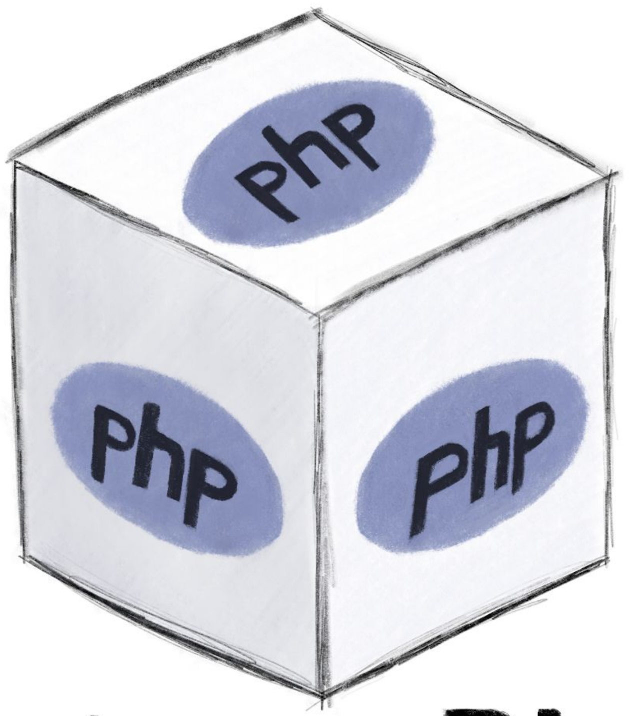
PHP



Beginner Blocks

PHP Beginner Blocks
Version v1.0
Written by Jared York

INTRODUCTION

PHP powers the majority of the web. Nearly 33% of all websites on the internet use Wordpress, according to Hosting Tribunal. This free course covers the basics of the language. While not required, prior programming knowledge is highly beneficial. Let's set up our environment!

SETTING UP

Take a second and open a text editor or code editor. Anything works, even text editors like Notepad or Text Edit. There are more dedicated code editors that are suited for programming, such as Visual Studio Code, or Atom, but like I mentioned, anything will work fine. Before we dive into the code, you will need to download a web server, if you haven't already. On Windows, using a "bundle installer" such as WAMP Server works is easy to install and works well. We recommend using either WAMP Server or XAMPP. Both bundle installers have easy guides to help get the web server up and running. An advantage of WAMP, is that it installs an icon into your system-tray from which you can stop and restart services, as well as change Apache settings such as adding a new folder alias for your project. If you're running Mac OS, you have more options as you're running a more Unix-ey environment. However, if you prefer a bundle installer like WAMP, with a cleaner interface, we strongly recommend you check out MAMP. If you're using a Linux distribution such as Ubuntu, we recommend checking out this guide on DigitalOcean. Once you've installed and set-up a web server, create a new folder in the directory where your web server hosts files from. Open your text editor and create a new file, *index.php*, and save it in the location of this new folder. Inside this file type:

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

Navigate to your browser and type the localhost address your web server uses for serving your file. For example, for WAMP Server, you would type *localhost/<directory name>/index.php* and the page should then appear. If you browser decides to try searching with the term "localhost/<directory name>/index.php", try adding an *http://* in front of localhost. When the page appears, you should see all sorts of information related to your PHP installation. If you see this, then your PHP installation is installed correctly. Now it's time to jump into the code!

HELLO WORLD

In the same file, erase everything and start by typing:

```
<?php
```

This signifies the start of PHP code, and usually starts directly at the top of PHP files. We will be writing code on more than line, which is why we won’t add the ending ?> tag on the same line. In-fact, most developers don’t even write an end tag on most PHP files at all.

Type the following after the opening PHP tag:

```
echo “Hello world!”;
```

If you haven’t already, it’s recommended you have your code editor on one side of your screen, and your web browser on the other side. Refresh your browser, and you should see the text, “Hello world!” on your screen. The echo keyword allows use to print anything before the semicolon to the web page. In PHP it’s important to always add a semicolon ‘;’ to the end of each line.

VARIABLES

While it’s pretty great to be able to print text you type in onto the screen, it will come in handy to do a bit more than that. There’s a reason we’re learning PHP and not learning how to use a printing press right? Erase the hello world line from above, then add the following code:

```
$first_name = ‘John’;  // enter your actual first name
$last_name = ‘Smith’;  // enter your actual last name

echo $first_name . ‘ ’ . $last_name;
```

In most all programming, it’s important to be able to temporarily store bits of data for use further down in the code. If we run this code, we should see the words “John Smith” display. The above code does a number of things. First, we are setting a variable, *first_name* to “John”, and *last_name* to “Smith”. Variables are always declared with $. If you’ve ever used another language where you declare variables with another keyword such as *var*, *Int, String,* etc., the ‘$’ works very similarly. The important thing to know about PHP is it has very loose typecasting. For example, if you were to check if 5 == “5”, it would evaluate true, even though the first operand is a integer and the second is a string. In the above example, when we print out the first and last name, notice that the two variables are separated by a period, two quotes, and another period. We can use a period to combine a string on the left side with the string on the right side of the period. This is also known as string concatenation. It is super useful for printing out results like we did above with the first and last name. Before moving on I better explain what a string is, because if you haven't programmed before, this is new. Strings allow you to specify a string of characters and is encapsulated with single or double quotes. If you're using single quotes to specify a string, you can even add double quotes inside. This will be useful for more complicated concepts.

IF STATEMENTS

In programming, you might want to check if a certain condition is true. Erase everything after the opening PHP tag and add the following:

```
$amount_of_cookies = 12;

if ($amount_of_cookies == 12) {
        echo “You have a dozen cookies!”;
}
else {
        echo “You do not have a dozen cookies.”;
}
```

If you go to run this code, you should see the text, “You have a dozen cookies!” If so, fantastic! You have wrote your first if-statement in PHP. Now, try changing the value of *amount_of_cookies* to 11, then refresh your browser. You should now see “You do not have a dozen cookies.” displayed. If-statements work as follows:

```
If a condition is true, then execute the following code {
        // first condition
}
```

You can then add the word “else” after the closing curly brace and add a code block for when the condition evaluates false:

```
else {
        // code executed when the initial condition is false
}
```

You can even check for a second condition if the initial condition is false. You can add “else if” to accomplish this:

```
else if (<another condition here>) {

}
```

Let’s backup for a second though. Under where we defined the *amount_of_cookies* variable, declare a new variable, *has_enough*, and set it to *true*.

Modify our if statement:

```
if ($amount_of_cookies == 12) {
```

And change it to:

```
if ($amount_of_cookies == 12 && $has_enough == true) {
        echo “You have a dozen cookies and you have enough.”;
}
```

If we change the value of *amount_of_cookies* from 11 back to 12 and refresh the browser, we should now see “You have a dozen cookies and you have enough.” displayed. Change the text in our else statement to “You do not have a dozen cookies or you do not have enough cookies.” Then, set *has_enough* to *false*. The && comparison operator let’s us check if multiple conditions are true, it means AND. You can have any number of conditions and comparison operators you like. You can even nest if-statements inside each other.

Before we move on, it's important to point out the different comparison operators PHP provides us. You can find all of them in the following table:

Operator	**Name**	**Example**	**Output**
==	Equal	$a == $b	If $a is equal to $b, return true.
===	Identical	$a === $b	If $a is equal to $b, and they are both the same type, return true.
!=	Not equal	$a != $b	If $a is not equal to $b, return true.
<>	Not equal	$a <> $b	If $a is not equal to $b, return true.
!==	Not identical	$a !== $b	If $a is not equal to $b, or they are not the same type.
<	Less than	$a < $b	If $a is strictly less than $b.
>	Greater than	$a > $b	If $a is strictly greater than $b.
<=	Less than or greater to	$a <= $b	If $a is less than or equal to $b.
>=	Greater than or equal to	$a >= $b	If $a is greater than or equal to $b.
<=>	Spaceship	$a <=> $b	If values on either side are equal, return 0; If values on the left side is greater, return 1; If values on the right is greater, return -1.

SWITCH STATEMENTS

another type of statement that is very useful in PHP are *switch* statements. Switch statements allow you to check if a value is equal to multiple results. Completely erase everything after our opening PHP tag then add the following:

```
$ingredient_needed = “flour”;

switch ($ingredient_needed) {
        case “flour”: {
                echo “You need more flour!”;
                break;
        }

        case “eggs”: {
                echo “You need more eggs!”;
                break;
        }

        case “butter”: {
                echo “You need more butter!”;
                break;
        }

        case “salt”: {
                echo “You need more salt!”;
                break;
        }

        default: {
                echo “I’ve never heard of that ingredient.”;
                break;
        }
}

echo $ingredient_needed;
```

FUNCTIONS

Functions allow you to reuse code. That's it. Done. …Okay, maybe it's not that simple, but functions are pretty straight forward. Let's erase the code after the opening tag, and start by writing our very own function:

```
function sayHello() {
        echo “Hello world!”;
}
```

By default, this won't do anything even though we've declared the function. We have to *call* the function in order to execute the code inside. You can add function calls anywhere in the code -- almost. We'll be talking scope in a second. Let's first call the function we've declared:

```
sayHello();
```

Refreshing the browser, we should see “Hello world!” displayed. Okay, great. What if we want it to say hello to us? Let's declare two variables above the *sayHello* function:

```
$first_name = “Jared”;  // Enter your actual first name here
$last_name = “York”;  // Enter your actual last name here
```

Of course, let's change the hello world string to the following:

```
echo “Hello ” + $first_name + “ ” + $last_name;
```

If we refresh the page, you'll notice only the word “Hello” shows up without our name. This is because we can't access variables outside the *scope* of the function. In PHP, you cannot access variables outside of the function you're in. *Well, how would you get the value of a variable?* you might ask. It's pretty simple, actually! Say hello to *parameters*. Parameters allow you to pass variables into a function call as *arguments*. It will all make sense shortly. Change your *sayHello* function declaration to:

```
function sayHello($first, $last)
```

Then change what we're echoing to:

```
echo “Hello ”. $first . “ ” . $last;
```

Finally, we have to modify the function call to pass in the *first_name* and *last_name* variables as arguments:

```
sayHello($first_name, $last_name);
```

Refresh the browser, and you should see:

```
Hello <your first name> <your last name>
```

Let’s put both functions and if-statements into practice! Erase the code in your *sayHello* function. Then, add the following:

```
if ($first != null && $last !== null) {
        echo “Hello” . $first . “” . $last;
}
else {
        if ($first == null && $last == null) {
                echo “First and last name is null.”;
        }
        else if ($first == null) {
                echo “The first name is null.”;
        }
        else if ($last == null) {
                echo “The last name is null.”;
        }
}
```

Refresh the page in your web browser, and you should still see Hello with your first and last name printed on the screen. Looking at the code, you might be wondering what *null* means in PHP. Simply put, *null* pretty much means represents missing value. Let’s remove one of the arguments, either *first_name* or *last_name* from the *sayHello* function call. If you refresh the page again, you should see “The first name is null,” if the first name is missing, “The last name is null,” if the last name is missing, or if you removed both arguments, “First and last name is null." should display. Insert the following directly before the line, "if ($first != null && $last !== null) {":

$displayed_name = false;

Then, after this line in the first if-statement block:

echo "Hello " . $first . " " . $last;

Add:

$displayed_name = true;

After the ending curly brace of the else block, add:

return $displayed_name;

When we use the *return* keyword within a function, we can essentially send a value back where we called the function. We can retrieve the value by assigning a variable to the function call.

Let's change the line for calling *sayHello* to:

$has_displayed_name = sayHello($first_name, $last_name);

We can also echo an HTML break tag, *<br>*, as follows to help make our output more readable:

echo '<br>';

That's one of the great things about echo, the ability to add HTML elements!

Now, if we try echoing *has_displayed_name*, we will see either "1" displayed, if the value is *true*, or nothing appears at all, if the value is *false*. But what if we want to elaborate on this response with a quick one liner? We can do that. Add the following line after the line with our function call:

echo $has_displayed_name ? "Your name has been displayed" : "Failed to display your name.";

Above, we are using what is called a *ternary operator*. Ternary operators are pretty fantastic because they allow us to execute small bits of code in a very compact way, depending on whether a condition is *true* or *false*. The first part of this operator, before the "?" mark is for specifying the condition we want to test. The second part, between the question mark and colon, is for what you want the value of *has_displayed_name* to be if the condition is *true*. The last part, after the colon, is what you want the value of *has_displayed_name* to be if the condition tests to be *false*. Using ternary operators can be great in the real world. For example, I use ternary operators frequently for basic form validation checking if a text boxes have been filled in or not.

Another way we can set data is via setting a constant with the define function. We can reference this constant anywhere in the code, regardless of scope. Let's give it a try! At the top of the document, type:

```
define("MY_NAME", "Jared York", false);
```

The first argument is a string which represents the name of the constant. The second argument is the value of the constant. The third argument is optional, and specifies whether the constant name is case-sensitive or not when referenced.

At the bottom of our script, we can type:

```
echo MY_NAME;
```

Notice how we didn't include a "$" before the constant name. This is because constants don't require a "$" before the constant name, which is pretty neat. If we change the third parameter (is case-insensitive), to true, then you could reference constants in various combinations. Here are a few examples:

```
echo My_Name;
echo mY_NaMe;
echo MY_name;
echo my_nAme;
echo My_namE;
```

I think by now you get the point. I'm not sure why anyone would want to make their constant case in-sensitive. It's generally good conventions to capitalize constant names and set it as case-sensitive.

Before we move on, it's important to point out that echo isn't the only way to display text on the screen. Using a *print* statement is another way to display text, however you can only output a single string, and it always returns *1*. Like *echo*, the *print* statement can be used with or without parenthesis. The nice thing about using an *echo* statement is you can provide multiple parameters, enclosed in parenthesis. For example, you could write:

```
echo(“Hello”, “world”, “!”);
```

STRING MANIPULATION

There are many functions PHP provides to be able to manipulate strings. For example, we are able to calculate the length of a string, replacing parts of a string, separate strings by delimiters, find substrings, etc. Let’s take a look at some of these functions!

At some point in your PHP adventures, you will probably encounter a situation where you want to measure the length of a string. We can utilize a handy little function, *strlen*, to accomplish this. Remove all the code in your script except for the opening PHP tag as usual. Then, let’s add the following:

```
$str = “Hello world, this is a test string!”;
echo strlen($str);
```

If we run our *index.php*, we should now see “35” printed on the webpage.

This is great… but what if we want to count the words in a string? We can do that too with *str_word_count*. Let’s clear everything again and add the following:

```
$str = “I need a cup of coffee.”;
echo str_word_count($str);
```

Then we should see “6” displayed on our webpage.

But what if we want to find and replace a string within a string? We can do that too. Clear your file, then define the following string:

```
$str = “The quick brown fox jumps over the lazy dog.”;
```

Perhaps we wanted to change the word “jumps” to “stumbles.” We can accomplish this by adding the following:

```
echo str_replace(“jumps”, “stumbles”, $str);
```

If we run this in our browser, we should see:

```
The quick brown fox stumbles over the lazy dog.
```

This is because the function *str_replace* returns the modified string. We can also add a fourth argument to the *str_replace* function to store the amount of times text was replaced. Let’s give it a try, but keep our existing code. Speaking of replacing, let’s replace:

```
echo str_replace(“jumps”, “stumbles”, $str);
```

With:

```
echo str_replace(“jumps”, “stumbles”, $str, $count);
```

After the echo statement we’ve added previously, let’s add another to display the amount of replacements that occurred:

```
echo “<br>Amount replacements: $count”;
```

We can also reverse the order of the characters in a string wit the *strrev* function. Let’s add a third echo statement to what we’ve already added:

```
echo “<br>Reversed string: ” . strrev($str);
```

If we refresh the webpage now, we should see the reverse of the string, “The quick brown fox jumps over the lazy dog.”:

```
.god yzal eht revo spmuj xof nworb kciuq ehT
```

Fantastic! These are some of the methods of string manipulation that PHP provides us.

ARRAYS

Arrays allow us to store more than one value under one variable name. Instead of typing something like:

```
<?php
$day1 = “Sunday”;
$day2 = “Monday”;
$day3 = “Tuesday”;
$day4 = “Wednesday”;
$day5 = “Thursday”;
$day6 = “Friday”;
$day7 = “Saturday”;
?>
```

We can instead define this in a more efficient way, via a couple different methods. Erase everything in your PHP script except for the opening PHP tag. Then, Add the following:

```
$days_of_week = array(“Sunday”, “Monday”, “Tuesday”, “Wednesday”, “Thursday”, “Friday”,
“Saturday”);
```

This is what’s known as an *indexed* or *numeric* array. This type of array stores each element of the array with a numeric index. Do note that the numeric index counts up from zero instead of one. A slightly more visual representation of this concept would be the following:

```
0  ->  Sunday
1  ->  Monday
2  ->  Thursday
3  ->  Wednesday
4  ->  Thursday
5  ->  Friday
6  ->  Saturday
```

We can also manually assign values to array indexes of a variable. Note, we don’t even need to initialize the array beforehand, we can just assign values:

```
$days_of_week[0] = “Sunday”;
$days_of_week[1] = “Monday”;
$days_of_week[2] = “Tuesday”;
```

```
$days_of_week[3] = “Wednesday”;
$days_of_week[4] = “Thursday”;
$days_of_week[5] = “Friday”:
$days_of_week[6] = “Saturday”;
```

We can retrieve the value of a given index, for example, *4*, to display the corresponding value, *Thursday*:

```
echo $days_of_week[4];
```

Obviously, we don’t have to echo the resulting value, you can use it anywhere but this is an example use case.

This is great and all, but what if we want to assign values to keys in an array, like key/value pairs? Not a problem, PHP supports a type of array called an *associative array*. Associative arrays allow you to specify keys using strings, then assign them to a value of any type. What if we wanted a way of keeping track of items sold during a week? We can utilize associative arrays as the data structure for this scenario. Remove the code we wrote previously and type:

```
$items_sold[“Sunday”] = 0;
$items_sold[“Monday”] = 26;
$items_sold[“Tuesday”] = 29;
$items_sold[“Wednesday”] = 34;
$items_sold[“Thursday”] = 23;
$items_sold[“Friday”] = 31;
$items_sold[“Saturday”] = 15;
```

Associative arrays can also be defined in a more compact way like so:

```
$items_sold = array(“Sunday”=>0, “Monday”=>26, “Tuesday”=>29, “Wednesday”=>34,
“Thursday”=>23, “Friday”=>31, “Saturday”=>15);
```

If we wanted to retrieve the value for any particular key of this array, we can use:

```
echo $items_sold[key];  // this could be Monday, Wednesday, Thursday, etc.
```

Since arrays can hold essentially anything, we can even add arrays inside arrays. This type of array is known as a *multidimensional array*. Multidimensional arrays can contain arrays as elements, and those arrays can even contain arrays, and so on. Perhaps we wanted to store basic weather data for a given week. We can use a multidimensional array like so:

```
$weather_data = array(
  array(
    "day" => "Sunday",
    "condition" => "sunny",
    "high" => 74,
    "low" => 61,
    "wind" => "6 mph"
  ),
  array(
    "day" => "Monday",
    "condition" => "partly cloudy",
    "high" => 78,
    "low" => 63,
    "wind" => "8 mph"
  ),
  array(
    "day" => "Tuesday",
    "condition" => "rain",
    "high" => 73,
    "low" => 62,
    "wind" => "11 mph"
  ),
  array(
    "day" => "Wednesday",
    "condition" => "storms",
    "high" => 78,
    "low" => 67,
    "wind" => "18 mph"
  ),
  array(
    "day" => "Thursday",
    "condition" => "partly sunny",
    "high" => 73,
    "low" => 64,
    "wind" => "13 mph"
  ),
  array(
    "day" => "Friday",
    "condition" => "partly cloudy",
    "high" => 68,
    "low" => 61,
    "wind" => "11 mph"
  ),
  array(
    "day" => "Saturday",
    "condition" => "cloudy",
    "high" => 74,
    "low" => 62,
    "wind" => "7 mph"
  )
);
```

If you wanted to say, retrieve the wind value on Wednesday, you would write:

```
$weather_data[3][“wind”];
```

When we refresh the browser, we should now see:

```
18 mph
```

There will probably a point where you need to view the structure and values of an array. There are a couple ways to perform this. A couple of the most common methods include the *var_dump* and *print_r* functions. Let’s first try utilizing the *var_dump* function. Clear your file, then add the following to your script:

```
$days = array(“Sunday”, “Monday”, “Tuesday”, “Wednesday”, “Thursday”, “Friday”,
“Saturday”);

var_dump($days);
```

If we run our script, we should then see the following output:

```
array(7) { [0]=> string(6) "Sunday" [1]=> string(6) "Monday" [2]=> string(7) "Tuesday" [3]=>
string(9) "Wednesday" [4]=> string(8) "Thursday" [5]=> string(6) "Friday" [6]=> string(8)
"Saturday" }
```

By using *var_dump*, we can see type data as well as the key values displayed. Now, let's try replacing *var_dump* with *print_r*. If we refresh the browser now, we should see this output:

```
Array ( [0] => Sunday [1] => Monday [2] => Tuesday [3] => Wednesday [4] => Thursday [5]
=> Friday [6] => Saturday )
```

Notice there's no type data. That's because it omits the type data. The purpose of the *print_r* function is to print human-readable information about a variable.

Sometimes, you might need to sort an array of strings or numbers. PHP provides a few basic functions to sort these types. Here is a table displaying the built-in sorting functions PHP gives us:

sort()	rsort()	Used for sorting indexed arrays
asort()	arsort()	Used for sorting associative arrays by value
ksort()	krsort()	Used for sorting associative arrays by key

Let's start by using the *sort* function. This function allows you to sort indexed arrays in ascending order alphabetically. Just before the *print_r* call, add the following line:

```
sort($numbers);
```

If you refresh the page, you should see:

```
Array ( [0] => Friday [1] => Monday [2] => Saturday [3] => Sunday [4] => Thursday [5] =>
Tuesday [6] => Wednesday )
```

Now let's change our array to the following:

```
$days = array(1, 5, 8, 2, 9, 0, 4, 7, 3);
```

We should see a similar result as the string version, but with the order of numbers specified:

```
Array ( [0] => 0 [1] => 1 [2] => 2 [3] => 3 [4] => 4 [5] => 5 [6] => 7 [7] => 8 [8] => 9 )
```

We can use a similar function to sort this array in descending order alphabetically. Let's add our old array back, and clear out the number array we had before. The array that should be in your script should look like the following:

```
$days = array("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday");
```

Change *sort* to *rsort*, then refresh the webpage.

```
Array ( [0] => Wednesday [1] => Tuesday [2] => Thursday [3] => Sunday [4] => Saturday [5]
=> Monday [6] => Friday )
```

Similarly, we can use *rsort* to sort the numbers in descending order. First, let's change the array we have currently to the numbers array we had before:

```
$days = array(1, 5, 8, 2, 9, 0, 4, 7, 3);
```

If we refresh the page now, we should see:

```
Array ( [0] => 1 [1] => 5 [2] => 8 [3] => 2 [4] => 9 [5] => 0 [6] => 4 [7] => 7 [8] => 3 )
```

We can also sort associative arrays in ascending order by value using the *asort*. Clear your file except for the opening PHP tag, then add the following code:

```
$goods_sold = array("cakes"=>12, "cookies"=> 48, "cupcakes"=>24, "bread loafs"=>15,
"muffins"=>23);
```

Next we can use the *asort* function and display the result with the *print_r* function again:

```
asort($goods_sold);
```

```
print_r($goods_sold);
```

If we refresh the browser we should now see our different key/value pairs in order from least to greatest by the value.

```
Array ( [cakes] => 12 [bread loafs] => 15 [muffins] => 23 [cupcakes] => 24 [cookies] => 48 )
```

We can also sort associative arrays in descending order by value. Let's change the *asort* function call to *arsort*. If we take a look at our page now, we should see:

```
Array ( [cookies] => 48 [cupcakes] => 24 [muffins] => 23 [bread loafs] => 15 [cakes] => 12 )
```

Not only can we sort associative arrays by value, but we can also sort them by key. We should replace *arsort* to *ksort*. If we refresh the page now, we should see the array elements listed in ascending order, alphabetically by key.

```
Array ( [bread loafs] => 15 [cakes] => 12 [cookies] => 48 [cupcakes] => 24 [muffins] => 23 )
```

We can also sort associative arrays in descending order by key. To do this, we can utilize the *krsort* function. In our case, items will be printed reverse alphabetically, since it's sorted in descending order:

```
Array ( [muffins] => 23 [cupcakes] => 24 [cookies] => 48 [cakes] => 12 [bread loafs] => 15 )
```

LOOPS

Loops allow us to repeat code until a condition is met. They can be useful for automating tasks that repeat, and keeps the codebase more organized. PHP provides us four different types of loops:

- while - Repeats a block of code until a condition evaluates to true.
- do… while - the block of code executes once then the condition is evaluated. If the condition evaluates to true, the statement will repeat as long as the condition specified is true.
- for - Repeats a block of code until the counter reaches a specified number.
- foreach - Repeats a block of code for every element in an array.

We should start with the while loop. As stated above, a while loops allows you to repeat a block of code until a condition is true. First, let's clear out our script, then add the following code:

```
$i = 0;
while ($i < 5) {
        $i++;
        echo "i = " . $i . "<br>";
}
```

First, we are setting the variable *$i* to 0. Then the code in the while loop will repeat until $i is equal to five. Before we echo the value of *$i*, we also increment (add to) *$i*.

The next type of loop PHP gives us are do...while loops. The do-while loop is a type of while loop. This loop will execute a block of code once, then the condition specified is evaluated. If the condition is true, the statement will repeat as long as the condition specified is true. Let's give this a try. Let's erase the while loop we just made, then add the following:

```
$i = 1;
do {
        $i++;
        echo "The number is " . $i . "<br>";
}
while ($i < 5);
```

The main difference between while and do-while loops is important. With a while loop, the specified condition to be evaluated is tested at the beginning of each iteration. If the conditional expression evaluates to false, the loop will not be executed.

A do-while loop will always execute once, even if the conditional expression is false, since the condition is evaluated at the end of the iteration rather than the beginning.

If we take a look at our page in the browser now, we should see:

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

The for loop will repeat a block of code until the specified condition is met. This loop will usually be used to execute a code block a certain number of times. The general syntax used with for loops looks like the following:

```
for (initialization; condition; increment) {
   // code to execute
}
```

The parameters used in for loops have the following meanings:

- *initialization* - used for initializing counter variables, evaluates once unconditionally before the first execution of the code block.
- *condition* - in the beginning of an iteration, the condition is evaluated. If the condition evaluates to *true*, the loop continues. If it evaluates to *false*, the execution of the loop stops.
- *increment* - it updates the loop counter to a new values. It is evaluated at the end of every iteration.

As usual, let’s remove the code in our script file, then add the following code:

```
for ($i = 0; $i < 5; $i++) {
        echo “The number is ” . $i . “<br>”;
}
```

The first thing this code does is set the counter, *$i*, to *0*. If *$i* is less than five, repeat the code, add one, then stop once *$i* reaches *5*. Now if we refresh our page, we should see:

```
The number is 0
The number is 1
The number is 2
The number is 3
The number is 4
```

The last type of loop that PHP provides us is the foreach loop. The foreach loop allows us to iterate over arrays. The syntax for a foreach loop appears like the following:

```
foreach ($array as $value) {
        // code to execute
}
```

Let’s clear everything out, then add an array:

```
$ingredients = array(“flour”, “salt”, “sugar”, “eggs”, “butter”, “oil”);
```

Then let’s add our foreach loop:

```
foreach ($ingredients as $value) {
        echo $value . “<br>”;
}
```

When we refresh the page now, we should see the following displayed:

```
flour
salt
sugar
eggs
butter
Oil
```

We can also use a foreach loop to iterate through an associative array as well. Let’s clear out our PHP file, then add the following:

```
$weather_today = array(
        “condition”=>”sunny”,
        “high”=>76,
        “low”=>63,
        “wind”=>”12 mph”
);

foreach ($weather_today as $key => $value) {
        echo $key . “: “ . $value . “<br>”;
}
```

If we refresh the page now, we should now see:

condition: sunny
high: 76
low: 63
wind: 12 mph

OPERATORS

PHP provides us a number of arithmetic operators that we can use. These operators are used to perform common operations such as addition, subtraction, multiplication, division, etc. Here is a list of the PHP arithmetic operators we have:

Operator	Description	Example	Result
+	Addition	$a + $b	Sum of $a and $b
-	Subtraction	$a - $b	Difference of $a and $b
*	Multiplication	$a * $b	Product of $a and $b
/	Division	$a / $b	Quotient of $a and $b
%	Modulus	$a % $b	Remainder of $a divided by $b

Let's take a look at these operators in the wild. Here is an example:

```
<?php
$a = 16;
$b = 32;
echo  $a + $b;  // the output is 48
echo "<br>";
echo $a - $b;  // the output is -16
echo "<br>";
echo $a * $b;  // the output is 512
echo "<br>";
echo $a / $b;  // the output is 0.5
echo "<br>";
echo $a % $b;  // the output is 16
```

```
?>
```

Adding to this, PHP also provides ways to use operators when assigning values to variables. These are called *assignment operators*. Take the following example:

```
$a = 10;
$a = $a + 1;  // $a is now equal to 11
```

Okay… this is great… but we can do better. Instead of using “$a = $a + 1;”, we can instead utilize:

```
$a += 1;
```

This is a bit more compact! Assignment operators are a pretty neat way to perform an operation while assigning a value at the same time. Here is a table of the available assignment operators PHP provides us:

Operator	Description	Example	The Equivalent To
=	Assign	$a = $b	$a = $b
+=	Add and assign	$a += $b	$a = $a + $b
-=	Subtract and assign	$a -= $b	$a = $a - $b
*=	Multiply and assign	$a *= $b	$a = $a * $b
/=	Divide and assign	$a /= $b	$a = $a / $b
%=	Divide and assign modulus	$a %= $b	$a = $a % $b

If we take the addition assignment operator, for example, we can construct something like the following:

```
$x = 5;
$x += 2;
```

```
echo $x;  // The result is 7
```

This works similarly for the assignment operators listed above.

MATH OPERATIONS

Often in PHP, you will encounter situations where you will need to make calculations. PHP provides us a handful of functions to help you perform anything from simple calculations to advanced calculations. We can utilize the arithmetic operators we've just learned about to make these calculations. Multiple arithmetic operators can also be used in the same operation. It is important to point out that every math operation has a certain level of precedence. Usually multiplication and division operations are performed before addition and subtraction. Parentheses, however can modify this precedence. Expressions enclosed in parentheses are always evaluated first, no matter what the operation's precedence level is. Let's take a look at the following example:

```
<?php
echo 2 + 6 * 30;           // results in 182
echo (2 + 6) * 30;         // results in 240
echo 2 + 6 * 30 / 2        // results in 92
echo 5 * 12 / 4 - 2        // results in 13
echo 5 * 12 / (4 - 2);     // results in 30
echo 5 + 12 / 4 - 2;       // results in 6
echo (5 + 12) / (4 - 2);  // results in 8.5
?>
```

There may also be a point where you want to take the absolute value of a number. PHP provides a little function to accomplish this: *abs*. Here is an example:

```
<?php
echo abs(3);       // results in 3 (integer)
echo abs(-3);      // results in 3 (integer)
echo abs(6.4);     // results in 6.4 (double or float)
echo abs(-6.4)     // results in -6.4 (double or float)
?>
```

As seen, if the number given is negative, the value returned by *abs* is positive. However, if the number is positive, the function just returns the number given.

There will also probably be a circumstance where you need to round in a variety of ways. PHP provides us three functions to do this: *ceil*, *floor*, and *round*. The *ceil* function always rounds a number up, no matter what. The *floor* function always rounds a number down, regardless. Finally, the *round* function rounds up if the value is greater or equal to .5, and rounds down if less than .5 of the number given. Here is an example:

```
<?php
echo “ceil:” . ceil(3.2) . “<br>”;      // results in 4
echo “floor:” . floor(3.7) . “<br>”;  // results in 3
echo “round:” . round(3.49);          // results in 3
?>
```

GET AND POST

Web browsers usually communicate with the server via one of the two HTTP (Hypertext Transfer Protocol) methods: GET and POST. Both ways pass the information differently and have different pros and cons, as covered below.

Let’s start with the GET method. You may notice if you look at the address bar while on some websites, you might find the characters “?” and “&.” These two characters signal that the URL contains URL parameters. Let’s take a look at the following example, of which I search for the term “php” with DuckDuckGo. Here is the URL:

https://duckduckgo.com/?q=php&t=hn&ia=web

In the above URL, there is a parameter named *q*. The value “php” is assigned to this parameter. To add a single URL parameter, the character “?” needs to be appended to the end of a URL. Further parameters need a “&” between each parameter. It’s recommended that only simple text data is sent via the GET method.

There are advantages and disadvantages of using the GET method. One of the nice things about using the GET method, is that it’s possible to bookmark the URL. This is useful as a simple way store data for a site (but not flexible). One of the disadvantages to using the GET method, is that you should not pass sensitive information such as login information, or payment details using the

GET method. This is because URL parameters are fully visible in a URL query string. These URLs are also potentially stored in the memory of the client's browser. Since the GET method stores data in a server environment variable, there is a limitation to the length the URL can be. Therefor, there is a limit to the total data that can be sent.

Let's get back to writing some code as we wrap up. Completely remove everything in your PHP file, including the opening PHP tag. Next, let's write the following HTML markup, within our PHP script:

```
<!DOCTYPE html>
<html>
        <head>
                <title>PHP GET Method Example</title>
        </head>

        <body>
                <?php
                if (isset($_GET[“my_name”])) {
                        echo “<p>Hello ” . $_GET[“my_name”] . “!</p>”;
                }
                ?>

                <form method=“get” action=“<?php echo $_SERVER[“PHP_SELF”]; ?>”>
                        <label for=“input_name”>My Name:</label>
                        <input type=“text” name=“my_name” id=“tb_my_name”>
                        <button type=“submit”>Submit</button>
                </form>
        </body>
</html>
```

Let's try refreshing our page in the browser. If we fill out our name in the textbox and click “Submit,” we should see, “Hello, <your name>!”

PHP also provides us with another superglobal variable, *$_REQUEST*. This superglobal variables holds the values of both the *$_GET* and *$_POST* variables, and the values of the *$_COOKIE* superglobal variable.

Let's change this bit:

```
if (isset($_GET[“my_name”])) {
        echo “<p>Hello,” . $_GET[“my_name”] . “!</p>”;
}
```

To:

```
if (isset($_REQUEST[“my_name”])) {
        echo “<p>Hello,” . $_REQUEST[“my_name”] . “!</p>”;
}
```

You will find after refreshing the page that functionally, the line behaves the same as before.

CONCLUDING THOUGHTS

While aged, PHP is an extremely mature language. Acting a binder that holds the majority of websites and databases together, PHP will most likely power the back-end of the Web for the years to come. It is our hope that this course has been useful for you! I will most likely be writing more advanced tutorials and courses covering form handling.

If you have any questions, comments, or other feedback, I’d love to know. You can reach me via email at jared.york@yorkcs.com. You can also tweet me @jaredyork_. What would you like me to write about? I’d like to know that too.

If you found this course valuable, and would like to hear about future courses and tutorials we write, please fill out the form.

While there isn’t source code available for this course, the source code for other courses of mine can be found on GitHub.