**Echo**

The echo function outputs strings. If you type

<?php

echo "Hello!";

?>

PHP will output Hello!.

Make sure to end your line of PHP code with a semicolon.

**Variables**

So far we've been outputting strings and doing math.

To do more complex coding, we need a way to "save" these values. We can do this using **variables**. A **variable**can store a string or a number, and gives it a specific case-senstive name.

**Examples:**

* $myName = "Beyonce";
* $myAge = 32;

All variable names in PHP start with a dollar sign ( $ ).

**Semicolons**

You've probably noticed that our lines of PHP code end in semicolons (;). PHP requires semicolons at the end of each **statement**, which is the shortest unit of standalone code. (For example,echo "Hello!"; or 2 + 2;)

**Comments**

Just like we sometimes put comments in our CSS (using /\* this syntax \*/) or in our HTML (using <!-- this syntax -->), we can also put comments in our PHP code! We do that using two forward slashes (//), like so:

<?php

echo "I get printed!";

*// I don't! I'm a comment.*

?>

**Echo It!**

Nice work! PHP computed the value for you, but it didn't appear in your .php document because we didn't print it using echo. Let's fix that!

Let's finish this up! Beneath your existing PHP code, use echo to print out your name and your age, like so:

echo $myName;

echo $myAge;

IF&ELSE

$name = "Edgar";

if ($name == "Simon") {

print "I know you!";

}

else {

print "Who are you?";

}

?>

SWITCH

<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

<?php

$myNum = 2;

switch ($myNum) {

case 1:

echo "1";

break;

case 2:

echo "2";

break;

case 3:

echo "3";

break;

default:

echo "None of the above";

}

?>

</body>

</html>

**Multiple Cases. Falling Through!**

You sometimes want to make multiple expressions, all of which have the same result. Consider the followingif statement:

if ($i == 1 ||

$i == 2 ||

$i == 3) {

echo '$i is somewhere between 1 and 3.';

}

With a switch statement, you can do this by adding cases right after another without a break. This is called **falling through**. The following code works exactly like the above if statement:

case 1:

case 2:

case 3:

echo '$i is somewhere between 1 and 3.';

break;

**Using "Endswitch". Syntactic Sugar!**

You have two ways of creating a switch. First, there's the way we have made all the past exercises:

switch ($i) {

}

But we can also make it this way:

switch ($i):

endswitch;

This is called **alternative syntax**. It exists to provide [syntactic sugar](http://en.wikipedia.org/wiki/Syntactic_sugar)

Therefore, we can access a particular item of the array using its position, like this:

<?php

$myArray = array("red", "blue", "yellow");

echo $myArray[1];

*// outputs "blue"*

$myArray[1] = "green";

echo $myArray[1];

*// outputs "green"*

?>

Finally, you can remove elements using unset:

<?php

$array = array("red", "blue", "green");

unset($array[2]);

?>

You can even delete the whole array:

<?php

unset($array);

?>

**'For' Loop Syntax**

<?php

for ($i = 0; $i < 10; $i++) {

echo $i;

}

*// echoes 0123456789*

?>

**Loops + Arrays = ForEach**

The foreach loop is used to iterate over each element of an object—which makes it perfect for use with arrays!

You can think of foreach as jumping from element to element in the array and running the code between {}s for each of those elements.

Between the parentheses, we use the ($langs as $lang) syntax to tell PHP: "For each thing in $langs, assign that thing temporarily to the variable $lang."

<?php

$langs = array("JavaScript", "HTML/CSS", "PHP", "Python", "Ruby");

foreach ($langs as $lang) {

echo "<li>$lang</li>";//output in ranked order.If<p>……</p>,output in rows.

}

unset($lang);

?>

**While Loop Syntax**

while(cond) {

*// looped statements go here*

}

where the statements in side the curly braces { and } are executed as long as the condition cond is evaluated as true.

PHP offers the following alternative syntax for while loops:

while(cond):

*// looped statements go here*

endwhile;

for,while: first check condition, then execute. A do/while loop check the condition after each iteration.

<?php

$i = 0;

do {

echo $i;

} while ($i > 0);

?>

strlen() You pass it a string, or variable containing a string, and it returns the number of characters in that string.

<?php

*// get the length of a string and*

*// print it to the screen*

$length = strlen("david");

print $length;

?>

substr(). This function allows you to return a substring (piece of) of your string.

You pass this function the string you want to get a substring of, the character in your string to start at, and how many characters you want after your starting point. An example might be:

$myname = "David";

*// you can manipulate strings easily*

*// with built-in funtions too*

$partial = substr($myname, 0, 3);

print $partial;

*// prints "dav"*

NOTE: the second parameter (the starting character) is based on a zero-indexed array (i.e. the first character in your string is number 0, not number 1).

Two other very useful string functions are strtoupper() and strtolower(), which make your entire stringUPPERCASE or lowercase. Here is an example of each:

$uppercase = strtoupper($myname);

print $uppercase;

*// prints "DAVID"*

$lowercase = strtolower($uppercase);

print $lowercase;

*// prints "david"*

You can also call these functions on a string directly, like so:

print strtolower("David");

*// prints "david"*

strpos() find the position of the first occurrence of a substring in a string.

strpos("emily", "e"); *// 0*

strpos("emily", "i"); *// 2*

strpos("emily", "ily"); *// 2*

strpos("emily", "zxc"); *// false*

if (strpos("david","h") === false) {

print "Sorry, no 'h' in 'david'";

}

*// prints the "Sorry" message*

You can use round() to round your number to an integer, or to round off complex floating point numbers to a specific number of decimal places. This is accomplished by passing a second, optional parameter to round(), telling it how many decimal places you want the number rounded to. 保留几位小数，默认0

*// Round pi down from 3.1416...*

$round = round(M\_PI);

print $round; *// prints 3*

*// This time, round pi to 4 places*

$round\_decimal = round(M\_PI, 4);

print $round\_decimal; *// prints 3.1416*

NOTE: M\_PI is a PHP constant that is equal to pi.

A very common and useful function is rand(). This function returns a random number between two numbers. Optionally, you can provide your min and max numbers as parameters, like this:

*// prints a number between 0 and 32767*

print rand();

*// prints a number between 1 and 10*

print rand(1,10);

array\_push() takes two arguments: an array, and an element to add to the end of that array. Here's an example:

$fav\_bands = array();

array\_push($fav\_bands, "Maroon 5");

array\_push($fav\_bands, "Bruno Mars");

Passing an array to count()will return the number of elements in that array. Like this:

print count($fav\_bands); *// prints 2*

PHP has a sort() function for just such an occasion!

$array = array(5, 3, 7, 1);

sort($array);

print join(", ", $array);

*// prints "1, 3, 5, 7"*

PHP also has the opposite function:rsort().

$array = array(5, 3, 7 ,1);

rsort($array);

print join(":", $array);

*// prints "7:5:3:1"*

The typical structure of a function is as follows:

function name(parameters) {

statement;

}

<?php

// Here we define the function...

function helloWorld() {

echo "Hello world!";

}

// ...and here we call it!

helloWorld();

?>

In PHP, the return keyword does just that. It returns to us a value that we can work with. The difference between this and echo or print is that it doesn't actually display the value.(use print/echo to show the value).

function squareValue($number) {

echo $number \* $number;

}

$n = 6;

squareValue($n); *// echos 36*

PHP is an **object-oriented programming language**, which means that you can create objects, which can contain variables and functions.

When talking about **objects**, you refer to variables belonging to these objects as **properties** (or attributes or fields), and functions are called **methods**.

The basic class syntax looks like the following:

class Classname {

}

And you can create new instances of this class using the following syntax:

$obj1 = new Classname();

As you remember, properties are pieces of data bound to an object, and you can imagine an object as a bundle of information and actions.

class Fruit {

public $count = 3;

public $type;

}

$apple = new Fruit();

$apple->type = "apple";

print $apple->count; *// 3*

print $apple->type; *// apple*

Then we set the $type property of$apple to the string "apple".

Finally, we print out the two properties of $apple.

class Classname {

public $prop1 = true;

public $prop2;

public $prop3;

public $prop4;

public function \_\_construct($prop2, $prop3, $prop4) {

$this->prop2 = $prop2;//transfer values

$this->prop3 = $prop3;

$this->prop4 = $prop4;

}

public function meth1() {

return "A long sentence containing some properties.";

}

}

//outside the class

$obj1 = new Classname("tired", "54321", 54321);//instance

$obj2 = new Classname("your", "name", 99);

echo $obj1->prop1;

echo $obj2->prop4;

echo $obj1->meth1();

echo $obj2->meth1();

//calling a method

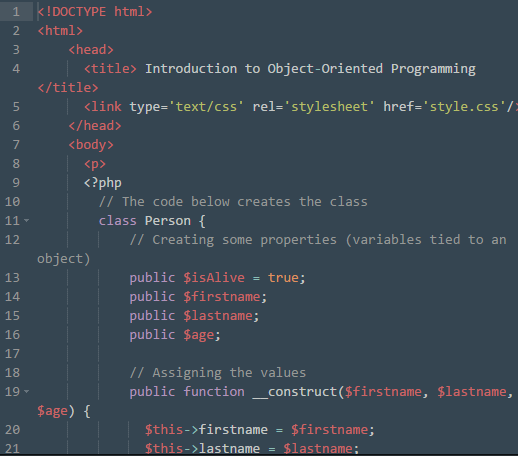
 \_\_construct function is a special one, which is called when a new object is created using a new keyword.

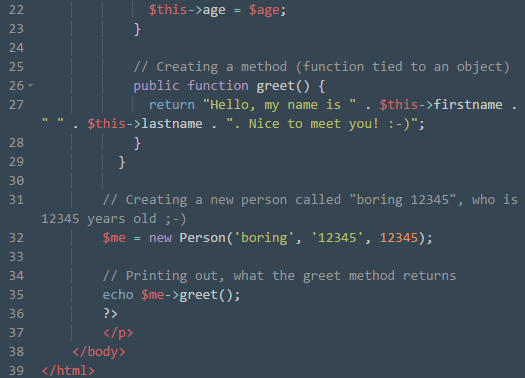
Finally, the weird way to assign the values: $this->prop1 = $prop1 means that the value you pass in the \_\_constructor() function via the new keyword is assigned to $this, which represents the object you are dealing with, and ->prop1 is the actual property of the object. we have to use the $this keyword, if we want to access some properties in a class.

public function meth2() {

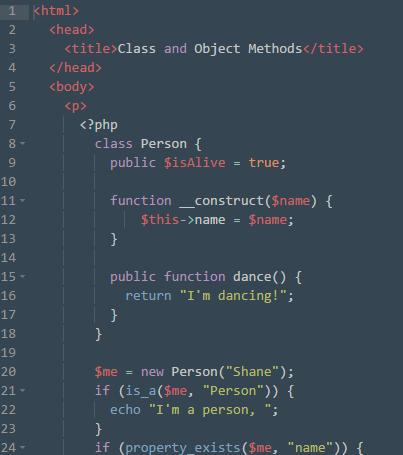
return "Some text " . $this->name . " some further text";

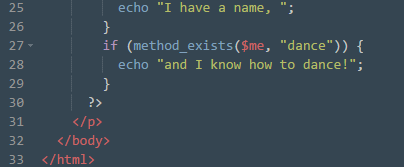
}



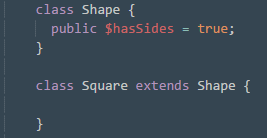


We're demoing three useful built-in methods: is\_a(), which we use to find out if a particular object is an instance of a given class; property\_exists(), to see if an object has a given property; and method\_exists(), to see if an object has a given method.





Inheritance is a way for one class to take on the properties or methods of another class. We can cause one PHP class to inherit from another with the extends keyword.



Sometimes we want a child class (or**subclass**) to be able to override a property or method of its parent class (or **superclass**).

For instance, we might have a Shapeclass with a $sides property set totrue, but we might want Square to override this property and set $sidesto 4 (since a square always has four sides). That would look something like this:

class Shape {

$sides = true;

}

class Square extends Shape {

$sides = 4;

}

It's pretty easy—you just create a new property or method in the child class with the same name as the one in the parent class, and the child's version will always take precedence over the inherited version.

In PHP, a parent class can prevent its methods from being overridden by its children with—you guessed it—the final keyword.

variables that don't change. These are prefixed with the const keyword (short for **constant**).

PHP lets us set constants on a class-by-class basis! Each class has its own **scope**, which is the context in which its variables can be used.

class Immortal extends Person {

*// Immortals never die!*

const alive = true;

}

*// If true...*

if (Immortal::alive) {

echo "I live forever!";

}

*// echoes "I live forever!"*

In the example above, we use :: to access the alive constant inside the Immortal class.

Note that constants **do not** start with $.

The static keyword lets you use a class' property or method without having to create an instance of that class. It works like this:

class Person {

public static $isAlive = "Yep!"

public static function greet() {

echo "Hello there!";

}

}

echo Person::$isAlive;

*// prints "Yep!"*

Person::greet();

*// prints "Hello there!"*

Here we create an array of three numbers. Remember that array positions start from 0; therefore to access the third item in this array, we type:

$myArray = array(1, 2, 3);

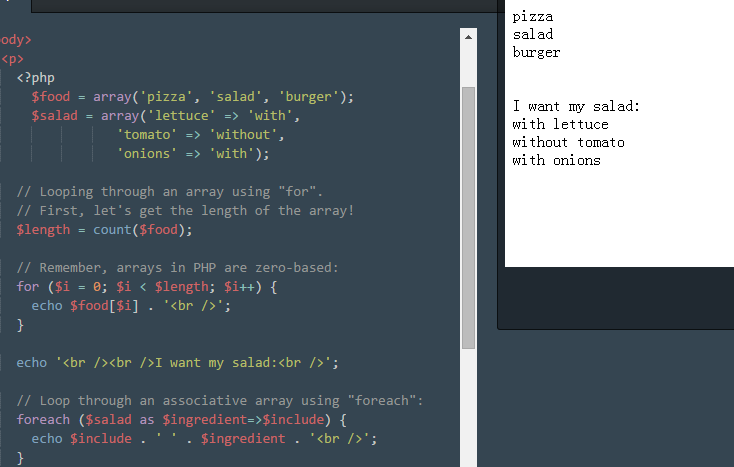
$myArray[2];

An associative array makes use of (key => value) pairs. Both arrays contain the same values, but in the associative array, we can access the value using a specified "key".

echo '<br />'; //换行



You can think of an associative array (also called a **map**) as being the same as a normal array, but instead of using an integer to refer to the value, you use a defined key.





array\_push($deck, array(‘4 of Diamonds’, 4));

$deck is an array which contains 3 rows, each being a playing card. To retrieve a card, we would first get the row for that card, then get the value we require. If we access $deck[2], we would get the third row. That will return another array containing 2 values: the first (0) which is a string that has the value "7 of Diamonds", and the second (1) which is an integer that has the value 7.

//return “Array” in fact.

If we want the "7 of Diamonds" string, we would simply use $deck[2][0];.

An array can contain values ("val1", "val2", etc... )

As an associative array, it can contain keys **and** values ("key1" => "val1", "key2" => "val2", etc... )

A multidimensional array can contain arrays within arrays!

To loop through an array containing values only, you can use the for loop. For an array containing keys and values, you can use the foreach loop.