PHP PANDAS

THE PHP PROGRAMMING LANGUAGE FOR EVERYONE

DAYLE REES

PHP Pandas (PHP7!)

The PHP Programming Language for Everyone.

Dayle Rees

This book is for sale at http://leanpub.com/php-pandas

This version was published on 2016-04-29

Tweet This Book!

Please help Dayle Rees by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I'm learning about PHP and Pandas AT THE SAME TIME. You can too! @ http://leanpub.com/php-pandas #PHPPandas @dayleree s

The suggested hashtag for this book is #PHPPandas.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#PHPPandas

Also By Dayle Rees

Laravel: Code Happy

Laravel: Code Happy (ES)

Laravel: Code Happy (JP)

Laravel: Code Bright

Code Happy (ITA)

Laravel: Code Bright (ES)

Laravel: Code Bright (SR)

Laravel: Code Bright (JP)

Laravel: Code Bright (IT)

Laravel: Code Bright (TR) Türkçe

Laravel: Code Bright (PT-BR)

Laravel: Code Bright (RU)

Laravel: Code Smart

PHP Pandas (ES)

PHP Pandas (IT)

PHP Pandas (FR)

PHP Pandas (TR)

Contents

Acknowledgements

First of all, I would like to thank my girlfriend Emma, for not only putting up with all my nerdy antics but also for taking the amazing red panda shots for both books! Love you, Emma!

Thanks to my parents, who have been supporting my interest in these math boxes for thirty years! Also thanks for buying a billion copies or so of the first book for family members!

Thank you to everyone who bought my other books Code Happy and Code Bright, and all of the Laravel community. Without your support, I'd never have had the confidence to continue writing.

Errata

This may be my third book, and my writing will have improved since the last one, but I assure you that there will be many, many errors.

You can help support the title by sending an email with any errors you have found to me@daylerees.com[1] along with the section title.

Errors will be fixed as they are discovered. Fixes will be released within future updates to the book.

[1] mailto:me@daylerees.com

Feedback

Likewise, you can send any feedback you may have about the content of the book or otherwise. You can send an email to me@daylerees.com[2] or tweet to @daylerees.

I will endeavour to reply to all mail that I receive.

[2] mailto:me@daylerees.com

Translations

If you would like to translate PHP Pandas into your language, then please send an email to me@daylerees.com[3] with your intentions. I will offer a 50/50 split of the profits from the translated copy, which will be the same price as the English copy.

Please note that the book is written in markdown format.

[3] mailto:me@daylerees.com

1. Introduction

Well hello there! Aren't you just the most handsome AND/OR beautiful reader on the planet! Well done you for buying PHP Pandas, and for taking the first step towards your career as a world-famous web developer.

Who am I? Well, that's a simple question! My name's Dayle, and I'll be your author for this adventure. I've been writing books for beginners for a few years now and have taken many other charming readers like yourself on adventures to learning new skills. We'll make new discoveries together, and all along the journey, rest assured that I'll be right by your side.

> Why do you write like a crazy person?

Excuse me? Oh, this. Well you see, this is the only way that I know how to write. If you're looking for a technical book full of science teacher stern-ness (Is that a word? I hope that's a word.) then I'm afraid you've come to the wrong place. I write my books for people. I like to think that we're buddies, sitting in the pub, talking about PHP over a pint of Special Bre… Fosters.

The truth is that the beginners that I've written for tend to like my writing style. They're not looking to gain a maths degree from this book. Instead, they're looking to learn a thing or two about PHP, and that, I can promise you!

Oh hey, you'll also notice that we're talking right now. You don't get that from other authors do you? You see, I have this magic power that will make you talk to me and ask your questions.

> Wait, how did you do tha…

That would be a trade secret. Sorry, we can't share that just yet, but don't you feel glad that you get to be a part of this adventure, and not just an observer?

> I guess so… Sure I'll give it a go.

Excellent.

Well now's about the time where any other book would be telling you about PHP, it's whole history, its application, its author and about a million other things. Well, we've already established that I'm not the most traditional author, and I'm not fond

of such chapters. You've bought this book to learn about PHP, so you've already built up a little curiosity about the language. I think this is all you're going to need.

PHP is a programming language that powers most of the sites out there on the big, wide ol' interwebs. It was originally written by a guy called Rasmus Lerdorf, who can often be seen smiling in pretty much any image you find of him on Google. Now, Rasmus is a great guy, and in my own way I thank him each and every day for this language that has given me a trade, but I think that's all you need to know about him. Other PHP books would probably be telling you his favorite cereal about now, but instead, how about we jump in and start learning?

This book is for ***absolute*** beginners. This means that if you've never tried programming before in your life, then you're in luck, my friend! If you've already tried programming, then you'll do just fine. If you're a PHP expert, then now's a time for a refresh of your skills, and maybe you'll pick up a few tips and tricks along the way.

I've been using my girlfriend (no dev experience), my non-technical colleagues, and random people on the street, forcing my book upon them as guinea pigs to see how it goes down with folks that have no prior knowledge of PHP. My little guinea pigs did exceedingly well, so now it's your turn, squeak squeak!

My goal for this book is for it to become the most fun, factual, and fantastic PHP book that's on the market. I want it to be **the** book that gets recommended when someone is about to become a PHP developer. I've worked hard to make it accessible to everyone, so if you enjoy this adventure, then please tweet about it, blog about it, buy copies for your friends and family, or just print it out and slap people in the face with it as you pass them on the street.

This book is a syntax book for PHP. It's not going to teach you how to make websites (I'm working on the title in the series for this). Instead, it's the first step that will build your foundation knowledge of the language so that when you come to build your first website, you're gonna be @%Â£ˆ hot, baby!

If you read the book, and you feel like something is missing, that a certain chapter is confusing, or there's anything else bothering you, then please send me an email to `me@daylerees.com` to let me know! I'm incredibly responsive (thanks to all my media queries… haha… programmer joke), and I want this book to be perfect for everyone.

If you read the book, and you didn't find anything wrong, well… send me an email to tell me you enjoyed it! I'd love to hear from you.

Right then, let's not waste any more time. You've got some skills to learn! Flip the page, imagine the Jurassic Park theme when they open the gates and prepare to enter the world of development!

2. Installation

Before we begin working with PHP, we must first install it. You see, PHP is an application like any other. It needs to be installed on our system before it can process PHP code.

The method of installation varies greatly depending on the operating system that we are using. For that reason, I've provided three different guides for installing PHP. The first section will explain how to install PHP on a Linux distribution, namely Ubuntu due to its popularity. The second section will explain how to install PHP on an Apple Mac OSX system. Finally, the third section will explain how to install PHP on the Windows operating system.

We'll only be installing the console version of PHP. We won't be setting up a web server just yet. We'll get to that in a later title. The console version of PHP is all we need to get started with our learning process.

Remember, you only need to read the appropriate section for your computer. Once you have PHP installed, go ahead and skip to the next chapter of the book.

Linux

The best way to install PHP on a unix-based Linux distribution is to use a package manager. The package manager available depends greatly upon the distribution of Linux that you have chosen. I've decided to provide instructions for installing PHP on Ubuntu, one of the most popular distributions of Linux.

Ubuntu uses the `apt` package manager to install its packages. To install the console version of PHP we need to install the `php5-cli` package. Let's do this now. First, open a new terminal. You'll need to type the following instruction.

```
1 $ sudo apt-get install php5-cli
```

You don't need to type the dollar sign; that's just the terminal prompt to show you that we're typing it into the console. Once you hit enter, apt will retrieve the PHP application package, and install it for you.

That's it! You're done. Well, you should be. Let's check, shall we? Simply type:

```
$ php -v
```

This command is used to show the current version of PHP installed. You should see something similar to the following.

```
PHP 5.5.13 (cli) (built: Jun  5 2014 19:13:23)
Copyright (c) 1997-2014 The PHP Group
Zend Engine v2.5.0, Copyright (c) 1998-2014 Zend Technologies
```

Yours won't be exactly the same, after all, we're all different, aren't we? In the example above, the PHP version is `5.5.13`. Hopefully, your PHP version number should be `5.4.0` or greater.

If your version isn't right then, you'll have to consult the documentation for your Linux distribution to find out how to install the appropriate version.

Go ahead and skip to the next chapter, you're done!

Mac OSX

On the Macintosh operating system, PHP comes pre-installed. Go ahead, open up the Terminal application and type the following to find the version of PHP you're using.

```
$ php -v
```

Don't type the dollar sign; that's the terminal prompt! You should see something similar to the following, but not exactly the same.

```
PHP 5.4.24 (cli) (built: Jan 19 2014 21:32:15)
Copyright (c) 1997-2013 The PHP Group
Zend Engine v2.4.0, Copyright (c) 1998-2013 Zend Technologies
```

The PHP version in the example above is `5.4.24`. As long as your version of PHP is greater than `5.4` then you're fine, and can move to the next chapter.

If yours isn't, that's okay. We can use a third party package manager for OSX to install a newer version of PHP.

We're going to use a package manager called 'Homebrew' or just 'Brew' for short. To install Homebrew, follow the instructions found on the following site:

brew.sh[1]

I don't want to copy the instructions here, as they often change between different releases. Once you have Homebrew installed, it's time to install a newer version of PHP. I recommend installing version `5.5`. You can do this using the following command.

```
$ brew install php55
```

Next, you need to add the location for this version of PHP to your system PATH variable. Don't worry, just type the following.

```
$ PATH=~/usr/local/Cellar/php55/5.5.13/bin:$PATH
```

You may need to update the version number to match the version of PHP that Homebrew has installed on your system. Now let's have another go at checking the version of PHP.

```
$ php -v
```

Hopefully, this time, you'll have a version greater than PHP `5.4`. Go ahead and skip to the next chapter.

Windows

Installing PHP on Windows is a little more difficult, at least for me it is. I've tested the instructions below on my Windows 10 machine, but if you have any difficulty replicating these steps, let me know, and I'll find someone who's more Windows-savvy to rewrite this section.

First, head over to:

http://windows.php.net/download

Here you'll want to download the latest PHP 5.4 and above zip archive. Once the archive has been downloaded, you'll want to extract it to a sensible location. I chose to extract mine here:

```
C:\Users\Dayle\PHP
```

You're going to need a command prompt to execute the scripts that we write in this book. So here's a great way of running a command prompt on Windows.

Right click on your desktop, or any folder and choose 'Create Shortcut'. In the target box enter:

[1] http://brew.sh/

```
cmd.exe
```

Click next, and name your shortcut “PHP”.

Finally, you’ll want to right click your shortcut and click ‘Properties’. On the ‘Shortcut’ tab, change the ‘Start In’ field to match the location where you extracted the PHP archive. Click ‘OK; when done.

Double click on your PHP shortcut and you should be greeted with a command prompt. Type…

```
php -v
```

..and you should be greeted with the PHP version information. Confirm that the version is greater than or equal to PHP 5.4, and then move to the next chapter.

Once again, sorry for the roughness of this subchapter. I’ve not used Windows as a development machine for some years now. If anyone has a better way of running PHP on Windows, kindly email your instructions to receive your 5 minutes of fame within this chapter!

3. Finding Answers

I know. That's a kinda fluffy title isn't it? You're going to have to trust me when I say that this is important stuff. This chapter is about your confidence as an up and coming developer. Learning is hard, but don't worry; I'm going to help you through this.

Developers are robots.

Why did you decide to pursue development? No, wait! Let me guess. You saw a rockstar PHP developer swagger out of a Limousine into one of New York's hottest night spots, order five bottles of Cristal and spend the evening chilling with Jay-Z and the ghost of Tupac.

It's true; a developers life is a glamorous one. I have to write these chapters within my 5 hours of sobriety a day. You've probably seen a developer writing some code and thought…

> Oh man, that dev must be a robot. They know all of those code words and functions and how they all work.

When people without development experience approach developers, they assume they are genius types with mathematics honour degrees. Perhaps this is true for some developers, but it's certainly not true for me. I'd like to think that other developers would agree.

The truth is, we're not perfect. We're not even close to perfect. If you think that developers know all of these PHP functions and snippets from memory, then you are fooling yourself into thinking that you will never have the capacity to keep up.

It's just not true. We don't memorise everything. In fact, a majority of the code that we use day to day is from reference. We are Google warriors. There are functions in PHP that do the most simple things to lines of text, and I look at the PHP documentation almost every week to find the order of the parameters that I pass to them.

When I'm completely stuck, I'll try using Google to see if another developer has found a similar challenge. Often I'll find a suitable solution that another developer has discovered, or enough information to point me to a solution. Of course, this works both ways, I'll try and give my solutions back to the community. I'll post

answers on Stack Overflow, and I'll contribute to forums or discussions. Giving back to the community is important.

So you see, we aren't robots. We don't know everything about the language, and we don't have a solution to every problem. However, we are fantastic researchers. We are opportunists. We are resourceful problem solvers. We are developers.

The art of Googling

When people tell you to Google something, it's easy to take it as an insult. Or perhaps sarcasm? It's not. Google is our homepage for good reason. Let's learn how we can find answers to common development issues.

We're writing a program, and somewhere we need to reverse a sentence so that 'Pandas rule!' becomes '!elur sadnaP'. We have no idea how to approach this task. We're just getting started with PHP.

We know that in PHP a sequence of text is called a 'string.' We know this because we didn't give up on this crazy book with the Panda examples, and we discovered this fact in a later chapter. Right?

So we know what we want to do. We would like to reverse a string. Let's construct a search query for Google.

```
1 reverse string
```

Nope, wait! The problem here is that there are thousands of programming languages. Seriously, computers have been around for a while!

If we search for 'reverse string' then we're going to get answers for C++, ASP.NET, Erlang, you name it. Our focus is on PHP. We don't care about these other languages. We'll have time to play with them later when we become PHP masterminds. Let's fix this problem by adding the language to the search query.

```
1 php reverse string
```

Perfect. Let's take a look at the results that we get back from our Google search. This might be a good time to mention that I don't work for Google, and I'm not working for commission. Feel free to use Bing if you prefer it, but you might end up buying a used horse trailer rather than finding a string reverse function. So where are those results?

Reverse a string - PHP

http://www.php.net/manual/en/function.strrev.php[1]

Reverse a string with php - Stack Overflow

http://stackoverflow.com/questions/11100634/reverse-a-string-with-php[2]

By asking the right question, we receive some useful resources in return. The PHP Manual (sometimes known as the PHP API docs) and Stack Overflow are two of the most useful problem-solving resources for PHP available on the internet. I'm not saying they always have the right answer. There are other great sites too, but I'm sure you'll see a pattern in how often your searches result in browsing pages on these two sites.

Right now we're looking for some tool to reverse a string. We're not looking to solve an abstract problem; we know exactly what we want.

Go ahead and click that first link, we'll be greeted with the lovely PHP manual page for a function called `strrev()`. You don't need to know what a function is yet. Don't worry if this is over your head.

Once you're up to speed with functions you'll see that this PHP manual page offers all that we need to know about using the `strrev()` function, and examples of how to use it.

You see by asking the right questions we received all the help we needed to continue with our work. We had no prior knowledge of the `strrev()` function, but instead we knew the problem that we had to solve. That was enough to lead us to the solution. It doesn't matter if we have to go back to this page later.

Perhaps we don't use the function frequently enough to need to remember its usage pattern. Although, you'll find that if you begin to use the function more and more, and that you frequent the manual page, then before to long you won't need guidance for that problem. You'll instantly think 'Hey I should use that `strrev()` function that I use all the time, and I know exactly how it works!'. It will become muscle memory, and will be part of your toolset.

So the lesson that I'm hoping you have learned from this chapter is that you shouldn't panic. You don't need to remember everything, and it's perfectly natural to ask for help. In fact, it's human to ask for help, and it's human to learn from your experiences.

Congratulations! You're a human, not a robot.

[1] www.php.net/manual/en/function.strrev.php
[2] http://stackoverflow.com/questions/11100634/reverse-a-string-with-php

4. Files

Here's a shocker for you. PHP code is kept in files. I'm sorry, but it's true! You are going to be working with lots and lots of files. Well actually, sometimes one file, but later you'll be working with many, many files!

Now that we have that shocking truth out of the way, isn't it time that you learned how to create a PHP file.

> Dayle, I understand the fundamentals of a computer file system.

Well done buddy! Good for you, but that's not where we're going with this. You see, most PHP files have something in common. I'm talking about the PHP script tag.

Take a close look at this little fellow.

Example 01: PHP tag.

```
1 <?php
```

Beautiful isn't she? What a glossy coat. A fantastic specimen.

> I.. erm..

What? Don't you feel the same about her? Trust me, after many years of development in PHP you will find her quite beautiful. You'll see her when you close your eyes to go to sleep at night. She's your best friend. She lets you use PHP.

I always prefer to lead with a practical example, so let's try something together. Create a new file, and call it `test.php`. PHP files usually have the `.php` extension. In honesty, we can execute them without it, but you should stick to it because if you don't, then the bigger developers will laugh at you, steal your lunch and make you cry. Just kidding... developers are a friendly bunch, but you really should use the `.php` extension.

First of all, let's write the words...

Example 02: Some text.

```
Pandas rule!
```

...into the file, and save it.

Great, now let's execute the file. We can use this by calling the `php` application at the command line or unix shell, and passing the file name as a parameter. For example, on my Mac, I'll be typing the following.

Example 03: Executing a PHP file.

```
php test.php
```

You'll see the words `Pandas rule!` outputted to the screen. This is because everything outside of our beautiful PHP tags is outputted when the application is executed. Let's try something else. We are going to use our first PHP tag.

Let's edit the file so that it reads as follows.

Example 04: PHP segment.

```
<?php

// Pandas are awesome!

?>
Pandas rule!
```

Let's execute the file again. What's the output that we get?

Example 05: Output.

```
Pandas rule!
```

> Hey wait! Where's the rest?

Well spotted, my soon-to-be developer! There's a section of our file missing. This is because everything between our PHP tags is treated as PHP code, and is processed accordingly.

So what are the PHP tags? Well, you've met the PHP opening tag already. Do you remember our beautiful friend `<?php`. The `<?php` tag marks the beginning of our PHP code. So when does it end? Well that's where the `?>` tag comes into play.

Now that you know how the PHP tags work, it's easy for us to spot the PHP code in this file. It's the following line.

Example 06: Comment.

```
1  // Pandas are awesome!
```

So what does this line do? Nothing. It's known as a comment. It helps developers to document their code. Don't worry. We'll learn more about comments later.

Well that was a nice short chapter, wasn't it? Now it's time for some good news. In the next chapter, you'll be writing your first ***real*** lines of PHP code.

Excited? Then why wait! Flip that page.

5. Basic Arithmetic

Now I'm sure you've heard that programming is all math. Right? Well, it's time for math. Let's get started.

$$\left|\sum_{i=1}^{n} a_i b_i\right| \le \left(\sum_{i=1}^{n} a_i^2\right)^{1/2} \left(\sum_{i=1}^{n} b_i^2\right)^{1/2}$$

Now solve for X.

Just kidding. There's no X in that equation. In fact, it's not even an equation, so that was a terrible joke. Hey! They can't all be side-splitters. The truth is, I have no idea what that mess does either. We aren't all math gurus.

Statements

Let's try something that's a little bit closer to my level of mathematics. You know how to make a PHP file, and you know how to open and close PHP tags. So let's jump straight into a PHP file. We'll call it `math.php`. Here's the content.

Example 01: Addition.

```
1  <?php
2
3  3 + 3;
4
5  ?>
```

Hold on a second. We aren't going to output anything after our PHP code. Why bother with the closing tag? The truth is, most PHP developers omit this tag if no content follows our PHP code. Let's do that.

Example 02: We don't need a closing tag.

```
1  <?php
2
3  3 + 3;
```

Much better!

Right, just in case your math skills aren't quite as sharp as my own, let me help you out a little. When you add three to three, you get six. Okay, now you're ready.

The line `3 + 3;` contains a statement. It's a line of PHP code that will be evaluated by PHP. They normally end with a semi-colon. It looks like this: `;`. You're going to forget about them all the time at first, but don't worry, soon you'll even be ending your sentences with them;

Given that you now understand basic addition, what do you think will be the output when we execute this file?

> Seven point five.

Well, special reader, let's see if you're right. Go ahead and run `php math.php` to see what happens.

Example 03: Output.

```
1  [nothing here]
```

Woah! Nothing. This language is stupid. Let's give up. Okay, I'm kidding again. I have a cheesy sense of humour, don't worry, you'll get used to it.

Why didn't we get any output? Well, it's because we didn't tell PHP to output anything. PHP is obedient. Let's go ahead and tell it to give us the answer. We'll use `echo`. It's a PHP language construct that will allow us to see the result of a statement.

Let's alter our statement to include `echo`.

Example 04: The echo statement.

```
1  <?php
2
3  echo 3 + 3;
```

There we go. We place `echo` before the statement that we want to see the result of. Let's try running our application again! Here we go...

Example 05: Output.

```
6
```

Woohoo! Six! **NOT SEVEN POINT FIVE**! Now that's what I'm talking about. We get to see the result of our first statement evaluation with PHP. That's exciting stuff, right?

> I could have done that on a calculator.

I know, I know. It's not exactly rocket science. Rocket science will be covered in a later chapt... Wait, I've already told that joke in another book. I need to get some new material.

Arithmetic Operators

I know that our `3 + 3` example is simple code, but we'll soon get to bigger and better things. Did you know that there are more mathematical operators? I'm sure some of these ring a bell.

```
+ Addition
- Subtraction
* Multiplication
/ Division
% Modulus
```

Now, I'm sure you'll have seen some of these operators before. I know that multiplication and division look a little different to the signs that you may have learned about in school. This is common to most programming languages, and you'll find that the division sign is easier to type on a keyboard. Don't let them worry you, before too long you'll be completely used to them.

If you've not used the 'Modulus' operator before, then it's simple to explain. It can be used to calculate the remainder of a division. For example, the operation '3 % 2' would result in the figure '1'. It's commonly used to determine whether a number is odd or even by dividing by two.

Now let's give PHP something hard to think about, shall we?

Example 06: Harder math.

```
<?php

echo 4 + 3 * 2 / 1;
```

So, what's the result? Well, it can be difficult to calculate in our heads because we don't know which order to process the pairs of calculations. Should we add three to two first? Or maybe divide two by one first. Hmmm. Tricky!

Of course, in mathematics, we learn to use rounded brackets to separate the concerns of an equation. We can do the same with PHP. Let's give it a go.

Example 07: Brackets for separation of concerns.

```
<?php

echo (4 + 3) * (2 / 1);
```

Now we can be sure that `4 + 3` and `2 / 1` are evaluated first, and the resulting values are multiplied. Great, we run our script and get the result...

Example 08: Output.

```
14
```

Awesome, but isn't that cheating? What would we get without the brackets? Let's take them out again.

Example 09: Without brackets.

```
<?php

echo 4 + 3 * 2 / 1;
```

So what's the result? Let's run our script.

Example 10: Output.

```
10
```

That's a totally different figure. Why is that? Well, it's because PHP isn't handling our operators in the same order. Let's take a little time to learn about the order of our operators.

Here's how PHP handles the order of operators.

```
* Multiplication
/ Division
% Modulus
+ Addition
- Subtraction
```

The operator with the highest priority can be found at the top of the table. So this means when PHP examines `4 + 3 * 2 / 1` it will first calculate `3 * 2 = 6`, then `6 / 1 = 6` and finally `4 + 6` to give us the answer `10`.

When I'm writing mathematical lines of code, I like to use brackets to avoid any confusion. I also find that it helps to clarify the intent of the line, causing it to be more readable.

Procedure

PHP code is parsed procedurally. This means that it is read and executed on a statement by statement basis. While it's possible to put more than one statement on a line, this is uncommon amongst PHP developers. This means that we can also approach the code line by line. We can see this in action by adding more statements to our PHP file. Let's try the following.

Example 11: Multiple statements.

```
<?php

echo 2 + 2;
echo 3 + 3;
echo 4 + 4;
echo 5 + 5;
```

Now, let's execute the file...

Example 12: Output.

```
46810
```

> FORTY SIX THOUSAND GIGAWATTS!?

Calm down reader! We only told PHP to output the results, not to place spaces or newlines into the output. This means that PHP has calculated the values correctly. If we space out the result that PHP has given us like so...

Example 13: Output with added clarity.

```
4 6 8 10
```

...then we see that the calculations are in fact correct. It's just that PHP is very obedient and has outputted the values directly after one another.

I've mentioned many times before that PHP is a flexible and lenient language. Let's put that to the test, shall we? Up until now, our statements have a single space between each 'word' (or number). Let's add some extra spaces in an inconsistent format to see what happens. Here's our modified code.

Example 14: White space.

```
<?php

echo    2           +   2;
echo                   3   +3;
echo 4+4;
echo    5+     5        ;
```

While it doesn't look very pretty, if you were to execute the code you'd find that it will work perfectly. PHP doesn't care about the amount of white space between the words within its code. It just deals with it. (Insert dog with shades...)

You'll notice that some of the arithmetic operations, for example `4+4`, don't require a space at all. While this is true, it isn't consistent with all syntax variations. For example, consider the following snippet.

Example 15: No whitespace after echo.

```
<?php

echo5 + 5;
```

If you attempt to execute this script, you'll find that PHP will throw a notice 'Use of undefined constant echo5 - assumed 'echo5'. This is because it doesn't know what the word `echo5` is telling it to do. For this reason, it's always best to place at least one space between your words.

As for statements, if we were masochistic we could choose to put all of the statements on a single line. Here's an example.

Example 16: Multiple statements, single line.

```
1  <?php echo 2 + 2; echo 3 + 3; echo 4 + 4; echo 5 + 5;
```

This is perfectly valid PHP, but you won't find many developers doing it. Having a single statement on each line makes it much easier to read and understand a source file. It also causes problems for version control systems!

We've seen that PHP doesn't care if you use multiple spaces in its source code, but it also considers a newline a whitespace character. This means that the following snippet is completely legal.

Example 17: One statement, multiple lines.

```
1  <?php
2
3  echo
4  2
5  +
6  2
7  ;
```

Don't believe me? Go ahead and try it! While the code functions as intended, it's not exactly the most readable piece of code. If I catch you writing code like this then you're due for a spanking!

There is one practical use to breaking a line, however. If the line is exceedingly long, then it also becomes a readability issue. We can resolve this issue by applying a new line at an appropriate reading length. Many developers also apply four spaces (or your current tab setting) to the next line to indicate that it is a continuation. This is similar to how formal works of text use an indented sentence to indicate a new paragraph.

Here's an example of a line break for readability purposes.

Example 18: Clean line-breaking.

```
1  <?php
2
3  echo (3 * 5) / (7 / 12) * (7 * 6) + (7 % 3)
4      + (6 + 7) * (12 / 3);
```

That's some serious math, but hopefully, you will find it much easier to read.

It's also worth noting that you can also place empty lines within your code to add clarity. Here's an example.

Example 19: Extra line breaks for clarity.

```
1  <?php
2
3  echo 3 + 2;
4
5  echo 7 * 7;
6
7  echo 5;
```

So you see, PHP can be extremely flexible, but don't forget to add that end of line semicolon because it will never forgive you.

EVER;

6. Variables & Assignment

Now we're getting to the meat and potatoes! Variables are an extremely useful and well-abused part of the developers toolkit. Let's get started, shall we?

Tiny Boxes

I'd like you to think of variables as tiny little boxes that we keep things in. Variables are words that start with a dollar `$` sign. Let's take a look at an example.

Example 01: Basic assignment.

```
1  <?php
2
3  $three = 3;
```

If you think of the variable `$three` as a little box, then we've put the value `3` into it. That's what the equals sign does. In math we use the equals sign to indicate the result of an equation, however, in PHP, it's an entirely different story.

In PHP, the equals `=` sign is known as the assignment operator. It's used to ***set*** something. We are telling PHP to ***set*** the variable `$three` to the number `3`.

If you execute the script we have created above, you'll find that PHP doesn't output anything at all. This is because assignment is purely assignment. We aren't telling PHP to output anything. However, now that we have set the variable `$three` to the value `3`, we can use the `echo` construct on it.

Example 02: Echoing a value.

```
1  <?php
2
3  // Set our variable to the value three.
4  $three = 3;
5
6  // Output the value of our variable.
7  echo $three;
```

First, we set our variable, and then we use the `echo` construct to output the value that it's holding. If we execute our code, then we receive `3` as output.

This is great because it means we can give nicknames to things. You know, just like those mean kids at school. For example, the number '3.14159265359' is a very beautiful number to lovers of circles, but it's awfully hard to remember, isn't it? Let's give it a nickname. We'll call it Pete. No, wait, I have a better idea.

Example 03: An appropriate variable name.

```
1  <?php
2
3  $pi = 3.14159265359;
```

There, now we have created a new variable called `$pi` that holds the value `3.14159265359`. This means that we can use the variable anywhere in our code to perform calculations. Here are some examples.

Example 04: Using variables in statements.

```
1  <?php
2
3  // Assign pi to a variable.
4  $pi = 3.14159265359;
5
6  // Perform circumference calculations.
7  echo $pi * 5;
8  echo $pi * 3;
```

After setting `$pi`, we can use it in other statements to perform calculations.

We can declare and assign as many variables as we like, but there are some rules that we need to follow when choosing names. Variable names can contain numbers, letters, and underscores. However, they ***must*** start with either a letter or underscore, never a number! They are case sensitive, which means that `$panda` is different to `$pAnda`. Here are a few examples.

Example 05: Naming variables.

```php
<?php

$panda = 1;      // Legal
$Panda = 1;      // Legal
$_panda = 1;     // Legal
$pan_da = 1;     // Legal
$pan_d4 = 1;     // Legal
$pan-da = 1;     // Illegal
$4panda = 1;     // Illegal
```

While variable names can contain underscores and start with capitals, it's a common practice to use a naming format known as `camelCasing`. Don't worry; it doesn't require a camel.

camelCased names start with a lowercase character. Variables that are to be named with multiple words will have the first character of subsequent words capitalized. Here are some examples.

Example 06: camelCased variable names.

```php
<?php

$earthWormJim = 1;
$powerRangers = 1;
$spongeBobSquarePants = 1;
```

Do you remember how our statements return a value? Well, our assignments are also statements. Can you guess what this means? That's right; they also return a value. We can prove this by using our good ol' friend the `echo` construct.

Example 07: Statements return a value.

```php
<?php

echo $panda = 1337;
```

We receive the number `1337` as the output. This is because the assignment of the `$panda` variable is performed before it is outputted. This process allows us to use a clever trick. It's not something you're going to use very often, but I think it's a pretty cool trick to know. Go ahead and take a look at this example.

Example 08: Multiple assignment.

```
<?php

$firstPanda = $secondPanda = $thirdPanda = 1337;
```

The snippet above might look a little crazy, but it makes more sense if you read it from right to left. The `$thirdPanda` is assigned the value `1337`, next the `$secondPanda` is assigned the value of `$thirdPanda`, and finally the `$firstPanda` is set to the value of the `$secondPanda`. This means that all variables are set to the final value. Neat, right?

Just my type

Until now we have been working with numbers. It would be boring if those were the only types of values that we can use, right? I think it's about time we examined the other possibilities. Here are some of the common values used within PHP applications.

- integer
- float
- boolean
- string
- null
- array

There are a few more, but let's not complicate matters right away. We need to learn little by little. You don't want knowledge overload!

Let's take a look at these types one by one. First, we have integers. These are whole numbers; we've been using these in our previous examples.

Example 09: Integers.

```
<?php

$panda = 2;
$redPanda = -23;
```

Floats are floating point numbers. They have decimal points, and thus contain fractions. They can be used in a similar fashion to integers. In fact, we've used one already. Do you remember our friend `$pi`? That was a float. Let's move on to something new shall we?

Example 10: Floats.

```php
<?php

$panda = 2.34;
$redPanda = -23.43;
```

Booleans are binary data types. No, don't panic! We aren't going to do any binary arithmetic. It's just a way of expressing that they can be one of two values. A boolean can either be `true` or `false`. Later on, we'll take a look at how boolean values can be used to change the flow of our application.

Example 11: Booleans.

```php
<?php

$panda = false;
$redPanda = true;
```

Next, up we have the 'string' value. Strings are used to store a word, a character, or a sequence of text. Strings are special, so I've decided to dedicate a short chapter to them. We'll come back to this!

Example 12: Strings.

```php
<?php

$panda = 'Normal Panda';
$redPanda = "Red Panda";
```

Null is a special value. It is nothing. Nil. Zero. Well, it's not zero. Zero is numeric, and we can use an integer for that. Nulls are exactly nothing. Null is the value that a variable has before assignment has been performed. It's a useful value, and you're going to see a lot of it in the future.

Example 13: Null values.

```php
<?php

$noPanda = null;
```

Arrays are another special type of value. In fact, this is my favorite one of all. So much that I've decided to dedicate a full chapter to them. For now, all you need to know is that it's a value that holds a collection of other values. Woah! Inception stuff, right?

Example 14: Arrays.

```php
<?php

$countThePandas = [1, 2, 3];
$morePandas = array(5, 6, 7, 8);
```

Advanced Assignment

In a previous chapter, we discovered the operators that we can use on variables, and we've mastered the assignment operator. So what happens when we put them both together? Will it create a new black hole and consume the entire universe? I'm feeling a little daring, shall we find out?

Example 15: Assignment with addition.

```php
<?php

// Set a value.
$panda = 3;

// Attempt to create a black hole.
$panda += 1;

// Universe withstanding, dump the value.
var_dump($panda);
```

First, we set a variable to the integer value of three. Next, we've plopped the addition operator onto the front of the assignment operator and supplied another integer value of one.

We can use the function `var_dump()` (more on functions later!) to interrogate not only the value held within a variable but also its type!

What did we get back from the dump?

Example 16: Output.

```
int(4)
```

Awesome! The universe is saved. It looks like we have a four? Well, I suppose that makes sense. We know that `$a + $b` returns a value without setting it, and we know

that the assignment operator is used to set the value of variables. This does both. We're telling PHP to set the value of `$panda` to its current value plus one.

You can use this syntax with any of the operators that we've discovered so far. There's only one catch. Don't place the operator on the other side of the equals sign. Trust me; I tried it. A portal opened to a dark underworld, half dinosaur, half human creatures broke through and began to terrorize Cardiff. Only with the aid of a homemade flamethrower (powered by PHP) was I able to fend off the vicious creatures. I'd hate to see it happen to you. Please be careful!

Next up, we have the incremental operator. Let's not forget the decremental operator too. She tends to get a little less attention. In fact, let's showcase her abilities.

I prefer to lead with an example. Consider the following snippet.

Example 17: – After.

```
<?php

// Set a value.
$panda = 3;

// Decrease the value.
$panda--;

// Dump the value.
var_dump($panda);
```

There in the middle, do you see her? The beautiful decremental operator. We simply place two minus signs after the variable. What does it do? Well, here's the result of the code snippet.

Example 18: Output.

```
int(2)
```

As we can see, the value of `$panda` has been decreased by one. It's a quick shortcut to decreasing a value. Likewise, using a `++` can be used to increase a value. Those are the only two operators that work, though. Don't you be cheeky and try to use the multiplication operator. It just won't work as you expect it to!

I wonder what will happen if we put the operator before the value? Let's have a go, shall we?

Example 19: – Before.

```php
<?php

// Set a value.
$panda = 3;

// Decrease the value.
--$panda;

// Dump the value.
var_dump($panda);
```

What's the answer? Aren't you excited?

Example 20: Output.

```
int(2)
```

Oh, it's the same. Well that was rather boring, wasn't it? I know a little secret. It's not the same. Sure, the value we received back looks identical, but my example doesn't do it credit.

Let's craft a different example. We'll show the state of a value **before** the operator is used. We'll examine the result of the statement **when** the operation is used, and finally, we'll examine the value **after** the operator is used. We don't expect the after-value to be any different.

Example 21: The stages of –.

```php
<?php

// Set a value.
$panda = 3;

// Dump BEFORE.
var_dump($panda);

// Dump DURING.
var_dump(--$panda);

// Dump AFTER.
var_dump($panda);
```

Let's execute the code. What are the three values we receive?

Example 22: Output.

```
int(3)
int(2)
int(2)
```

The first value is three. We must have expected that I mean, all we did was set it, right? The result of the statement using the decremental operator is equal to two. The resulting value is also two. That means that the value is decreased on the second line.

Let's move the operator to the other side of the value, shall we? Like this:

Example 23: The stages of – part two.

```
<?php

// Set a value.
$panda = 3;

// Dump BEFORE.
var_dump($panda);

// Dump DURING.
var_dump($panda--);

// Dump AFTER.
var_dump($panda);
```

Look very closely to spot the difference. Let's take another look at the result.

Example 24: Output.

```
int(3)
int(3)
int(2)
```

Hey!? That middle value is different! Why hasn't it been decreased? Well, by swapping the operator, we've told PHP to decrease the value **AFTER** the current line. The result of the operation line is the same value as it was initially.

Let me summarize.

```
$value-- - Change value *after* current line.
--$value - Change the value on the current line.
```

Why is this useful? Well, here's a use for you. I'm sure if you're creative you will find more. Using the operator that changes **after** the current line, we can set another variable to its value, and decrease the original value on the same line. Like this:

Example 25: Assign and increase.

```
<?php

// Set a value.
$panda = 3;

// Assign, and then increase.
$pandaFriend = $panda++;
```

What we've done here, is saved a line. It's a bit of a shortcut. Here's how it would look if not for the incremental operator.

Example 26: Incrementing explained.

```
<?php

// Set.
$panda = 3;

// Assign.
$pandaFriend = $panda;

// Increase.
$panda = $panda + 1;
```

In a later chapter about loops, you'll find another use for this operator. In the next chapter, we'll be taking a closer look at strings.