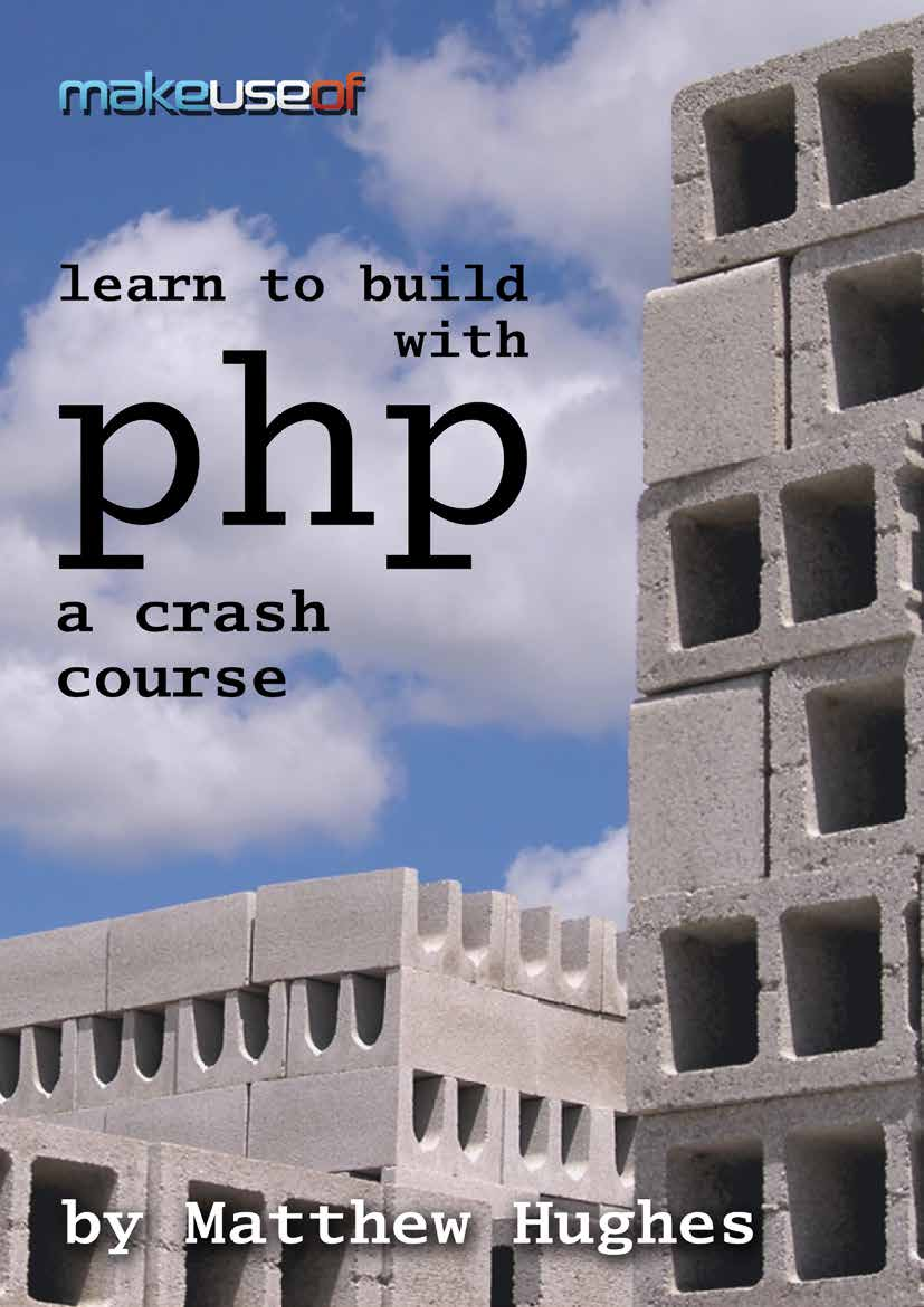
makeuseof

learn to build with php

a crash course

by Matthew Hughes



by Matthew Hughes
http://www.matthewhughes.co.uk/

Published March 2014

share:

Table Of Contents

1. Introduction

What was your favorite subject at school?

If you're anything like me, I bet you loved the arts. The humanities. You know, the subjects derided by some as being vacuous and nebulous in nature, but regardless you loved studying them because you loved being creative.

I bet you never thought of your IT classes in the same way you thought of English Language or art; as a purely creative endeavor.

That's a pity. Learning to program is a bit like doing a creative language class. You have an idea, and you can execute that however you like. It's pure creativity, but instead of baring your soul on paper, you are commanding a computer to do your bidding. If you can dream it, and if you can describe it in a way your computer understands, then you can make it.

The way we talk to computers is through abstractions called programming languages. There are a whole bunch of these out there, each with their own advantages, disadvantages and truly bizarre idiosyncrasies. They're imperfect by nature, but people use them to create incredible and wonderful things.

One of these languages is called PHP.

You may have heard of it before. This is the language that Facebook, WordPress and Wikipedia use to serve billions of requests, daily. It is the de-facto language used for teaching people to program for the web. It's beautifully simple, but brilliantly powerful.

And in this guide, I'm going to teach you how you can use it to build your own websites.

Do you have a killer startup idea you don't quite know how to execute? Do you want to learn the language used to extend WordPress? Are you just curious about web programming? Do you just want to learn the skills needed to stay relevant in the modern, tech-oriented knowledge economy?

Whatever your motivation, this book aims to teach you the basics of the PHP programming language. But first, let's have a bit of a history lesson.

1.1 The History Of PHP

In the infancy of the Internet, things were a bit… Well? Flat.

Sites weren't particularly interactive, and those who visited these sites were faced with a relatively one-directional experience. Adding even the simplest aspects of user engagement was incredibly difficult, left in the domain of the computer scientists and the expert programmers.

And then Rasmus Lerdorf came around. This Canadian-Greenlandic coder created the PHP programming language, which allowed people to easily add the simplest facets of interaction to their web pages. It was new, it was brave, and it took off almost immediately.

Rasmus Lerdorf couldn't have predicted the impact his idea would have upon the world.

A community began to form, with programmers and companies willingly providing time and money to fuel the development of the language. Slowly but surely, PHP began posing a serious challenge to Sun (now Oracle) and Microsoft, who were hoping to gain traction in the web-development market with their Java and ASP platforms. The rise of the PHP programming language could only be described as startlingly rapid.

That was 20 years ago. An age, in the computer world. Since then, the PHP programming language has become the preferred way for millions of programmers, who use PHP in their jobs, to get involved in open source and to bring their ideas to life. It's a staple of the digital world.

You can become one of those millions. This book will show you how.

1.2 What Are We Going To Cover?

This is a pretty short book, but we're going to cover a lot. In just a few pages, we're going to create a simple clone of Twitter.

Whilst it's not going to have the same feature-set and polish of the popular micro-blogging site, but we will be able to post 140 character messages with an account we will log into.

1.3 All About LAMP

By now, we should know that PHP is an incredible language for creating interactive web pages. But we haven't talked about how we turn that code into a real-life product. So, let's do that.

For the most part, PHP code runs within a web server. A web server is responsible for sending web pages to anyone who navigates to a specific domain name or IP address.

The most common choice of web server is the ludicrously popular Apache web server. This open source, cross-platform software project powers the majority of the Internet, with 45% of all websites serving pages from the Apache web server. However, it's helpful to note that there are other web servers available, including LightTTPD and Microsoft's IIS.

This forms the second letter in the LAMP acronym, which stands for Linux, Apache, Mysql and PHP. We know what PHP is. I just explained to you what Apache is. You might have heard of Linux at some point. But what is the 'M' in Lamp? MySQL.

Let's talk about MySQL. I'd put money on your website having to store information that your web application has gathered. What's more, I bet you'd want to store that information somewhere that is structured, safe and organized. Yep, you're going to have to use a database, and for most purposes MySQL is a pretty solid choice. We're going to talk about this later on.

Finally, let's briefly touch on Linux. Most PHP websites are served from systems running the popular, open-source Linux operating system. However, you don't have to use Linux as your development environment if you don't want to. Everything in this book can be used on OS X, Windows and Android

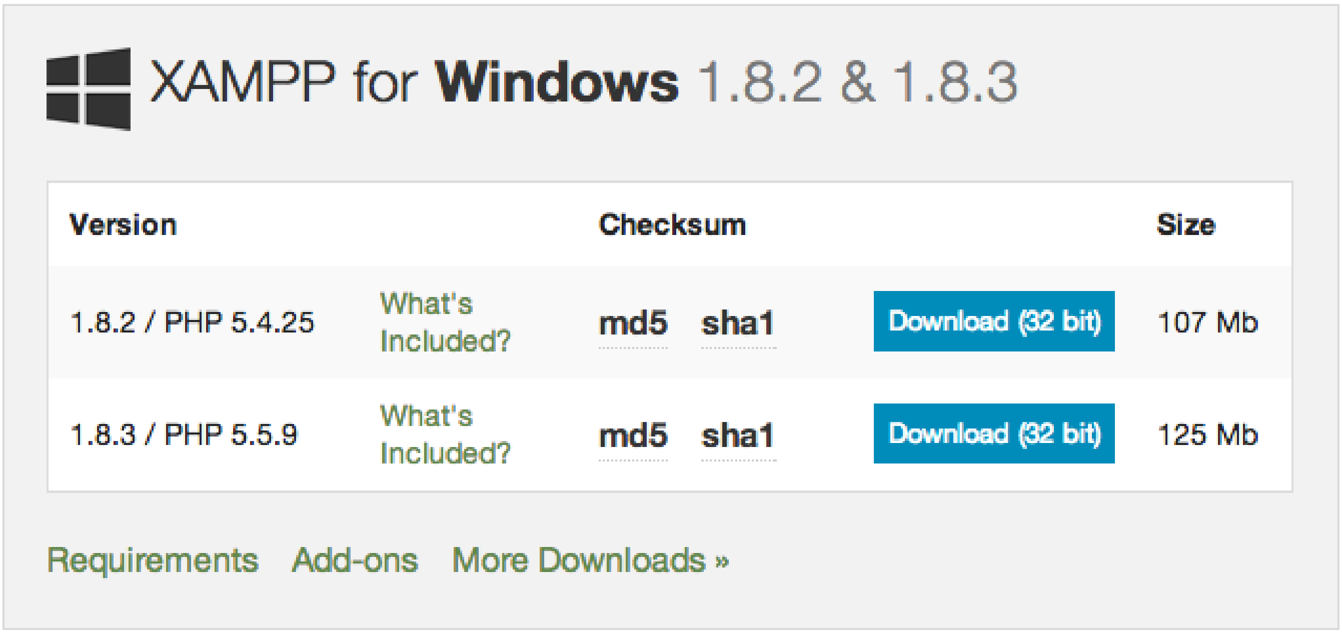
An operating system, Apache, MySQL and an installation of the PHP programming language are the four constituent parts of the most common PHP configurations. But how do we get our hands on them?

1.4 Setting Up Your Development Environment

Unlike some programming languages, setting up a PHP development environment is easy. Indeed, there are a number of packages which do all the hard-work for you, and save you the hassle of installing each component (PHP, MySQL and Apache) individually.

Windows

The easiest way to set up a PHP development environment in Windows is with XAMPP by Apache Friends. This package includes MySQL, a copy of the PHP programming language and the Apache web server, as well as an administrative control panel, and plugins for SSL (the protocol used to encrypt traffic on a network) as well as sending Email.



XAMPP for Windows 1.8.2 & 1.8.3

Version		Checksum			Size
1.8.2 / PHP 5.4.25	What's Included?	md5	sha1	Download (32 bit)	107 Mb
1.8.3 / PHP 5.5.9	What's Included?	md5	sha1	Download (32 bit)	125 Mb

XAMPP is free, and can be found on the Apache friends site. It's important to stress that there are two versions of XAMPP available. One has a version number of 1.8.2 and runs a slightly older version of the PHP programming language, and the other has a version number of 1.8.3 and runs a current version of PHP. Whilst I would strongly recommend that you download the latest version, the contents of this guidebook should work with both.

Linux

Linux is a bit tricky. There are a number of operating systems that exist under the 'Linux' umbrella, although they each handle the installation of software packages in different ways.

If you're using Ubuntu and any distribution which uses the Ubuntu repositories, you can run:

*sudo apt-get install lamp-server^*

This will install a LAMP server, with all the components required for following this article. This process is explained in greater depth in this article, where I show you how to install a LAMP server as part of installing the WordPress blogging platform.

These instructions will not work on distributions that use YUM or RPM for their package management, with the instructions for setting up a LAMP server differing significantly. I would recommend you to have a look at the documentation that came with your operating system.

However, there is another option. Remember XAMPP? Well, it just so happens that it comes with support for Linux, and can be downloaded here. However, where possible, I would strongly recommend that you install your LAMP server through your package manager.

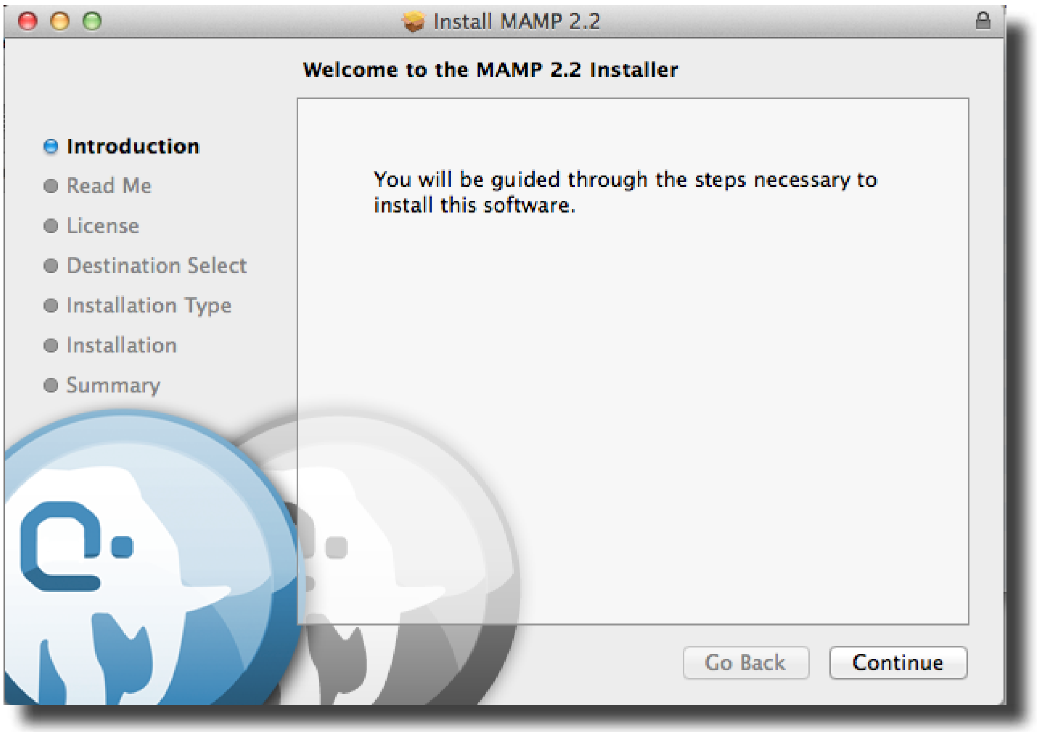
There are a few reasons for this. Firstly, it would be slightly better integrated with your operating system and can be easily updated. Furthermore, installing PHP via the command line is good practice should you ever deploy your application to a VPS server.

OS X

I use OS X as my main development platform. I like the flexibility it provides me, and setting up a PHP development environment in OS X is insanely easy.

I'm quite partial to MAMP. This comes in two products, with one being free and the other costing $59 USD (or €39). However, the free version is more than adequate for the purposes of this guide book.

Getting MAMP is a matter of grabbing a ZIP file from the website, double-clicking a pkg file and pressing 'continue' as often as necessary.



You will be guided through the steps necessary to install this software.

As before, it's entirely possible for you to create a PHP development environment using XAMPP, which is also ported to OS X. It's entirely up to you.

Android

Android? I mean, Android is great for sending tweets and killing time on Angry Birds. But software development? Nah. right?

Wrong. If you've bought an Android cell phone in the past year or so, odds are quite good that it's running a CPU that's just as powerful as any VPS you'll get for under $10. And that means that it's good enough to run PHP, Apache and MySQL.

There are a lot of Android LAMP servers on the market, but I really like Palapa Server. It runs nicely on an aging Nexus 7 tablet, and I've even managed to shoehorn Android onto it without any real difficulties. It's not the ideal development environment, but it's possible.

1.5 Choosing The Right Text Editor

You're probably familiar with what word processors are. Odds are good that you've used Microsoft Word, Open Office or Google Docs to write letters, school assignments or business documents.

But you might not know that it's not possible to use a regular word processor to develop software and websites. Why is that? Mainly because when you write a document, you leave all sorts of extraneous markup and formatting in the file. The end product isn't just the words you write, but also the alignment of each word and its styling.

As a result, when writing code, we use text editors. What are they? Simply put, these allow you to write files which are saved in purest plaintext. No formatting. Just characters.

When writing code, I tend to use Sublime Text 2. It comes with an indefinite free trial (although, it does occasionally nag you to upgrade), and heaps of features, which makes writing software with it a joy.

In particular, it comes built in with syntax highlighting for PHP, Javascript and HTML, which makes it really easy to read the code you produce. You can download Sublime Text 2 here, and it is available for Linux, Windows and OS X. If you're not convinced, you can read more about Sublime Text 2.

If you're on Android, you'll find that your choices are quite limited. I'm fond on VimTouch, which is available for free on the Google Play store. Vim has a pretty steep learning curve, but it's well worth a try. Read more about why it's worth giving Vim text editor a chance.

1.6 Prerequisites

We're going to jump straight in to learning PHP. Whilst I plan to gently introduce you to this amazing programming language, there are some things I'm expecting you to understand beforehand.

Specifically, I'm going to expect that you understand how a website is structured with HTML. If you don't know your <p> tags from your <span> tags, that's not a problem. MakeUseOf has a XHTML book which will bring you up to speed. Read through that and once you're feeling confident, read on.

Feeling adventurous? Why not learn about the latest version of HTML with our HTML5 e-book? Whilst it's not essential, it might help you later on.

2. Hello World!

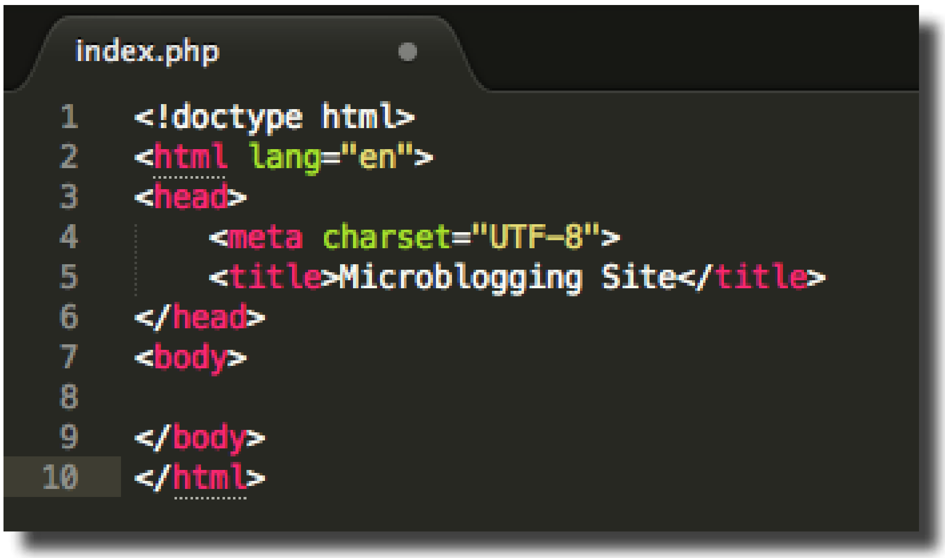
Time for a whistle-stop tour of PHP. And where better to start than the traditional ‘Hello World!’ program.

But first, we’re going to need to know where to store our PHP files. We store them in a place called the ‘Document Root’, which sounds complicated, but it really isn’t. All that means is whatever is stored in this folder will be available to anyone who visits the computer’s IP address with their web browser.

The location of your document root varies on how your PHP environment is set up. If you are using MAMP on OS X, you can find it in /Applications/MAMP/htdocs. If you’ve installed your LAMP server on Linux using your distribution’s package manager, your Document Root directory will most likely be /var/www. On XAMPP, your root directory is located in C:/xampp/htdocs/.

Once you’ve navigated to the Document Root folder, create a file called ‘index.php’ and add the following lines.

```
<!doctype html>
<html lang=”en”>
<head>
<meta charset=”UTF-8”>
<title>Microblogging Site</title>
</head>
<body>
</body>
</html>
```



index.php

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Microblogging Site</title>
</head>
<body>

</body>
</html>
```

Nothing’s going on here, but we do have the skeleton of a web page. This homepage will be found at ‘localhost’. Sometimes it’s followed with a port number, which usually is ’80’, ‘8888’ or ‘8080’. Although, depending on the PHP development package you use, can vary. If you’re unsure, refer to the documentation.

Now, let’s write our first lines of PHP! In between <body> and </body>, write

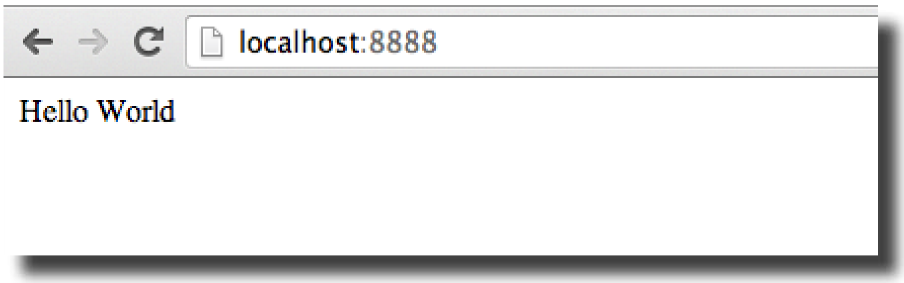
```
<?php echo(“Hello World!”); ?>
```

So, let’s break this down.

All PHP code has to be between a ‘<?php’ and ‘?>’. If it isn’t then the web server will not execute it. Then, we have ‘echo’. As I’m sure you’ve guessed, this function prints content to the browser. Finally, we have the content we want to print out. This is surrounded in parentheses and speech marks. It’s worth noting that parentheses are (for the most part) optional, when it comes to calling a function.

As we have finished the ‘echo’ statement, we finish it with a semicolon. If this is missing, your code will not work.

If it works, you should see 'Hello World' in your browser.



Hello World

2.1 Does PHP Have To Be Surrounded By HTML?

No.

If we're performing an action that should be displayed immediately within the web browser, we can have it nested within the HTML document. This is called inline PHP, and that's what we used to print 'Hello World' to the screen.

However, for anything more complex, we should always aim to put it in its own PHP document. As with the previous example, the file should end with a '.php' extension and all code should start and end with '<?php' and '?>'.

2.3 Basic Language Concepts

Before we go on, let's look at some language concepts in PHP. Whilst this isn't an exhaustive list, it does include the essentials required to be productive as a PHP programmer. Once we've gone through these, we're going to look at using some of these concepts within the context of forms.

Variables

Variables are a concept found in virtually all programming languages. They are used to store a value, which can be retrieved, used and changed later on.

You might be familiar with a language that requires you to specify the value of a variable. These include C#, C, C++ and Java, and usually look a bit like this.

int x = 10;

You also might be familiar with Javascript, where variables are declared with the 'var' keyword.

var x = 10;

In PHP, however, variables are declared with a dollar symbol.

$x = 10;

PHP variables cannot start with a number or a special character other than an underscore. Furthermore, they cannot be called 'this', as this is a reserved keyword.

If Statements

If statements are useful. They allow you to execute code contingent upon a specific condition being met. Consider the following code.

```
$x = 5;
if ($x == 4) {
echo("Hello World");
} elseif ($x == 3) {
 echo("Hello Dave");
} else {
echo("Hello Brian");
}
```

So, the first condition being examined is if $x equals four (note the double equals symbols). Since it isn't, the PHP interpreter will look at the next conditional statement, which is if $x equals 3. If it doesn't, it will move on to the final statement, which will echo out 'Hello Brian' if none of the conditions have been met.

You can check if a variable is empty (also known as 'null', or a 'null value') by preceding it with a ! in the 'if' statement. For example:

```
if (!$x){
echo("x is empty");
}
```

While Statements

While statements execute code repeatedly whilst a condition is being met. Consider the following code.

```
$x = 10;
while ($x > 1) {
 echo($x);
$x = $x - 1;
}
```

This code looks at the value of $x, and if it is greater than one, it will echo the value of $x and then remove one from it. It will do this, until the condition of '$x > 1' is no longer met. That is to say, that it equals 1 or less.

For Loops

For loops are, as a concept in programming, often quite intimidating to beginners. They shouldn't be, though. Whilst they're ostensibly quite complex, they're really easy to understand when broken down. Let's write a simple for loop that counts from one to ten.

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

What's happening here? Well, first we create a variable with a value of 0 ($i = 0;). We then set the condition of 'if $i is less than 10, carry on' ($i < 10). We then add one to I ($i++) and execute the contents of the curly braces, looping back on the original code.

Functions

Functions are a useful tool when it comes to programming. They allow you to write code that is more consistent, and spend less time writing the same things over and over again by compartmentalizing code into a single snippet that can be called when required.

They're also really simple to create. This is how we make a function that echoes out 'Hello World' when called. This function will be called 'sayHello()'.

```
function sayHello(){
echo("Hello World");
}
```

You can also pass functions values. These are known as parameters, and are put within the two parentheses in the first line of the declaration. For example:

```
function sayHello($hello){
echo($hello);
}
```

And can be called as follows:

```
sayHello("Hello World");
```

Finally, we can get functions to return values.

```
function returnHello(){
return “Hello World”;
}
```

And can be used as follows.

```
x = returnHello();
echo(returnHello());
```

2.4 Moving On

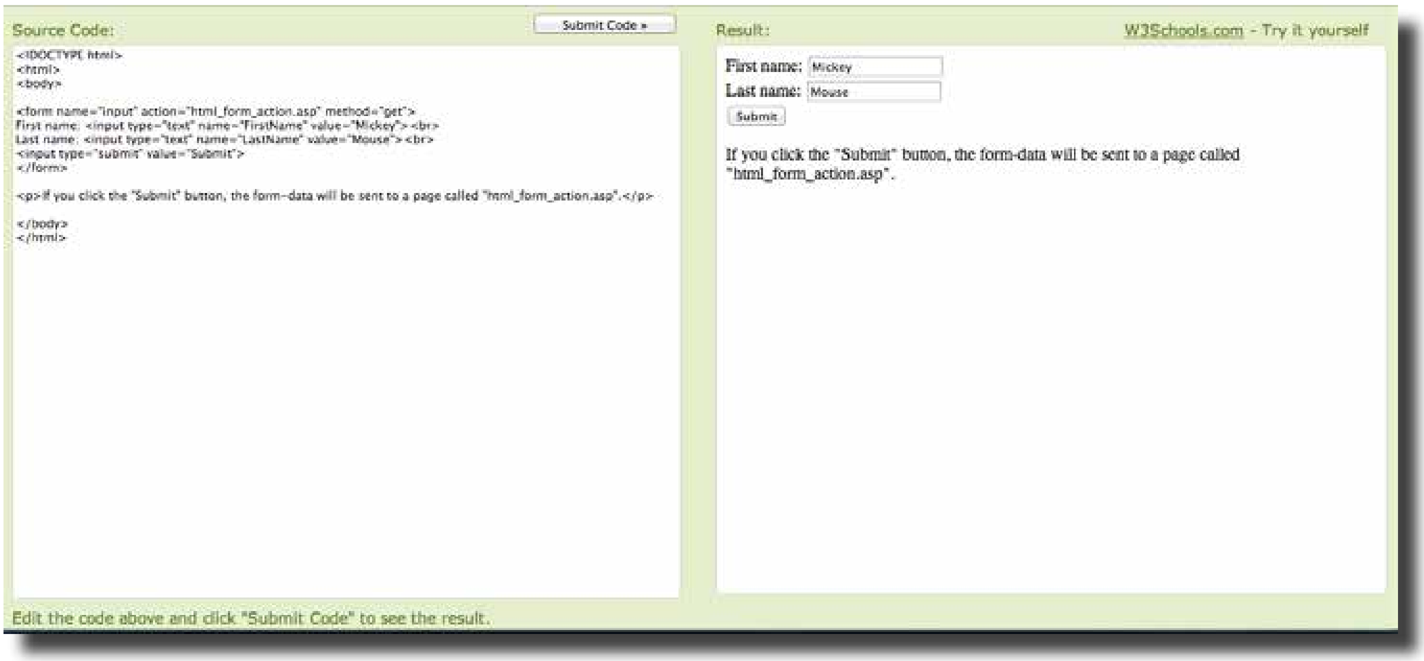
We’ve learned the basics of flow control in the PHP language, as well as how we can use variables and functions to store values and snippets of code for future use. In the next chapter, we will expand upon our understanding of the PHP programming language by looking at how we can capture data with forms, thus forming the first piece of our Twitter clone.

3. Forms

3.1 How Forms Work In HTML

Forms are everywhere.

No, seriously. They are. When you leave a comment on MakeUseOf; when you buy something off Amazon and have to write down your address and credit card numbers; when you compose and submit a Tweet, you are filling out a form.



```
<!DOCTYPE html>
<html>
<body>

<form name="input" action="html_form_action.asp" method="get">
First name: <input type="text" name="FirstName" value="Mickey"><br>
Last name: <input type="text" name="LastName" value="Mouse"><br>
<input type="submit" value="Submit">
</form>

<p>If you click the "Submit" button, the form-data will be sent to a page called "html_form_action.asp".</p>

</body>
</html>
```

First name: Mickey
Last name: Mouse
Submit

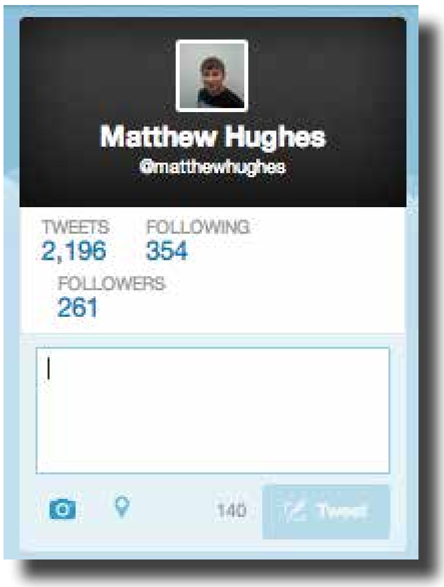
If you click the "Submit" button, the form-data will be sent to a page called "html_form_action.asp".

Edit the code above and click "Submit Code" to see the result.

I know I sound like a broken record, but if you haven't read the MakeUseOf HTML5 guide, you should. You don't need it for this chapter, but it'll show you some cool tricks you can do with forms in the latest version of the HTML markup language.

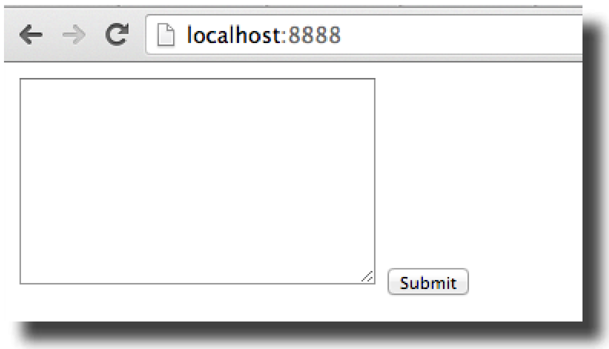
3.2 Creating Our First Form

You've probably seen Twitter before.



The core of a Tweet is a multi-line text box, and a button that submits it to Twitter's servers. So, how do we create a rudimentary version of that in HTML? Well, it looks something like this:

```
<body>
        <form action="postForm.php" method="post">
                <TextArea name="microBlog" id="microBlog" cols="30" rows="10">
                </br>
                </TextArea>
                <input type="submit">
        </form>
</body>
```



So, let's break this down.

A form is enclosed within form tags. Form takes two arguments, with the first being a link to a PHP file which contains the code that handles our form. The second is a method, and this refers to how you send data to the web server. These can either be 'post' or 'get'.

Inside, we've got a TextArea element. Do you know what the difference between a TextArea and input element is?

It's a really subtle difference. They do pretty much the same thing, although a TextArea element allows you to input multiple lines of content, whereas an Input (when not used to submit a form) can only accept one line of content.

Since we're creating a clone of Twitter, we're going to use a TextArea element to capture the post. This element takes a few arguments. The first two are 'name' and 'id', which we've given the value of 'microBlog'. The second two are 'cols' and 'rows', which we have given the values of '30' and '10' respectively. These can be adjusted, as you see fit.

Finally, we have an input element. This has a type of 'submit', and is rendered in the browser as a button. Once pressed, it will pass the contents to 'postForm.php'.

We should have something which looks like this. Not the most pleasant thing in the world, but we'll worry about that later.

3.3 Handling This Input With PHP

In keeping with the gentle pace of this book, we're going to just look at how we can capture the input, and then print it to the screen. We'll look at persistence and storing it in the database later on.

So, remember that postForm.php file we mentioned earlier? Create that in the document root.

Now, add the following lines.

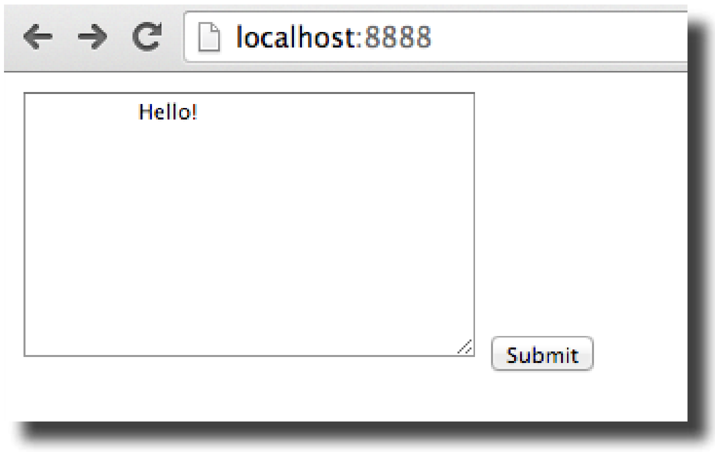
```
<?php
        $microBlog = $_POST['microBlog'];
        echo $microBlog;
?>
```

So, what's that $_POST thing then?

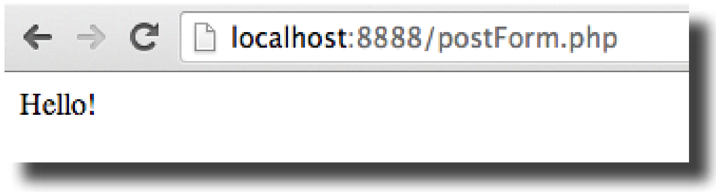
Well, that's what we call a super-global variable. That sounds complicated, but it isn't. In its simplest form, it means that whenever we need to reference a form input, we have to use that. At the end of $_POST, we write the ID of the form input we're referencing. This has to be sandwiched between square braces and quote marks.

We assign the contents of the 'microBlog' to a variable, and then echo it to the screen. Simple, really.

But does it work? Let's give it a try.

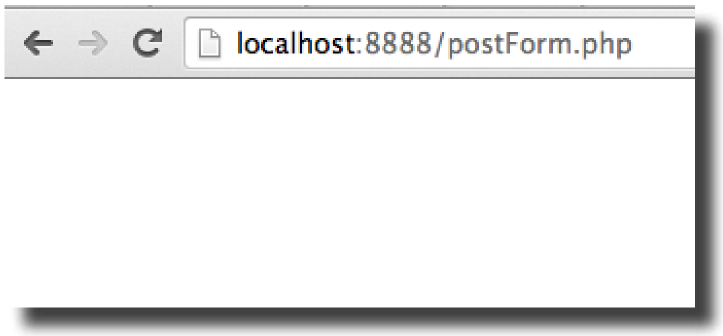


We navigate to our homepage and write something into the form. Then, we press 'submit'.



Hello!

As you can see, our input is returned back to us. Here's a question though. What happens if we close our browser tab and reopen localhost/postForm.php?



There's nothing there. Niet. Nada.

That's because it's not stored anywhere, except in the short-term memory of our web browser. Once it's gone, it's gone.

However, there's an easy way to ensure that our data is retained forever. We have to put it into a database. Whilst at first they're not the most exciting subject ever, they're pretty easy to get the hang of.

And in the next chapter, I'm going to show you how you can store your tweets in a database.

4. Databases

I have a confession to make. I really like databases.

Ever since computers were invented, we've needed to store content. Everything from the password you use to log into Facebook, to the configuration of your computer, to your bank account, is stored in a database. Earlier versions of databases were fundamentally primitive and limited. Since then, they have evolved and advanced, improving the speed and reliability of their functioning. This is largely a result of 50 or so years of development.

One way of storing data is with a relational database. This paradigm of storing data was introduced in the 80s and relies upon a mesh of interconnected tables, with data organized in rows and columns.

There are a huge number of database management systems that use this paradigm (known as RDBMS'), including Oracle, MsSQL and MariaDB. But we're only going to look at one. MySQL.

MySQL is a modern database management system. The number of websites which depend on it is dizzyingly large. It is the database behind WordPress and Facebook, and is based upon the reliable relational database paradigm.

Advantages of the MySQL database are too numerous to mention.

Firstly, it's free; both in cost and in terms of licensing. It can be found running on every platform, having been ported to some of the most obscure operating systems in the world.

Moreover, it's a reasonably lightweight software package, and can run quite comfortably on most low-powered VPS systems.

It's also remarkably easy to get started with MySQL, and there is a huge wealth of information out there for people who would like to get their feet wet with it, including some incredibly detailed and accurate user generated documentation.

In order to interact with MySQL, we need to use the SQL (Structured Query Language) programming language. This allows us to perform complex queries against a database, whilst strongly resembling written English. As a result, it is really easy for beginners to get started with.

But before we get to that, let's talk about MySQL datatypes.

4.1 MySQL Datatypes

MySQL requires that each column in a table be categorized with a specific datatype. For example, someone's age would be categorized as a number, whilst someone's name would consist of a variable number of alphabetic characters.

We specify datatypes to ensure the consistency of the data stored in the table.

Whilst this sounds complicated, I assure you it isn't. What we're talking about is knowing the data that we're planning to retain, and knowing how to classify it.

There are a huge number of datatypes in MySQL. We're only going to talk about two, which are more than adequate for the purposes of this book.

Varchar(x)

When you're inputting content that can contain a variance of numbers, special characters and letters, you're recommended to use a varchar field. These can be of varying length. A field size with a maximum length of thirty characters can be represented with varchar(30).

Integer

When storing numbers, we use an integer value. These can range from -2147483648 to 2147483647, although there are more datatypes available if you need to represent larger numbers. In that case, refer to the MySQL manual.

Other MySQL Datatypes

There are a huge number of datatypes available, which can be used to represent and retain all sorts of data. These include large pieces of text, true or false values and binary files. If you're curious to learn more, have a look at the MySQL documentation.

4.2 Creating Our Database

Okay, now it's time to start working on a database for our Twitter clone.

Firstly, we need to connect to MySQL. If you're using Linux and you've installed MySQL through your package manager, you can connect through the terminal with the following command.

mysql -u root -p

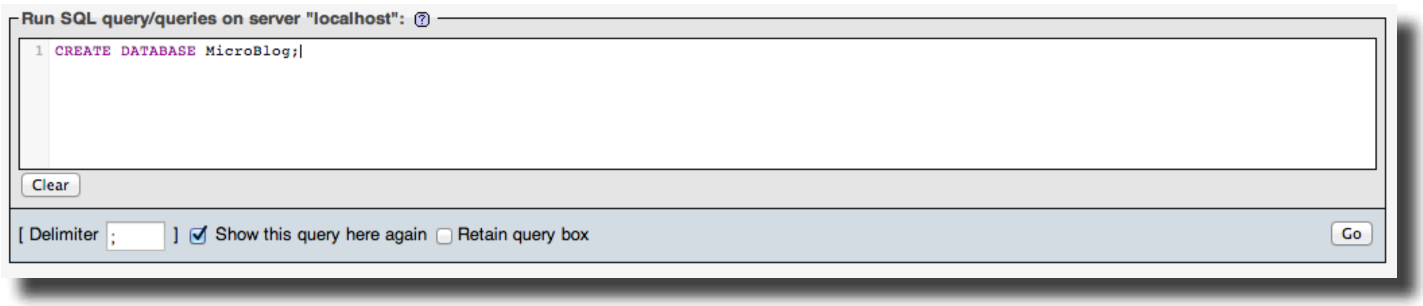
Upon which, you will be prompted for your password.

Users of XAMPP and MAMP can use PHPMyAdmin, which comes bundled in and allows you to edit your database with a nice web interface.

So, first we need to create the database for our website and create a table to hold our microblog posts.

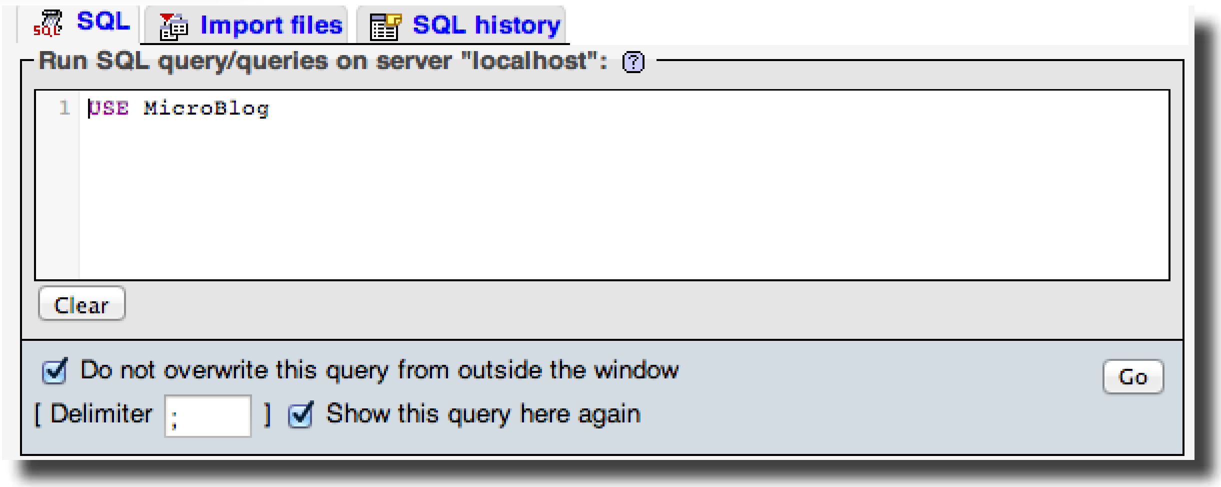
Either through PHPMyAdmin or through the MySQL console, run the following lines.

CREATE DATABASE MicroBlog;


Run SQL query/queries on server "localhost":

```
CREATE DATABASE MicroBlog;
```

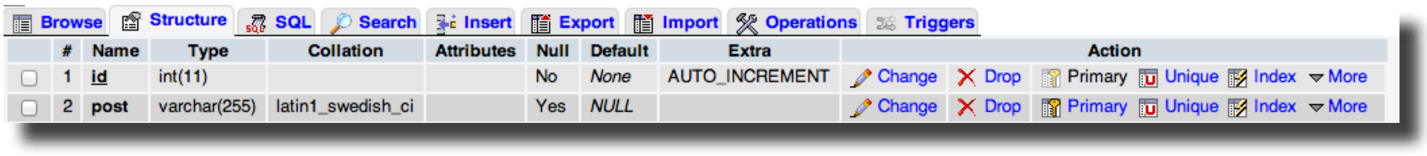
USE MicroBlog;


SQL
Import files
SQL history
Run SQL query/queries on server "localhost":
Clear
Do not overwrite this query from outside the window
Go
[Delimiter ;]
Show this query here again

```
USE MicroBlog
```

```
CREATE TABLE MicroBlog (
          id integer auto_increment,
          post varchar(255),
          primary key (id)
);
```

Unless you see an error message, your table has been created without a hitch.



#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	id	int(11)			No	None	AUTO_INCREMENT	Change Drop Primary Unique Index More
2	post	varchar(255)	latin1_swedish_ci		Yes	NULL		Change Drop Primary Unique Index More

So, a few things you might have noticed there. Firstly, we're ending each statement with a semi-colon, much like we did when we were writing PHP.

Secondly, things are pretty self-explanatory, aren't they? SQL reads like written English, and it's pretty easy to understand what's going on.

That said, there are some things in the CREATE TABLE statement which you may be unfamiliar with. The first is 'auto_increment'. What does this do?

Well, ID is a field that uniquely identifies each post. When we created it, we gave it an attribute of 'auto_increment', and whenever a new row is added to the database, that row is given a unique number. This number counts up by one, for each row that is added.

Finally, what is 'primary key (id)'? Well, we want 'id' to be completely unique. We also want ID to be a field which can identify a row. Making 'id' a primary key ensures that these conditions can be met.

4.3 The Wrong Way To Query The Database

So, how do we insert a post into our database? Great question.

The traditional way of doing this in PHP looked a bit like this:

$conn = mysqli_connect($DBServer, $DBUser, $DBPass, $DBName);

You would create a connection to the server using the hostname, database credentials and database name, and assign that to an object. In this instance, we've called it $conn (for connection).

We then would define the database query we wish to make.

$query = mysqli_query($conn, "INSERT INTO MicroBlog VALUES ('$post')");

And then we would close the connection to the database.

mysqli_close($conn);

So, what wrong with this? Well, it relies upon one key assumption; that any input passed to the database can be trusted.

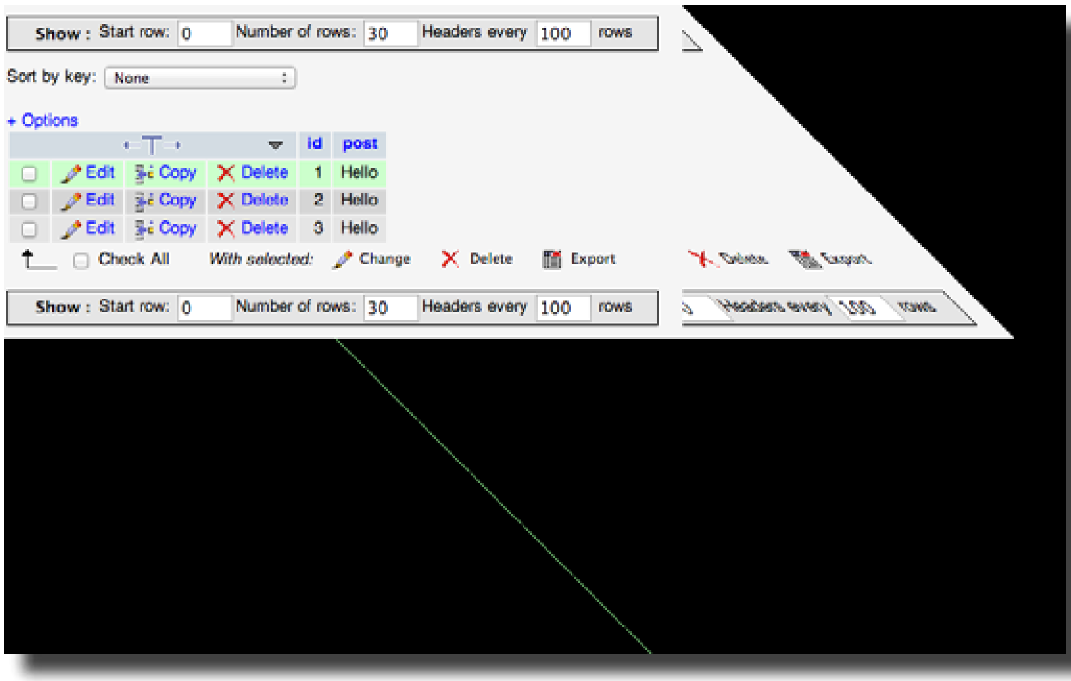
That isn't always the case. If you're not very careful, it's entirely possible to sneak in some arbitrary SQL code, which can then result in data leakages or the defacement of the records stored within your database.

LinkedIn learned about this the hard way. A poorly sanitized input lead to the unintended (and unauthorized) release of thousands of user records in a debacle that was referred to as LeakedIn. It was an incident that cost them dearly, both with respect to the financial cost of making good, as well as the goodwill lost by losing millions of rows of personal information.

So, how do we safely use a database with PHP?

4.4 Meet MeekroDB

MeekroDB is a library that makes it easy to interact with a database without being wide open to an external threat. It has been designed from the ground up to be impervious to SQL Injection attacks. It's free for non-profit use, but if you plan to use it in a commercial project, you need to pay for a license.



Show : Start row: 0 Number of rows: 30 Headers every 100 rows

Sort by key: None

+ Options

				id	post
☐	Edit	Copy	Delete	1	Hello
☐	Edit	Copy	Delete	2	Hello
☐	Edit	Copy	Delete	3	Hello

Check All With selected: Change Delete Export

Show : Start row: 0 Number of rows: 30 Headers every 100 rows

Download a copy of MeekroDB from the official website. Once you've got it, unzip it and place it in your document root directory.

Now, let's go back to your text editor. Open postForm.php and add the following lines.

```
require_once ‘meekrodb.2.2.class.php’;
DB::$user = ‘user’;
DB::$password = ‘password’;
DB::$dbName = ‘database’;
```

Change ‘user', ‘password' and ‘database' with your actual database username, password and database name.

Now, it's time to insert your post into your database. Add the following lines of code.

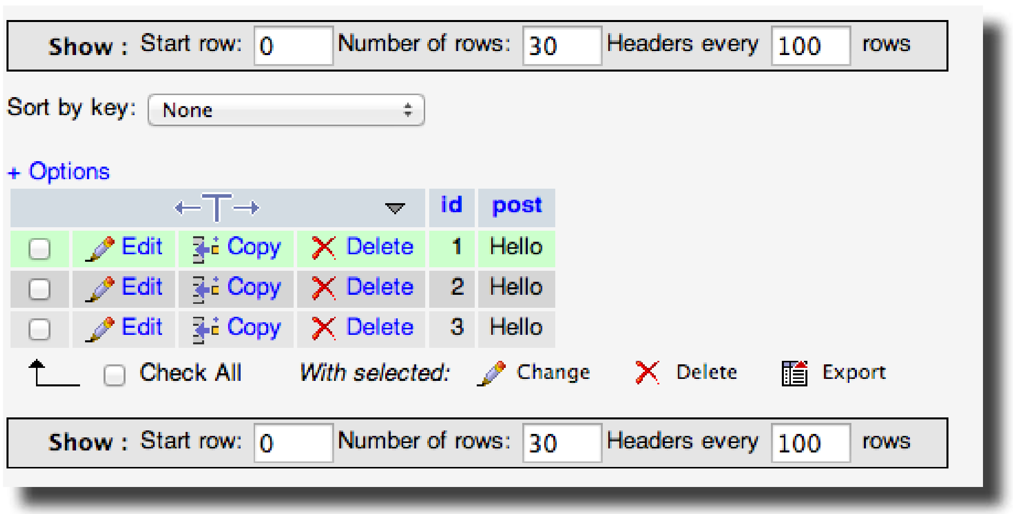
```
DB::debugMode();
$microBlog = $_POST[‘microBlog’];
DB::insert(‘MicroBlog’, array(
                 ‘post’ => $microBlog) );
```

So, let's break this down.

DB::debugMode(); throws out error messages if we do something wrong. As a result, it's really quite handy to leave this in our code, as it makes the process of development that bit easier.

DB::insert is called when we need to insert one or more items into the database. 'MicroBlog' refers to the database table, whilst 'post' contains the message which you've just posted to the website.

We can see if our posts have reached the database by opening up PHPMyAdmin and having a look.



Show : Start row: 0 Number of rows: 30 Headers every 100 rows

Sort by key: None

+ Options

				id	post
☐	Edit	Copy	Delete	1	Hello
☐	Edit	Copy	Delete	2	Hello
☐	Edit	Copy	Delete	3	Hello

☐ Check All *With selected:* Change Delete Export

Show : Start row: 0 Number of rows: 30 Headers every 100 rows

If you're using MySQL on Linux, and you've installed it from the package manager, you can open up MySQL from the terminal and run the following commands.

Use MicroBlog;

*Select * from MicroBlog;*

Simple, right? Give that a go and next, we're going to look at displaying our posts on the homepage. I promise, it's not too hard.

5. Getting Content From The Database

We've introduced MeekroDB already. This is the database library that allows us to interact securely with a MySQL database within the context of a PHP application.

If you've completed the previous chapter, you should have some posts in the database. However, they're just sitting in a database not being used. What a pity! So, let's look at getting them into the browser.

We know that we can use SQL is the language used by MySQL in order to query the database. Whilst MeekroDB allows you to insert content into your database without using SQL, you have to use some SQL in order to retrieve records.

5.1 Selecting And Presenting Results

One such statement is the 'Select Statement'. We have previously used this to see if our files had propagated to the database. We're going to use it again here.

Below the closing form tag, add <?php and ?> and in-between write the following lines of code.

```
 <?php
require_once 'meekrodb.2.2.class.php';
DB::$user = 'root';
DB::$password = 'root';
DB::$dbName = 'MicroBlog';
$results = DB::query("SELECT post FROM MicroBlog");
foreach ($results as $row){
        echo "<div class='microBlog'>" . $row['post'] . "</div>";
}
?>
```

Let's look at this a bit more closely. We know what the first five lines do. We previously used them as part of inserting our posts into the database.

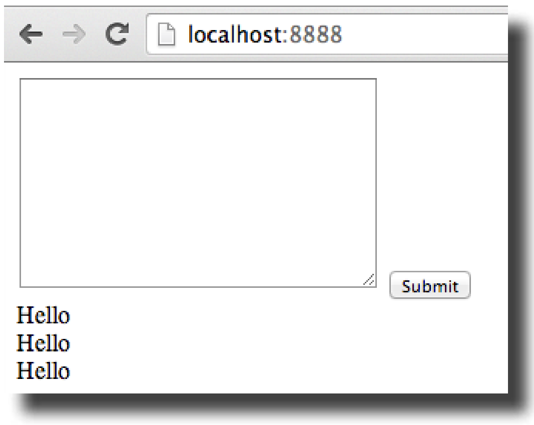
$results = DB::query("SELECT post FROM MicroBlog");

This line selects all posts from the table 'MicroBlog', and then copies them to a variable called $results.

And then it gets interesting. So, $results happens to consist of a number of items. This means that we can iterate over them using something that is strongly reminiscent of those 'for' loops that we previously looked at.

```
foreach ($results as $row){
                echo "<div class='microBlog'>" . $row['post'] . "</div>";
}
```

So, here we're going over each result, and mapping it to a variable. We then print it out in between some 'div' tags. These allow us to encapsulate each post, and apply stylings to each of them.



Hello
Hello
Hello

It doesn't look like much now, does it? Let's change that.

5.2 Styling

Create a new file called 'style.css' and add the following line of HTML code in-between the 'Head' tags.

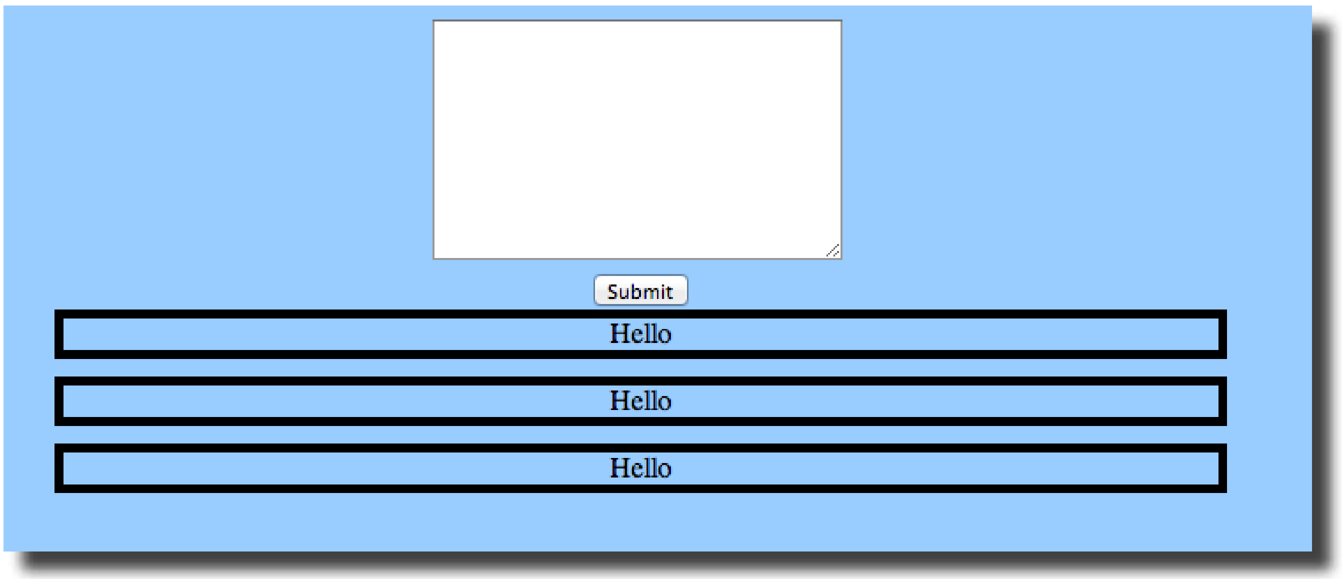
<link rel="stylesheet" type="text/css" href="style.css">

Now, it's time to make each post look a bit more distinctive and noticeable. In 'style.css', add the following lines.

```
body {
         background-color: #99CCFF ;
 }
form {
         text-align: center;
         margin-left: 300px;
         margin-right: 300px;
 }
  .microBlog {
         text-align: center;
         margin-left: 300px;
         margin-right: 300px;
         margin-bottom: 10px;
         border-style:solid;
         border-width:5px;
}
```

I won't insult your intelligence by explaining line-for-line what this does. You can probably work it out, just by reading it. We've given each post a bit of padding, so that it's relatively centered in the screen. We've also given it a border, and aligned the form in the center of the screen.

This produces something that looks a bit like this.



Hello

Hello

Hello

Cool, right?

Now, in our penultimate chapter, we're going to bring together everything we've learned so far, and look at how we can handle logins.

6. Logins And Authentication

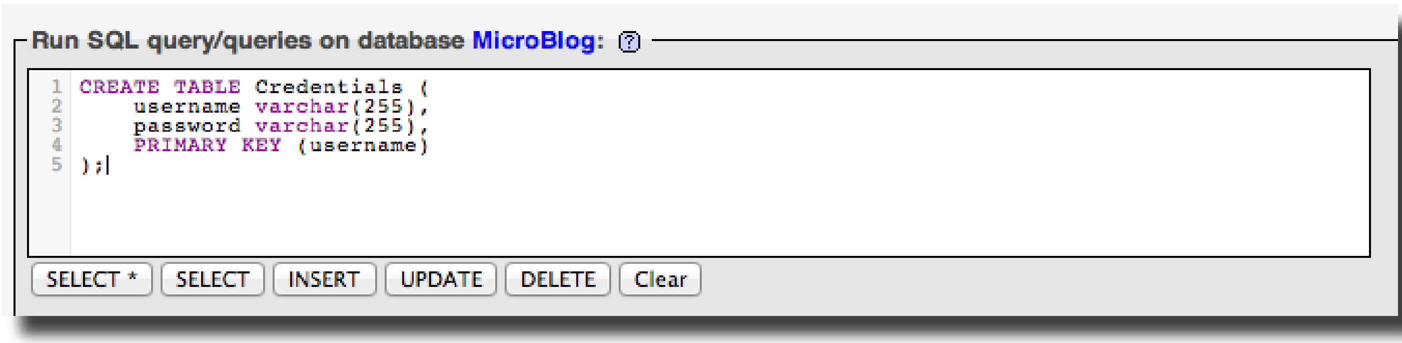
Let's imagine that we only want one person to be able to post updates on our website. It's almost as if we're creating a highly personal Twitter.

6.1 The Users Table

First, let's create a table to handle our login information. This will have two fields. The first is for the username and the second is for the password. This can be represented in SQL as follows:

```
CREATE TABLE Credentials (
         username varchar(255),
          password varchar(255),
          PRIMARY KEY (username)
);
```

As before, run this against your database, either through the terminal or through PHPMyAdmin.



Run SQL query/queries on database MicroBlog:

```
CREATE TABLE Credentials (
    username varchar(255),
    password varchar(255),
    PRIMARY KEY (username)
);
```

6.2 PHP Sessions

Now, we need to handle logins and registrations. We do that with PHP sessions. The way these work is reasonably simple. You have a number of variables that are present throughout a web application, and these can be in a state of presence or otherwise, depending on whether a user is logged in or not.

Before you can handle sessions, you have to first initialize the session. Add this line to the top of index.php.

<?php session_start(); ?>

This line has to appear in every file which accesses the session.

Now, we need to ensure that the post submission form is only visible to those who are logged in. Replace the form we created earlier with the following lines of code.

```
<?php if(isset($_SESSION['loggedin'])){
         echo '<form action="postForm.php" method="post">
                 <TextArea name="microBlog" id="microBlog" cols="30" rows="10">
                 </TextArea>
                 </br>
                 <input type="submit">
         </form>';
}
?>
```

$_SESSION['loggedin'] is a session variable. When it's set, the user will be able to see the form used to create posts. Let's add something else.

```
else {
                 echo '<form action="login.php" method="post">
                    Username: <input type="text" name="username" id="username" /> </br>
                    Password: <input type="text" name="password" id="password" />
                    <input type="submit">
                 </form>';
         }
```

When you browse to localhost/index.php now, you will see that you can no longer post a status, and you are being prompted to log in. It works!

But first, we're going to have to create an account to log in with. So, let's add the following lines to our index page.

```
         if (isset($_SESSION['loggedin'])){
                 echo '<a href="logout.php">Log Out</a>';
         } else {
                 echo '<a href="register.php">Register</a>';
         }
```

If our user is logged in, they will be presented with an option to log out. Likewise, if our user isn't logged in, they will be presented with an option to register.

Now, we need to create a register form. Create a new file called register.php and add the following.

```
<?php session_start(); ?>
<!doctype html>
<html lang="en">
<head>
         <meta charset="UTF-8">
         <title>Register</title>
         <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
         <h2>Register</h2>
         <form action="registerForm.php" method="post">
                 <p>Username: <input type="text" name="username" id="username"></p>
                 <p>Password: <input type="text" name="password" id="password"></p>
                 <input type="submit">
         </form>
          </body>
```

```
</html>
```

While you’re at it, add the following lines to style.css.

```
h2 { text-align: center; }
```

6.3 Registering Users

Now, we need to add some logic for registering a user. Create a new file called registerForm.php and add the following lines.

```
<?php
         require_once ‘meekrodb.2.2.class.php’;
         DB::$user = ‘root’;
         DB::$password = ‘root’;
         DB::$dbName = ‘MicroBlog’;
          $username = $_POST[‘username’];
         $password = $_POST[‘password’];
         $hash = password_hash($password, PASSWORD_DEFAULT);
         DB::insert(‘Credentials’, array(
                 ‘username’ => $username,
                 ‘password’ => $hash
         ));

         header(‘Location: http://localhost:8888/index.php’);
?>
```

A lot of this should be pretty familiar, but some stuff is new. I’ll explain that now.

Security is always at the forefront of the application developer. It’s always embarrassing when a major security breach happens, resulting in a deluge of customer information being leaked. Trust me. It’s never pleasant.

And it doesn’t help that people tend to recycle passwords. Yep, if your site gets hacked and someone gains access to your users’ passwords, it may lead to your users’ accounts on other sites being compromised.

Hashing allows you to encrypt a password with a cryptographic algorithm, making it remarkably difficult to reverse it to its original form. That task is accomplished with a single line of code in PHP.

$hash = password_hash($password, PASSWORD_DEFAULT);

However, the example I provided isn’t enough. Passwords need to be salted. What is this, you ask? Well, it’s another line of defense, and makes it near-enough impossible for your user’s passwords to be decrypted. However, this is a little too complicated for a small guide. As a result, I’d encourage you to do some further reading on the topic. This blog post gives a pretty solid guide to salting passwords with PHP [http://www.sitepoint.com/hashing-passwords-php-5-5-password-hashing-api/].

Finally, we have this curious line.

header(‘Location: http://localhost:8888/index.php’);

This does one task, which is to redirect the browser to the home page. Simple, really.

6.4 Logging In

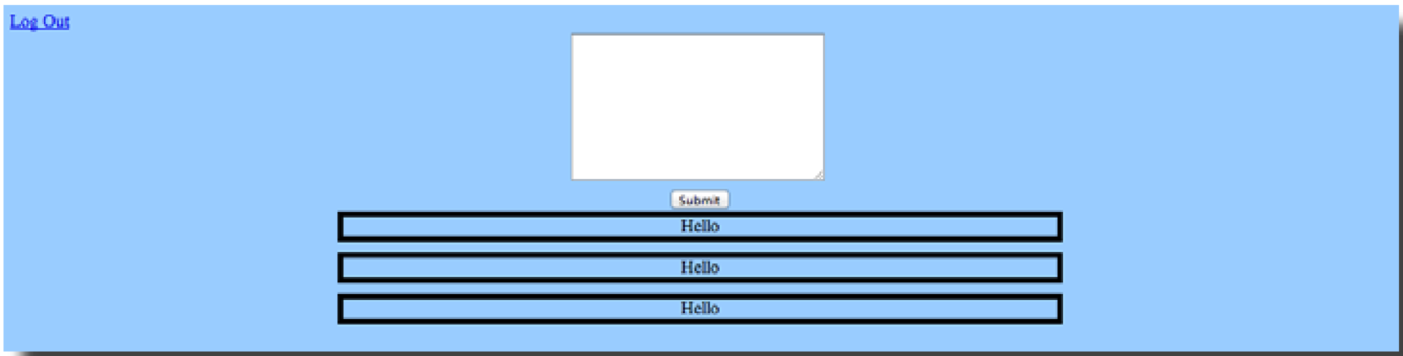
Create a file called ‘login.php’ and add the following lines.

```
<?php
         session_start();
         ob_start();
          require_once ‘meekrodb.2.2.class.php’;
         DB::$user = ‘root’;
         DB::$password = ‘root’;
         DB::$dbName = ‘MicroBlog’;
         $username = $_POST['username'];
         $password = $_POST['password'];
         $result = DB::queryFirstRow("SELECT * FROM Credentials where username = %s",
$username);
         $hash = $result['password'];
          if (password_verify($password, $hash)) {
                 $_SESSION['loggedin'] = 1;
                 header('Location: http://localhost:8888/');
         } else {
                 echo "Login failed";
         }
?>
```

Like before, there's nothing wildly new here. We've introduced DB::queryFirstRow. However, this is pretty transparent with what it does, and retrieves the first row from the database. This is slightly quicker than DB::query, and produces an object which we don't have to iterate over.

Another concept which has been introduced is the password_verify() function. This allows us to compare an unhashed and hashed password, and to see if it is a match. If it is, we then set $_SESSION['loggedin'] to a value (in this case, one) and then redirect to the homepage, where we are then able to log out and to post new tweets.

Now, test it out by seeing if you can log in and create a new post!



6.5 Logging Out

Logging out is easy. Create a new file with a name of 'logout.php'. Inside, add the following lines.

```
<?php
session_start();
session_destroy();
header('Location: http://localhost:8888/');
?>
```

The session which previously contained the activated 'loggedin' variable has now been destroyed, effectively logging us out. Once destroyed, the user is then redirected to the homepage.

7. Conclusion And Further Reading

I hope you enjoyed this short introduction to the PHP programming language.

I know it was a whistle-stop tour. I know it was brief. I know it was hurried. Despite this, I hope I communicated all my points clearly. If you are dissatisfied, or have further questions, please contact me. My email address is me@mat-thewhughes.co.uk, and I can also be reached on Twitter. My username is @matthewhughes and I am always happy to respond to reader feedback.

Before we wrap things up, I want to address a couple more things.

Firstly, the PHP programming language is an awesome, powerful tool. If you are eager to do some further reading, please check out PHP The Right Way and Programming PHP by O'Reilly. Both are amazing resources, and are highly recommended.

My colleagues at MakeUseOf have also previously recommended some free resources to learn PHP.

Finally, the source code to this project is available on my personal Github. You can find the repository here. If you're unsure of what Git is, check out this article about Git version control. If you don't know how to use Github, check out Git Real by CodeSchool. It will bring you up to speed pretty quickly, and it's free.

Thank you for your time.

Matthew Hughes

Guide Published: March 2014