PHP

**PHP in Action**

PHP is a programming language that can do all sorts of things: evaluate form data sent from a browser, build custom web content to serve the browser, talk to a database, and even send and receive cookies (little packets of data that your browser uses to remember things, like if you're logged in to Codecademy).

Check out the code in the editor. Looks familiar, doesn't it? That's because a lot of it is regular old HTML! The PHP code is written in the <?php and ?> . See how it generates numbers, creates lists, and adds text directly to your webpage?

**Why Learn PHP?**

"So what?" You might say. "I can do that with [JavaScript](http://www.codecademy.com/tracks/javascript)." And that's true! But JavaScript's knowledge can be limited.

JavaScript generally runs in the browser, or **client**. This means it only really knows what's going on in your browser, plus whatever information it gets from the website(s) you're connecting to.

PHP, on the other hand, runs on the same computer as the website you're visiting, which is known as the **server**. This means that it has access to all the information and files on that machine, which allows it to construct custom HTML pages to send to your browser, handle cookies, and run tasks or perform calculations with data from that website.

We've written a little PHP in the editor to the right, but it's not complete! On [line 8](javascript:void(0)), type My first line of PHP! between the ""s

<!DOCTYPE html>

<html>

<head>

</head>

<body>

<p>

<?php

echo "My first line of PHP!";

?>

</p>

</body>

</html>

**PHP and HTML**

PHP code can be written right into your HTML, like this:

<body>

<p>

*<?php*

*echo "I'm learning PHP!";*

*?>*

</p>

</body>

Your PHP code goes inside the <?php and ?> delimiters. Here we use the function echo to output I'm learning PHP!. We also end the line with a semicolon.

**PHP Files**

You might have noticed that our main file is now index.php instead of index.html. This is important! It tells the PHP interpreter that there's PHP code in the file to evaluate.

**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.

**Strings**

A string is a word or phrase between quotes, like so: "Hello, world!"

You can type a string all at once, like this:

<?php

echo "Hello, world!";

?>

Or use the **concatenation operator**, which glues several strings together:

<?php

echo "Hello," . " " . "world" . "!";

?>

The concatenation operator is just a dot (.). (If you're coming to PHP from JavaScript, the dot does the same thing for strings that + does in JavaScript.)

**Arithmetic**

In addition to outputting strings, PHP can also do math.

<?php

echo 5 \* 7;

?>

Here we use echo to multiply 5 and 7, and we end our line of code with a semicolon. PHP will output 35.

**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;)

You can think of a statement is a complete PHP thought. 19 + or echo aren't complete thoughts, so you wouldn't put semicolons at the end of them!

<?php echo "Use your semicolons!"; ?>

**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.*

?

**Conditionals and Control Flow**

**Comparisons**

So far we've seen:

* **strings** (*e.g.* "dogs go woof!")
* **numbers** (*e.g.* 4, 10)

Now let's learn about comparison operators.

List of comparison operators:

* > Greater than
* < Less than
* <= Less than or equal to
* >= Greater than or equal to
* == Equal to
* != Not equal to

**If statements**

Nice work on comparisons! Now let's see how we can use comparisons to ask yes or no questions.

Say we want to write a program that asks whether your name is longer than 7 letters. If the answer is yes, we can respond with "You have a long name!" We can do this with an if statement:

<?php

$age = 17;

if( $age > 16 ) {

echo "You can drive!";

}

?>

An if statement is made up of the ifkeyword, a condition like we've seen before, and a pair of curly braces { }. If the answer to the condition is yes, the code inside the curly braces will run.

**Adding an Else**

Great! We used an if statement to do something if the answer to the condition was yes, or true as we say in PHP.

In addition to doing something when the condition is true, we can do something else if the condition is false. We can do this using an if / else statement:

<?php

$name = "Edgar";

if ($name == "Simon") {

print "I know you!";

}

else {

print "Who are you?";

}

?>

Just like before, *if* the condition is true, then only the code inside the first pair of curly braces will run. *Otherwise*, the condition is false, so only the code inside the second pair of curly braces after the else keyword will run.

In the example above the condition $name== "Simon" evaluates to false since $nameis Edgar. Since the condition is false, only the code inside the curly braces after the else keyword runs, and prints out Who are you?

**Control Flow: Switch**

**Dipping Your Toe into Switch statements**

In the editor you see a simple switch statement. A switch statement comes in handy when you have a series of if/elseif/else statements with multiple expressions that depend on the same value. The switch statement also provides a bit of efficiency and readability. Switches work like if statements, if a condition is true, it executes a block of code.

<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

<?php

switch (2) {

case 0:

echo 'The value is 0';

break;

case 1:

echo 'The value is 1';

break;

case 2:

echo 'The value is 2';

break;

default:

echo "The value isn't 0, 1 or 2";

}

?>

</body>

</html>

**Switch Syntax**

A switch statement is similar to an if / elif / else statement in that you can check multiple conditions. Here's what it looks like:

$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";

}

1. A switch statement is made up of the switch keyword, a variable to check, and a pair of curly braces { }. Here we check the value of $myNum.
2. Then we have a case block for each comparison. For example case 1: echo "1";break; checks whether $myNum is equal to 1. If yes, it echos "1", and uses break to exit the switch statement.
3. Otherwise, the next case block runs.
4. If all cases return false, the defaultcase gets executed.

**Multiple Cases. Falling Through!**

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

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)

There's no difference when using either the **curly brace syntax** (*first example*) or **alternative syntax** (*second example*), it only provides readability, thus it's usually used when mixing HTML and PHP code in the same file.

<!DOCTYPE html>

<html>

<head>

<title></title>

</head>

<body>

<?php

$i = 5;

switch ($i):

case 0:

echo '$i is 0.';

break;

case 1:

case 2:

case 3:

echo "haha";

break;

case 4:

echo "dada";

break;

case 5:

echo '$i is somewhere between 1 and 5.';

break;

case 6:

case 7:

echo '$i is either 6 or 7.';

break;

default:

echo "I don't know how much \$i is.";

endswitch;

?>

</body>

</html>

**Arrays**

**What's an Array?**

An array is a list of items, a bit like a shopping list. It allows you to store more than one item in only one variable.

Think of it like this. When writing your shopping list, you could use a separate piece of paper for each item you need to buy (a variable). However this is silly and unneeded—could you imagine how hard it would be to carry all that paper around with you? So, you use one piece of paper for all of your items. This one piece of paper is your array.

In the editor do you see the bit of text that starts with $array =? That is our array. Don't worry about all the details just yet, we will explain in more detail later. For now, just see if you can work out what is happening.

**Array Syntax**

Have you noticed something familiar at the start of our array? That's right, it starts in the same way as a variable, with the $sign, and then a name, followed by =.

However, this is when things start to get different. When declaring an array, we have to use array(). This basically tells PHP that $array is an array and not something else, such as a regular old variable.

By now, I am sure you have noticed the text inside the ( and ). This is just the items in our array. So, currently, our array has the items "Egg," "Tomato," and "Beans" in it. You can add any type of information to an array, and you do it in much the same way as when declaring variables. Use "" when adding strings, and just enter the number when adding integers.

You must always remember, however, that each item in an array *must* be separated by a comma: ,.

<html>

<head>

<title>My First Array</title>

</head>

<body>

<p>

<?php

$friends=array("A","B","C");

?>

</p>

</body>

</html>

**Access by Offset with [ ]**

Each item in an array is numbered starting from 0. For example, when we create an array:

<?php

$myArray = array("do", "re", "mi");

?>

Each item is numbered starting from 0, like this:

+------+------+------+

| "do" | "re" | "mi" |

+------+------+------+

0 1 2

The item "do" is in position 0, the item "re" is in position 1, and so on.

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

<?php

$myArray = array("do", "re", "mi");

echo $myArray[0]

*// outputs "do"*

?>

1. First we create an array named $myArray
2. Then we use echo to output the first item in $myArray. Since items are numbered starting from 0, "do" is at position 0.

**Access by Offset with { }**

PHP is a very flexible language. When accessing arrays by offset, you can actually use two different types of syntax: the []syntax we've covered, or you can use curly braces ({}). You use the curly braces just like you use the square brackets:

<?php

$myArray = array("do", "re", "mi");

print $myArray{2};

*// prints "mi";*

?>

Both forms are equivalent, and using one or the other is totally up to you!

**Modifying Array Elements**

An item in an array can be changed by specifying its position and providing a new value, like this:

<?php

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

echo $myArray[1];

*// outputs "blue"*

$myArray[1] = "green";

echo $myArray[1];

*// outputs "green"*

?>

1. First we create a new array $myArraywith a list of colors.
2. Then we output the item at position 1. Since items are numbered starting from 0, "blue" is at position 1
3. Next we change the item at position 1 to "green".
4. Now when we output the item at position 1, we get "green".

**Deleting Array Elements**

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 Loops in PHP**

**What's a Loop?**

Programming can be tough work, and sometimes it's made tougher by having to do the same thing over and over. Let's say we want to echo a list of [leap years](http://en.wikipedia.org/wiki/Leap_year) in the editor. You might think we'd have to type:

<?php

echo 2004;

echo 2008;

echo 2012;

*// And so on*

?>

But there's a better way!

A **loop** is a useful bit of code that repeats a series of instructions for you. For instance, instead of typing echo many times like we did above, we can simply use the code in the editor to the right!

**'For' Loop Syntax**

Cool, right? Let's go through the syntax slowly, step-by-step. Here's an example that just echos the numbers 0 to 9:

<?php

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

echo $i;

}

*// echoes 0123456789*

?>

It breaks down like this:

1. A for loop starts with the forkeyword. This tells PHP to get ready to loop!
2. Next comes a set of parentheses (()). Inside the parentheses, we tell PHP three things, separated by semicolons (;): where to start the loop; where to end the loop; and what to do to get to the next iteration of the loop (for instance, count up by one).
3. After the part in parentheses, the part in curly braces ({}) tells PHP what code to run for each iteration of the loop.

So the above example says: "Start looping with $i at 0, stop the loop *before* $i gets to 10, count up by 1 each time, and for each iteration, echo the current value of $i."

($i++ is shorthand for $i = $i + 1. You'll see this a lot!)

**Writing Your First 'For' Loop**

Great work! Now let's put together our first for loop from start to finish.

A for loop that prints out the numbers 1 through 10 might look something like this:

for ($i = 0; $i < 11; $i++) {

echo $i;

}

This for loop counts up by 1 each time, all the way to 10.

You could change the third part of the forloop so that it counts up by 5 instead, like this:

for ($i = 0; $i < 11; $i = $i + 5) {

echo $i;

}

Instead of $i++, we have $i = i + 5 to count up by 5, all the way to 10.

**When to Use 'For'**

Great job! for loops are great for running the same code over and over, especially when you know ahead of time how many times you'll need to loop. (There are other kinds, such as while and do/while loops, that we can use when we *don't* know ahead of time how many times we'll need to loop, but we'll cover those in a later lesson.)

There's also a special kind of loop called a foreach loop that we can use to update or print out every element in a list—for example, an array. Let's cover foreachnext!

**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.

**Practicing with ForEach**

Let's walk through the foreach syntax step-by-step. First, here's a foreach loop that iterates over an array and prints out each element it finds:

<?php

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

foreach($numbers as $item) {

echo $item;

}

?>

First, we create our array using the array()syntax we learned in the [last lesson](http://www.codecademy.com/courses/web-beginner-en-8a35h).

Next, we use the foreach keyword to start the loop, followed by parentheses. (This is very similar to what we've done with forloops.)

Between the parentheses, we use the $numbers as $item) syntax to tell PHP: "For each thing in $numbers, assign that thing temporarily to the variable $item." (We don't have to use the name $item—just as with for loops, we can call our temporary variable anything we want.)

Finally, we put the code we want to execute between the curly braces. In this case, we just echo each element in turn.

**While Loops in PHP**

**Looping the Loop**

A loop is a structure that tells a computer to execute a set of statements multiple times. If you have a process that you want repeated hundreds of times, it pays to put it in a loop so you don't need to write hundreds of lines of code.

If you are working on these courses in order, you have already seen how a forloop can allow for a set number of loop iterations. But what about a situation where (due to randomness, perhaps) you don't know how many times the loop should repeat? In that case, you can use a while loop.

A while loop will execute as long as a certain condition is true. For example, the loop in the editor will simulate coin flips as long as the number of consecutive heads is less than 3.

**While Loop Syntax**

In the last exercise, you saw how a whileloop can be used to repeat a set of commands an unknown number of times. That loop used the following 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. In the last exercise, cond was the condition that the number of consecutive heads was less than 3: $headCount < 3.

It is important when writing loops to make sure that the loop will exit at some point. The loop

while(2 > 1){

*// Code*

}

will never exit and is an example of an *infinite loop.* **Avoid infinite loops like the plague!** This is why we need to include $loopCond = false; in [line 12](javascript:void(0)). If you submit an infinite loop in one of these exercises, you will need to reload the page to stop it.

**Your First While Loop**

Now it is time for you to write your own while loop from scratch. Maybe you could reproduce the behavior of one of your forloops from the previous course, or you could try to write your own coin flip program. The beauty of programming is that you can do whatever you want!

Unless you want to write an infinite loop on purpose (which you don't!), **do not write infinite loops!** And if you find you have submitted one, refresh the page to stop it.

**Using Endwhile**

PHP offers the following alternative syntax for while loops:

while(cond):

*// looped statements go here*

endwhile;

Note the colon after the end parenthesis and the semicolon after the endwhilestatement.

When they are embedded in HTML, loops that use this endwhile syntax can be more readable than the equivalent syntax involving curly braces.

    while(cond) {  
       // looped statements go here  
    }

Feel free to use whichever syntax you prefer... except on this exercise!

**How Do You Do-While?**

You may have noticed that a while loop checks the loop condition before each iteration of the code inside. A logical alternative is to check the condition *after* each iteration before looping back. A do/while loop does just that. One consequence of this difference is that the code inside a while loop can be bypassed entirely whereas the code inside a do/while loop will execute at least once.

This means that the loop condition can depend exclusively on code within the loop's body. This is the case for the code in the editor where each iteration represents a coin flip, and any time the result of the coin flip is tails, the loop stops.

**Completing the Loop**

In the previous exercise, you saw how a do/while could be used to ensure that the code in a loop executed at least once. For example:

<?php

$i = 0;

do {

echo $i;

} while ($i > 0);

?>

This do / while loop only runs once and then exits:

1. First we set $i equal to 0.
2. Second, the loop runs once and outputs $i, which is 0.
3. Then the condition while ($i > 0); is checked. Since $i is not greater than 0, the condition evaluates to false, andthedo/while` loop stops.

**Functions, Part I**

**Introducing Functions**

Functions are reusable pieces of code that you can use throughout an application, saving you lots of copying and pasting.

PHP has lots of built-in functions, and we'll learn some of them in these exercises. The first set of functions we'll learn about are string manipulation functions.

strlen() is one of the most common String functions in PHP. You pass it a string, or variable containing a string, and it returns the number of characters in that string. An example might be:

<?php

*// get the length of a string and*

*// print it to the screen*

$length = strlen("david");

print $length;

?>

<html>

<p>

<?php

// Get the length of your own name

// and print it to the screen!

$length=strlen("Marlina");

echo $length;

?>

</p>

</html>

**String Functions I**

Another very common string function is 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 string *UPPERCASE* 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"*

**String Functions II**

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*

The parameters passed to strpos() are the haystack and the needle. The function tries to find the needle in the haystack.

It returns either the index of the first character, or false if the needle cannot be found.

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

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

}

*// prints the "Sorry" message*

Use the editor to create your own haystackand needle combinations. You can use variables for both haystack and needle.

**Math Functions I**

Let's move on to a bit of arithmetic. The most common Math function you'll use is round(). This function rounds floating point numbers (numbers with decimal points in them) up or down.

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.

Here's an example:

*// 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*.

**Math Functions II**

A very common and useful function is rand(). This function returns a random number between two numbers. Optionally, you can provide your min and maxnumbers 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 Functions I**

Arrays are a very common thing to use in programming. In fact, array() is actually a function! Good job, you have already used an array function.

Aside from the array() function itself, array\_push() is arguably the most common and useful function for manipulating arrays.

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");

array\_push($fav\_bands, "Nickelback");

array\_push($fav\_bands, "Katy Perry");

array\_push($fav\_bands, "Macklemore");

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

print count($fav\_bands); *// prints 5*

**Array Functions II**

Another common thing to do with arrays is sort them. Handily enough, 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"*

Lastly, we use join(glue, array) so we can easily print out the representations of our sorted arrays in this exercise.

For this exercise, we'll use a comma (,) as the glue.

**Functions, Part II**

**Function Refresher**

Don’t Repeat Yourself! This is a very simple, yet fundamental principle in programming.

Whenever you feel the need to rewrite a block of code, remember that it can probably be written as a function instead. You've seen some of PHP's built-in functions, but you can also define your own!

By using one function instead of several blocks of the same code, you can reduce the amount of clutter in your document and keep your code neat and tidy.

**Function Syntax**

The typical structure of a function is as follows:

function name(parameters) {

statement;

}

1. The keyword function indicates that the code following it will be a user-defined function.
2. name indicates the name of a function, which is case *insensitive*. The name of a function can contain numbers, letters, underscores or dashes.
3. The **arguments**, or **parameters**, will be the optional input a function uses to perform calculations.
4. And of course, the **statements** themselves will be the code the function executes when it is called.