This document describes PHP 5+.

```php
<?php // PHP code must be enclosed with <?php tags

// If your php file only contains PHP code, it is best practice
// to omit the php closing tag to prevent accidental output.

// Two forward slashes start a one-line comment.

# So will a hash (aka pound symbol) but // is more common

/*
     Surrounding text in slash-asterisk and asterisk-slash
     makes it a multi-line comment.
*/

// Use "echo" or "print" to print output
print('Hello '); // Prints "Hello " with no line break

// () are optional for print and echo
echo "World\n"; // Prints "World" with a line break
// (all statements must end with a semicolon)

// Anything outside <?php tags is echoed automatically
?>
Hello World Again!
<?php


/************************************
 * Types & Variables
 */

// Variables begin with the $ symbol.
// A valid variable name starts with a letter or underscore,
// followed by any number of letters, numbers, or underscores.

// Boolean values are case-insensitive
$boolean = true;  // or TRUE or True
$boolean = false; // or FALSE or False

// Integers
$int1 = 12;   // => 12
$int2 = -12;  // => -12
$int3 = 012;  // => 10 (a leading 0 denotes an octal number)
$int4 = 0x0F; // => 15 (a leading 0x denotes a hex literal)
// Binary integer literals are available since PHP 5.4.0.
$int5 = 0b11111111; // 255 (a leading 0b denotes a binary number)

// Floats (aka doubles)
$float = 1.234;
$float = 1.2e3;
$float = 7E-10;
```

```php
// Delete variable
unset($int1);

// Arithmetic
$sum        = 1 + 1; // 2
$difference = 2 - 1; // 1
$product    = 2 * 2; // 4
$quotient   = 2 / 1; // 2

// Shorthand arithmetic
$number = 0;
$number += 1;      // Increment $number by 1
echo $number++;    // Prints 1 (increments after evaluation)
echo ++$number;    // Prints 3 (increments before evaluation)
$number /= $float; // Divide and assign the quotient to $number

// Strings should be enclosed in single quotes;
$sgl_quotes = '$String'; // => '$String'

// Avoid using double quotes except to embed other variables
$dbl_quotes = "This is a $sgl_quotes."; // => 'This is a $String.'

// Special characters are only escaped in double quotes
$escaped   = "This contains a \t tab character.";
$unescaped = 'This just contains a slash and a t: \t';

// Enclose a variable in curly braces if needed
$money = "I have $${number} in the bank.";

// Since PHP 5.3, nowdocs can be used for uninterpolated multi-liners
$nowdoc = <<<'END'
Multi line
string
END;

// Heredocs will do string interpolation
$heredoc = <<<END
Multi line
$sgl_quotes
END;

// String concatenation is done with .
echo 'This string ' . 'is concatenated';

// Strings can be passed in as parameters to echo
echo 'Multiple', 'Parameters', 'Valid';  // Returns 'MultipleParametersValid'


/********************************
 * Constants
 */

// A constant is defined by using define()
// and can never be changed during runtime!
```

```php
// a valid constant name starts with a letter or underscore,
// followed by any number of letters, numbers, or underscores.
define("FOO", "something");

// access to a constant is possible by calling the choosen name without a $
echo FOO; // Returns 'something'
echo 'This outputs ' . FOO;  // Returns 'This ouputs something'



/********************************
 * Arrays
 */

// All arrays in PHP are associative arrays (hashmaps),

// Associative arrays, known as hashmaps in some languages.

// Works with all PHP versions
$associative = array('One' => 1, 'Two' => 2, 'Three' => 3);

// PHP 5.4 introduced a new syntax
$associative = ['One' => 1, 'Two' => 2, 'Three' => 3];

echo $associative['One']; // prints 1

// List literals implicitly assign integer keys
$array = ['One', 'Two', 'Three'];
echo $array[0]; // => "One"

// Add an element to the end of an array
$array[] = 'Four';
// or
array_push($array, 'Five');

// Remove element from array
unset($array[3]);

/********************************
 * Output
 */

echo('Hello World!');
// Prints Hello World! to stdout.
// Stdout is the web page if running in a browser.

print('Hello World!'); // The same as echo

// echo and print are language constructs too, so you can drop the parentheses
echo 'Hello World!';
print 'Hello World!';

$paragraph = 'paragraph';
```

```php
echo 100;        // Echo scalar variables directly
echo $paragraph; // or variables

// If short open tags are configured, or your PHP version is
// 5.4.0 or greater, you can use the short echo syntax
?>
<p><?= $paragraph ?></p>
<?php

$x = 1;
$y = 2;
$x = $y; // $x now contains the same value as $y
$z = &$y;
// $z now contains a reference to $y. Changing the value of
// $z will change the value of $y also, and vice-versa.
// $x will remain unchanged as the original value of $y

echo $x; // => 2
echo $z; // => 2
$y = 0;
echo $x; // => 2
echo $z; // => 0

// Dumps type and value of variable to stdout
var_dump($z); // prints int(0)

// Prints variable to stdout in human-readable format
print_r($array); // prints: Array ( [0] => One [1] => Two [2] => Three )

/********************************
 * Logic
 */
$a = 0;
$b = '0';
$c = '1';
$d = '1';

// assert throws a warning if its argument is not true

// These comparisons will always be true, even if the types aren't the same.
assert($a == $b); // equality
assert($c != $a); // inequality
assert($c <> $a); // alternative inequality
assert($a < $c);
assert($c > $b);
assert($a <= $b);
assert($c >= $d);

// The following will only be true if the values match and are the same type.
assert($c === $d);
assert($a !== $d);
assert(1 === '1');
assert(1 !== '1');
```

```php
// 'Spaceship' operator (since PHP 7)
// Returns 0 if values on either side are equal
// Returns 1 if value on the left is greater
// Returns -1 if the value on the right is greater

$a = 100;
$b = 1000;

echo $a <=> $a; // 0 since they are equal
echo $a <=> $b; // -1 since $a < $b
echo $b <=> $a; // 1 since $b > $a

// Variables can be converted between types, depending on their usage.

$integer = 1;
echo $integer + $integer; // => 2

$string = '1';
echo $string + $string; // => 2 (strings are coerced to integers)

$string = 'one';
echo $string + $string; // => 0
// Outputs 0 because the + operator cannot cast the string 'one' to a number

// Type casting can be used to treat a variable as another type

$boolean = (boolean) 1; // => true

$zero = 0;
$boolean = (boolean) $zero; // => false

// There are also dedicated functions for casting most types
$integer = 5;
$string = strval($integer);

$var = null; // Null value


/********************************
 * Control Structures
 */

if (true) {
    print 'I get printed';
}

if (false) {
    print 'I don\'t';
} else {
    print 'I get printed';
}

if (false) {
```

```
    print 'Does not get printed';
} elseif(true) {
    print 'Does';
}

// ternary operator
print (false ? 'Does not get printed' : 'Does');

// ternary shortcut operator since PHP 5.3
// equivalent of "$x ? $x : 'Does'""
$x = false;
print($x ?: 'Does');

// null coalesce operator since php 7
$a = null;
$b = 'Does print';
echo $a ?? 'a is not set'; // prints 'a is not set'
echo $b ?? 'b is not set'; // prints 'Does print'


$x = 0;
if ($x === '0') {
    print 'Does not print';
} elseif($x == '1') {
    print 'Does not print';
} else {
    print 'Does print';
}



// This alternative syntax is useful for templates:
?>

<?php if ($x): ?>
This is displayed if the test is truthy.
<?php else: ?>
This is displayed otherwise.
<?php endif; ?>

<?php

// Use switch to save some logic.
switch ($x) {
    case '0':
        print 'Switch does type coercion';
        break; // You must include a break, or you will fall through
               // to cases 'two' and 'three'
    case 'two':
    case 'three':
        // Do something if $variable is either 'two' or 'three'
        break;
    default:
        // Do something by default
```

```
}

// While, do...while and for loops are probably familiar
$i = 0;
while ($i < 5) {
    echo $i++;
}; // Prints "01234"

echo "\n";

$i = 0;
do {
    echo $i++;
} while ($i < 5); // Prints "01234"

echo "\n";

for ($x = 0; $x < 10; $x++) {
    echo $x;
} // Prints "0123456789"

echo "\n";

$wheels = ['bicycle' => 2, 'car' => 4];

// Foreach loops can iterate over arrays
foreach ($wheels as $wheel_count) {
    echo $wheel_count;
} // Prints "24"

echo "\n";

// You can iterate over the keys as well as the values
foreach ($wheels as $vehicle => $wheel_count) {
    echo "A $vehicle has $wheel_count wheels";
}

echo "\n";

$i = 0;
while ($i < 5) {
    if ($i === 3) {
        break; // Exit out of the while loop
    }
    echo $i++;
} // Prints "012"

for ($i = 0; $i < 5; $i++) {
    if ($i === 3) {
        continue; // Skip this iteration of the loop
    }
    echo $i;
} // Prints "0124"
```

```php
/*******************************
 * Functions
 */

// Define a function with "function":
function my_function () {
    return 'Hello';
}

echo my_function(); // => "Hello"

// A valid function name starts with a letter or underscore, followed by any
// number of letters, numbers, or underscores.

function add ($x, $y = 1) { // $y is optional and defaults to 1
    $result = $x + $y;
    return $result;
}

echo add(4); // => 5
echo add(4, 2); // => 6

// $result is not accessible outside the function
// print $result; // Gives a warning.

// Since PHP 5.3 you can declare anonymous functions;
$inc = function ($x) {
    return $x + 1;
};

echo $inc(2); // => 3

function foo ($x, $y, $z) {
    echo "$x - $y - $z";
}

// Functions can return functions
function bar ($x, $y) {
    // Use 'use' to bring in outside variables
    return function ($z) use ($x, $y) {
        foo($x, $y, $z);
    };
}

$bar = bar('A', 'B');
$bar('C'); // Prints "A - B - C"

// You can call named functions using strings
$function_name = 'add';
echo $function_name(1, 2); // => 3
// Useful for programatically determining which function to run.
// Or, use call_user_func(callable $callback [, $parameter [, ... ]]);
```

```php
// You can get the all the parameters passed to a function
function parameters() {
    $numargs = func_num_args();
    if ($numargs > 0) {
        echo func_get_arg(0) . ' | ';
    }
    $args_array = func_get_args();
    foreach ($args_array as $key => $arg) {
        echo $key . ' - ' . $arg . ' | ';
    }
}

parameters('Hello', 'World'); // Hello | 0 - Hello | 1 - World |

// Since PHP 5.6 you can get a variable number of arguments
function variable($word, ...$list) {
    echo $word . " || ";
    foreach ($list as $item) {
        echo $item . ' | ';
    }
}

variable("Separate", "Hello", "World") // Separate || Hello | World |

/*******************************
 * Includes
 */

<?php
// PHP within included files must also begin with a PHP open tag.

include 'my-file.php';
// The code in my-file.php is now available in the current scope.
// If the file cannot be included (e.g. file not found), a warning is emitted.

include_once 'my-file.php';
// If the code in my-file.php has been included elsewhere, it will
// not be included again. This prevents multiple class declaration errors

require 'my-file.php';
require_once 'my-file.php';
// Same as include(), except require() will cause a fatal error if the
// file cannot be included.

// Contents of my-include.php:
<?php

return 'Anything you like.';
// End file

// Includes and requires may also return a value.
$value = include 'my-include.php';
```

```php
// Files are included based on the file path given or, if none is given,
// the include_path configuration directive. If the file isn't found in
// the include_path, include will finally check in the calling script's
// own directory and the current working directory before failing.
/* */

/********************************
 * Classes
 */

// Classes are defined with the class keyword

class MyClass
{
    const MY_CONST      = 'value'; // A constant

    static $staticVar   = 'static';

    // Static variables and their visibility
    public static $publicStaticVar = 'publicStatic';
    // Accessible within the class only
    private static $privateStaticVar = 'privateStatic';
    // Accessible from the class and subclasses
    protected static $protectedStaticVar = 'protectedStatic';

    // Properties must declare their visibility
    public $property    = 'public';
    public $instanceProp;
    protected $prot = 'protected'; // Accessible from the class and subclasses
    private $priv   = 'private';   // Accessible within the class only

    // Create a constructor with __construct
    public function __construct($instanceProp) {
        // Access instance variables with $this
        $this->instanceProp = $instanceProp;
    }

    // Methods are declared as functions inside a class
    public function myMethod()
    {
        print 'MyClass';
    }

    //final keyword would make a function unoverridable
    final function youCannotOverrideMe()
    {
    }

/*
 * Declaring class properties or methods as static makes them accessible without
 * needing an instantiation of the class. A property declared as static can not
 * be accessed with an instantiated class object (though a static method can).
 */
```

```
    public static function myStaticMethod()
    {
        print 'I am static';
    }
}

// Class constants can always be accessed statically
echo MyClass::MY_CONST;    // Outputs 'value';

echo MyClass::$staticVar;  // Outputs 'static';
MyClass::myStaticMethod(); // Outputs 'I am static';

// Instantiate classes using new
$my_class = new MyClass('An instance property');
// The parentheses are optional if not passing in an argument.

// Access class members using ->
echo $my_class->property;     // => "public"
echo $my_class->instanceProp; // => "An instance property"
$my_class->myMethod();        // => "MyClass"


// Extend classes using "extends"
class MyOtherClass extends MyClass
{
    function printProtectedProperty()
    {
        echo $this->prot;
    }

    // Override a method
    function myMethod()
    {
        parent::myMethod();
        print ' > MyOtherClass';
    }
}

$my_other_class = new MyOtherClass('Instance prop');
$my_other_class->printProtectedProperty(); // => Prints "protected"
$my_other_class->myMethod();               // Prints "MyClass > MyOtherClass"

final class YouCannotExtendMe
{
}

// You can use "magic methods" to create getters and setters
class MyMapClass
{
    private $property;

    public function __get($key)
    {
        return $this->$key;
```

```
    }

    public function __set($key, $value)
    {
        $this->$key = $value;
    }
}

$x = new MyMapClass();
echo $x->property; // Will use the __get() method
$x->property = 'Something'; // Will use the __set() method

// Classes can be abstract (using the abstract keyword) or
// implement interfaces (using the implements keyword).
// An interface is declared with the interface keyword.

interface InterfaceOne
{
    public function doSomething();
}

interface InterfaceTwo
{
    public function doSomethingElse();
}

// interfaces can be extended
interface InterfaceThree extends InterfaceTwo
{
    public function doAnotherContract();
}

abstract class MyAbstractClass implements InterfaceOne
{
    public $x = 'doSomething';
}

class MyConcreteClass extends MyAbstractClass implements InterfaceTwo
{
    public function doSomething()
    {
        echo $x;
    }

    public function doSomethingElse()
    {
        echo 'doSomethingElse';
    }
}


// Classes can implement more than one interface
class SomeOtherClass implements InterfaceOne, InterfaceTwo
{
```

```
    public function doSomething()
    {
        echo 'doSomething';
    }

    public function doSomethingElse()
    {
        echo 'doSomethingElse';
    }
}


/*******************************
 * Traits
 */

// Traits are available from PHP 5.4.0 and are declared using "trait"

trait MyTrait
{
    public function myTraitMethod()
    {
        print 'I have MyTrait';
    }
}

class MyTraitfulClass
{
    use MyTrait;
}

$cls = new MyTraitfulClass();
$cls->myTraitMethod(); // Prints "I have MyTrait"


/*******************************
 * Namespaces
 */

// This section is separate, because a namespace declaration
// must be the first statement in a file. Let's pretend that is not the case

<?php

// By default, classes exist in the global namespace, and can
// be explicitly called with a backslash.

$cls = new \MyClass();



// Set the namespace for a file
namespace My\Namespace;
```

```php
class MyClass
{
}

// (from another file)
$cls = new My\Namespace\MyClass;

//Or from within another namespace.
namespace My\Other\Namespace;

use My\Namespace\MyClass;

$cls = new MyClass();

// Or you can alias the namespace;

namespace My\Other\Namespace;

use My\Namespace as SomeOtherNamespace;

$cls = new SomeOtherNamespace\MyClass();


/**********************
* Late Static Binding
*
*/

class ParentClass {
    public static function who() {
        echo "I'm a " . __CLASS__ . "\n";
    }
    public static function test() {
        // self references the class the method is defined within
        self::who();
        // static references the class the method was invoked on
        static::who();
    }
}

ParentClass::test();
/*
I'm a ParentClass
I'm a ParentClass
*/

class ChildClass extends ParentClass {
    public static function who() {
        echo "But I'm " . __CLASS__ . "\n";
    }
}

ChildClass::test();
/*
```

```
I'm a ParentClass
But I'm ChildClass
*/

/**********************
*  Magic constants
*
*/

// Get current class name. Must be used inside a class declaration.
echo "Current class name is " . __CLASS__;

// Get full path directory of a file
echo "Current directory is " . __DIR__;

    // Typical usage
    require __DIR__ . '/vendor/autoload.php';

// Get full path of a file
echo "Current file path is " . __FILE__;

// Get current function name
echo "Current function name is " . __FUNCTION__;

// Get current line number
echo "Current line number is " . __LINE__;

// Get the name of the current method. Only returns a value when used inside a trait or object declarati
echo "Current method is " . __METHOD__;

// Get the name of the current namespace
echo "Current namespace is " . __NAMESPACE__;

// Get the name of the current trait. Only returns a value when used inside a trait or object declaratio
echo "Current namespace is " . __TRAIT__;

/**********************
*  Error Handling
*
*/

// Simple error handling can be done with try catch block

try {
    // Do something
} catch (Exception $e) {
    // Handle exception
}

// When using try catch blocks in a namespaced enviroment use the following

try {
    // Do something
} catch (\Exception $e) {
```

```
    // Handle exception
}

// Custom exceptions

class MyException extends Exception {}

try {

    $condition = true;

    if ($condition) {
        throw new MyException('Something just happend');
    }

} catch (MyException $e) {
    // Handle my exception
}
```

More Information

Visit the official PHP documentation for reference and community input.

If you're interested in up-to-date best practices, visit PHP The Right Way.

If you're coming from a language with good package management, check out Composer.

For common standards, visit the PHP Framework Interoperability Group's PSR standards.