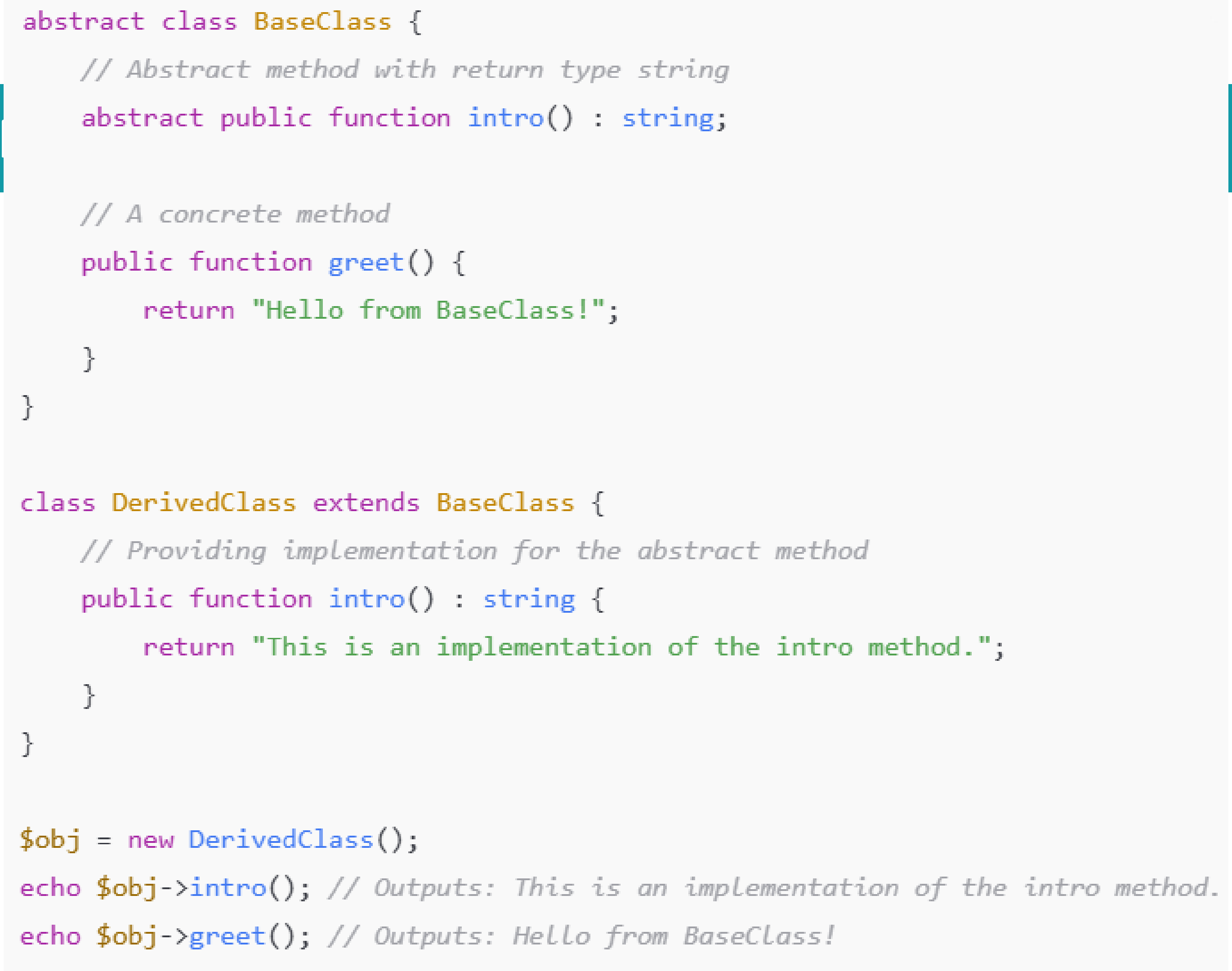
PHP OOP - Abstract Classes

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.

```php
<?php
abstract class ParentClass {
  abstract public function someMethod1();
  abstract public function someMethod2($name, $color);
  abstract public function someMethod3() : string;
}
?>
```

Ph



```php
abstract class BaseClass {
    // Abstract method with return type string
    abstract public function intro() : string;

    // A concrete method
    public function greet() {
        return "Hello from BaseClass!";
    }
}

class DerivedClass extends BaseClass {
    // Providing implementation for the abstract method
    public function intro() : string {
        return "This is an implementation of the intro method.";
    }
}

$obj = new DerivedClass();
echo $obj->intro(); // Outputs: This is an implementation of the intro method.
echo $obj->greet(); // Outputs: Hello from BaseClass!
```

```php
<?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();
?>
```

PhP

```php
<?php
abstract class ParentClass {
  // Abstract method with an argument
  abstract protected function prefixName($name);
}

class ChildClass extends ParentClass {
  public function prefixName($name) {
    if ($name == "John Doe") {
      $prefix = "Mr.";
    } elseif ($name == "Jane Doe") {
      $prefix = "Mrs.";
    } else {
      $prefix = "";
    }
    return "{$prefix} {$name}";
  }
}

$class = new ChildClass;
echo $class->prefixName("John Doe");
echo "<br>";
echo $class->prefixName("Jane Doe");
?>
```

PhP

PHP - Interfaces vs. Abstract Classes

Interface are similar to abstract classes. The difference between interfaces and abstract classes are:

- Interfaces cannot have properties, while abstract classes can
- All interface methods must be public, while abstract class methods is public or protected
- All methods in an interface are abstract, so they cannot be implemented in code and the abstract keyword is not necessary
- Classes can implement an interface while inheriting from another class at the same time

PhP

To implement an interface, a class must use the `implements` keyword.

A class that implements an interface must implement **all** of the interface's methods.

```php
<?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 ";
  }
}
```

```php
// 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();
}
?>
```

PhP

PHP OOP - Traits

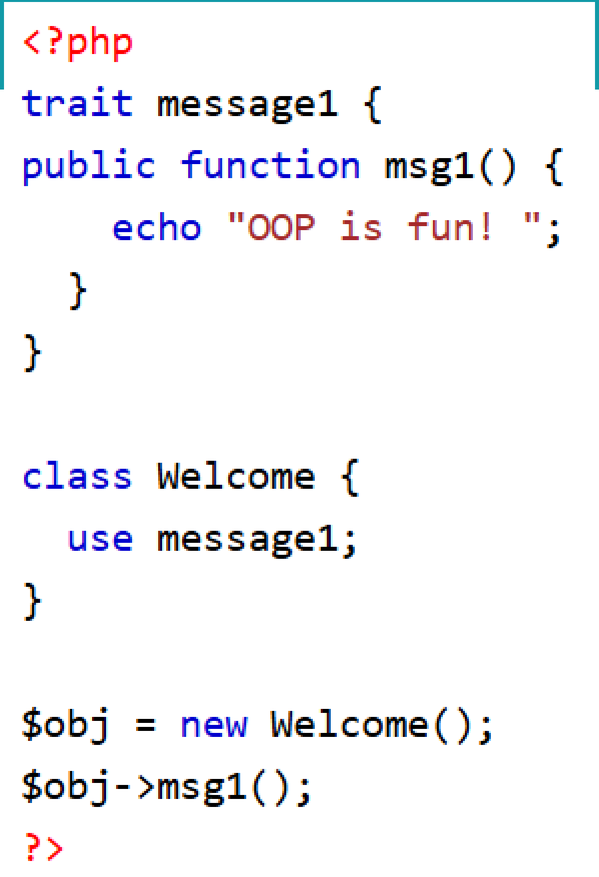
PHP only supports single inheritance: a child class can inherit only from one single parent.

So, what if a class needs to inherit multiple behaviors? OOP traits solve this problem.

Traits are used to declare methods that can be used in multiple classes. Traits can have methods and abstract methods that can be used in multiple classes, and the methods can have any access modifier (public, private, or protected).

PhP

PHP OOP - Traits



```
<?php
trait message1 {
public function msg1() {
    echo "OOP is fun! ";
  }
}

class Welcome {
  use message1;
}

$obj = new Welcome();
$obj->msg1();
?>
```

```php
<?php
trait message1 {
  public function msg1() {
    echo "OOP is fun! ";
  }
}

trait message2 {
  public function msg2() {
    echo "OOP reduces code duplication!";
  }
}

class Welcome {
  use message1;
}

class Welcome2 {
  use message1, message2;
}

$obj = new Welcome();
$obj->msg1();
echo "<br>";

$obj2 = new Welcome2();
$obj2->msg1();
$obj2->msg2();
?>
```

```
OOP is fun!
OOP is fun! OOP reduces code duplication!
```

Example Explained

Here, we declare two traits: message1 and message2. Then, we create two classes: Welcome and Welcome2. The first class (Welcome) uses the message1 trait, and the second class (Welcome2) uses both message1 and message2 traits (multiple traits are separated by comma).

PhP

PHP - Static Methods

Static methods can be called directly - without creating an instance of the class first.

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

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

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

PhP

PHP OOP - Static

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

  public function __construct() {
    self::welcome();
  }
}

new greeting();
?>
```

PhP

PHP OOP - Static

Static methods can also be called from methods in other classes. To do this, the static method should be `public`:

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

class B {
  public function message() {
    A::welcome();
  }
}

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

PhP

PHP OOP - Static

To call a static method from a child class, use the parent keyword inside the child class. Here, the static method can be public or protected.

```
<?php
class domain {
  protected static function getWebsiteName() {
    return "W3Schools.com";
  }
}

class domainW3 extends domain {
  public $websiteName;
  public function __construct() {
    $this->websiteName = parent::getWebsiteName();
  }
}

$domainW3 = new domainW3;
echo $domainW3 -> websiteName;
?>
```

PhP

PHP OOP - Static

A class can have both static and non-static properties. A static property can be accessed from a method in the same class using the `self` keyword and double colon (::):

```php
<?php
class pi {
  public static $value=3.14159;
  public function staticValue() {
    return self::$value;
  }
}

$pi = new pi();
echo $pi->staticValue();
?>
```

PhP

PHP OOP - Static

To call a static property from a child class, use the parent keyword inside the child class:

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

class x extends pi {
  public function xStatic() {
    return parent::$value;
  }
}

// Get value of static property directly via child class
echo x::$value;

// or get value of static property via xStatic() method
$x = new x();
echo $x->xStatic();
?>
```