### Php

## class

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

<?php  
class Fruit {  
  // Properties  
  public $name;  
  public $color;  
  
  // Methods  
  function set\_name($name) {  
    $this->name = $name;  
  }  
  function get\_name() {  
    return $this->name;  
  }  
}  
?>

<?php  
class Fruit {  
  // Properties  
  public $name;  
  public $color;  
  
  // Methods  
  function set\_name($name) {  
    $this->name = $name;  
  }  
  function get\_name() {  
    return $this->name;  
  }  
}  
  
$apple = new Fruit();  
$banana = new Fruit();

$banana->set\_name('Banana');  
  
echo $apple->get\_name();  
echo "<br>";  
echo $banana->get\_name();  
?>

<?php  
class Fruit {  
  // Properties  
  public $name;  
  public $color;  
  
  // Methods  
  function set\_name($name) {  
    $this->name = $name;  
  }  
  function get\_name() {  
    return $this->name;  
  }  
  function set\_color($color) {  
    $this->color = $color;  
  }  
  function get\_color() {  
    return $this->color;  
  }  
}  
  
$apple = new Fruit();  
$apple->set\_name('Apple');  
$apple->set\_color('Red');  
echo "Name: " . $apple->get\_name();  
echo "<br>";  
echo "Color: " . $apple->get\_color();  
?>

$apple = new Fruit();

var\_dump($apple instanceof Fruit);

?>

### Constructor

<?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  
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();  
?>

### \_\_destruct

 \_\_destruct() function that is automatically called at the end of the script

<?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");  
?>

### 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  
class Fruit {  
  public $name;  
  protected $color;  
  private $weight;  
}  
  
$mango = new Fruit();  
$mango->name = 'Mango'; // OK  
$mango->color = 'Yellow'; // ERROR  
$mango->weight = '300'; // ERROR  
?>

## OOP – Inheritance

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

<?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();  
?>

### OOP - Class Constants

Class constants can be useful if you need to define some constant data within a class.

<?php  
class Goodbye {  
  const LEAVING\_MESSAGE = "Thank you for visiting W3Schools.com!";  
}  
  
echo Goodbye::LEAVING\_MESSAGE;  
?>

### PHP - What are Abstract Classes and Methods?

<?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();  
?>

### OOP – Interfaces

Interfaces allow you to specify what methods a class should implement.

* 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  
interface Animal {  
  public function makeSound();  
}  
  
class Cat implements Animal {  
  public function makeSound() {  
    echo "Meow";  
  }  
}  
  
$animal = new Cat();  
$animal->makeSound();  
?>

### 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.

<?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();  
?>

### Static Methods

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

Static methods are declared with the static keyword:

<?php  
class greeting {  
  public static function welcome() {  
    echo "Hello World!";  
  }  
}  
  
// Call static method  
greeting::welcome();  
?>

### PHP Namespaces

Namespaces are qualifiers that solve two different problems:

1. They allow for better organization by grouping classes that work together to perform a task
2. They allow the same name to be used for more than one class

<?php  
$table = new Html\Table();  
$row = new Html\Row();  
?>

Sql

# Pdo

<?php  
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);  
?>

<?php  
try {  
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);  
} catch (PDOException $e) {  
// attempt to retry the connection after some timeout for example  
}

$stmt = $pdo->query("SELECT \* FROM users ORDER BY id DESC LIMIT 1");

$user = $stmt->fetch();

// select a particular user by id

$stmt = $pdo->prepare("SELECT \* FROM users WHERE id=?");

$stmt->execute([$id]);

$user = $stmt->fetch();

$dbh = new PDO("mysql:host=...;dbname=mysql", ...);

$dbh->query("create database newdatabase");

$dbh->query("use newdatabase");

### insert

$sql = "INSERT INTO users (name, surname, sex) VALUES (?,?,?)";

$stmt= $pdo->prepare($sql);

$stmt->execute([$name, $surname, $sex]);

$stmt = $pdo->prepare("SELECT \* FROM auction WHERE name LIKE ?")

$stmt->execute(array("%$query%"));

// iterating over a statement

foreach($stmt as $row) {

echo $row['name'];

}

### UPDATE

$sql = "UPDATE users SET name=?, surname=?, sex=? WHERE id=?";

$stmt= $pdo->prepare($sql);

$stmt->execute([$name, $surname, $sex, $id]);

$sql = "DELETE FROM users WHERE id=?";

$stmt= $pdo->prepare($sql);

$stmt->execute([$id]);

### Pdo defaults

$host = '127.0.0.1';

$db = 'test';

$user = 'root';

$pass = '';

$charset = 'utf8mb4';

### Select:

// select a particular user by id

$stmt = $pdo->prepare("SELECT \* FROM users WHERE id=?");

$stmt->execute([$id]);

$user = $stmt->fetch();

$stmt = $pdo->prepare("SELECT \* FROM users LIMIT ?, ?");

$stmt->execute([$limit, $offset]);

while ($row = $stmt->fetch()) {

echo $row['name']."<br />\n";

}

$stmt = $pdo->prepare("SELECT \* FROM users LIMIT :limit, :offset");

$stmt->execute(['limit' => $limit, 'offset' => $offset]);

$data = $stmt->fetchAll();

// and somewhere later:

foreach ($data as $row) {

echo $row['name']."<br />\n";

}

### DELETE table

DROP TABLE *table\_name*;

### Pdo delele

$sql = "DELETE FROM users WHERE id=?";

$stmt= $pdo->prepare($sql);

$stmt->execute([$id]);

### TRUNCATE

TRUNCATE TABLE *table\_name*;

$dsn = "mysql:host=$host;dbname=$db;charset=$charset";

$options = [

PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION,

PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC,

PDO::ATTR\_EMULATE\_PREPARES => false,

];

$pdo = new PDO($dsn, $user, $pass, $options);

### .htaccess

DirectoryIndex home.html

DirectoryIndex index.html home.html config.php

**Block a specific IP or range of IPs:**

Order Deny,Allow

Deny from 192.206.221.140

(Here 192.206.221.140 is a specific IPv4 Address)

Order Allow,Deny

Deny from 192.192.\*.\*

Allow from all

**301 Permanent Redirect**

Redirect 301 / <http://domain.com>

RewriteEngine on

RewriteCond %{HTTP\_HOST} ^geeksforgeeks.com [NC]

RewriteRule ^(.\*)$ http://www.geeksforgeeks.com/$1 [L,R=301,NC]