Class 1: PHP for site modularity

References

PHP and MySQL for Dynamic Web Sites (4th Ed.) by Larry Ullman
Head First PHP & My SQL by Lynn Beighley and Michael Morrison
Programming PHP (3rd Ed.) by Kevin Tatroe, Peter MacIntyre and Rasmus Lerdorf
http://devzone.zend.com/6/php-101-php-for-the-absolute-beginner/
http://www.youtube.com/watch?v=XJYtIZ2Aqqo
http://www.youtube.com/watch?v=gJbPQ5Qjbdk

Class 1 Homework

Read: Chapters 1, 2 and 3 of PHP and MySQL for Dynamic Web Sites
AND The whole of this document

Experiment with PHP includes in order to ensure that you understand how they work. The best form of the include function is probably:

```
<?php include($_SERVER['DOCUMENT_ROOT'].'/includes/file.php'); ?>
```

Where "includes" is the folder where your include file (file.php) is saved and that folder is in document root. Alternatively, you could use a relative link.

Simply place this within your markup where you want the content in *file*.php to appear.
Try some of the other PHP techniques such as building a contextual navigation include.

Remember, this will only work once uploaded to the web server. It will not work on your local computer, so get used to testing your websites live on the server.

Continue the work on the site makeover project.

Introduction to PHP

Static HTML

So far, all the websites we have built are “static”. In other words, the content we see in the browser window is rendered directly from the HTML markup, using the CSS rules for presentation. It doesn't matter how many times we view such pages, they do not change because the web server is always serving the same html and css files (unless we change them manually).

That's just boring!

Yes, it is. Websites should respond to the user and reflect temporal changes. For example, the copyright date in the page footer should always update to the current year, avoiding the necessity for changing it manually. Webpages should be self-maintaining.

Good news

The good news is that we can do all of this and more using a rather wonderful *server-side* scripting language called PHP.

Isn't JavaScript our scripting language of choice?

Well, JavaScript is a *client-side* scripting language, which means it can only act on a webpage once it has been served and downloaded to the local client (browser). Whereas, PHP runs on the server and the instructions we make are executed before the webpage is served. The beauty of this is that we can actually control how the page is built each time it is requested.

Crazy cool!

Sure is. PHP makes the building of even basic websites more efficient and makes them much easier to maintain.

Includes

Don't repeat yourself (DRY)

You will have noticed that there is quite a lot of repetition in your static websites. For example, the primary navigation is repeated on every page of your site, so to is the branding and the footer. Maintaining such sites is problematic because every instance of a primary navigation link will need to be modified if the site changes. Wouldn't it be great if the navigation could be changed just once in a single file and the whole site updates automatically? PHP can do this.

Modular websites

We can break our websites down into logical components, put each component into a separate file and then assemble each page as it is requested. We do this using server-side includes and we use PHP to make the whole thing work.

How does the whole “include” thing work?

You are familiar with the concept of linking a CSS file to a HTML file — in effect, the CSS is imported into the head of the HTML document. The same thing happens with includes except that the contents of the included file actually appears as part of the HTML when it reaches the browser. All we need to do is get PHP to tell the server which bits of HTML should be added to which part of the main file.

OK, what does this look like?

Well, if you have a homepage called index.html, the first thing we need to do is change its name to index.php so that the server knows it needs to be dealt with in a particular way (parsed). Once we've done that, we can add PHP to the file, knowing it will be properly processed.

Say we have saved the navigation part of our webpage (just plain html — nothing fancy) in a file called navigation.inc.php, we can include that file using the following bit of php:

```
<?php include ('navigation.inc.php'); ?>
```

The include function does all the work. The name of the include file can be anything (stuff.txt would work just as well), but the name I'm using here is designed to tell me what I need to know about its contents and how it should be used.

Yep, it's really that simple…

…well, there are a few more things you need to know before you become an include ninja, but that's the basics of it.

OK, what more do I need to know?

Well, working with files on a server using PHP is a little different from working with files on a server using HTML. PHP is a relatively simple yet very powerful scripting language and unlike HTML, it can be used to access files **above the web root**. This superpower means that file paths need to be dealt with differently.

So there's a catch?

Yes but it's not a big deal. Here's the thing — with HTML, we can point to any folder below web root using the slash character like this **/folder**. If we try the same technique with PHP, it doesn't work because PHP can see a few levels above web root and so we need to add those other folder levels to our path.

How can I find out what those folder are called?

Usually, you will need to contact your web host because the upper levels may not even be visible using FTP. It usually looks something like this:
/home/account_name/public_html/folder/ (where public_html is the web root). But in practice, this will vary from one host to another and depends how the server is configured.

This is all sounding rather complicated

Yes, it is but don't despair because PHP can rescue us from this uncertainty. It stores the path from the server account root to the web root in a *superglobal variable* (see, I told you PHP has super powers). The variable is called *$_SERVER['DOCUMENT_ROOT']* and it can be used in place of the bit of the path we don't know.

That is really useful, how do we use it

Well, in order to provide a full path to the include function, we need to add the variable to the bit of the path we do know. In PHP, we can build a compound string from variable and bits of text using the dot (period) character. This is technically known as "concatenation". Let's assume we have a file called **footer.inc.php** and it's in a folder called includes just below web root. We can build a path to that file from anywhere in our website using this code:

```
$_SERVER['DOCUMENT_ROOT'] . '/includes/footer.inc.php'
```

The beauty of this approach is that we never need to know what the bit of the path above web root is called and (even better) if we ever move our site to a different host that uses a different configuration, our site will not break because the super global variable always contains the correct information.

OK, I'm beginning to like this PHP business

PHP is written with ease of use in mind and with a little practice, you should find it quite easy to build modular websites.

So, show me the full include thingy

OK, so we want to include the **footer.inc.php** file in our homepage. All we need to do is use the include function as we did before, but use the concatenated string in place of the filename like so:

```
<?php include ($_SERVER['DOCUMENT_ROOT'] . '/includes/footer.inc.php'); ?>
```

And that's all there is to it?

Yes, of course you need to make sure that the syntax is correct; otherwise it just won't work, so pay special attention to all the quotes, semi-colons and parenthesis. Take a look at Chris Coyier's explanation for more background.

That method makes the path quite verbose, can we shorten it?

Well, we can't really shorten it but if a file contains a number of includes, it may be more efficient to set the include path before any include statements are made like so:

```
<?php set_include_path($_SERVER['DOCUMENT_ROOT']); ?>
```

This means we can drop the path above web root when we make subsequent include statements, like so:

```
<?php include('includes/footer.inc.php'); ?>
```

Notice that there is no leading slash in the path because the set_include_path function has already made sure that we will be starting at web root.

Dates

Great stuff, now show me the date thing

Making dates dynamic with PHP is even easier than building modular websites with the include function. We're going to do this using the *date* function (what else would it be called?).

What is the date?

The thing to remember about the PHP date function is that it always returns server time. So, if your server is in the UK, the returned time will be the current time in the UK. This differs from JavaScript time, which always returns the local time because it runs in the client browser. As long as you are aware of this, it's safe to proceed.

Hang on, that could cause complications

Yes, it might but as long as you know what's going on, you can work with it. For example, some applications built on PHP allow the user to set their local timezone. Once this is done, PHP can work out their local time and blog posts etc. can be tagged correctly.
There are many situations where it doesn't really matter. For example, the copyright year in our footer does not need to be switched on the stroke of midnight, as long as it gets updated around that time, that's good enough. The important thing is that we didn't need to do it manually — no one wants to interupt their New Year celebrations to update a copyright notice!

OK, let's do it

Currently, our non-dynamic HTML footer looks like this:

```
<footer>
Copyright &copy; 2013 David Watson. All Rights Reserved.
</footer>
```

The year, “2013” is just static text and will remain unchanged, irrespective of the actual year, unless we change it manually.

Let's replace the year text with a little bit of PHP:

```
<footer>
Copyright &copy; <?php echo date('Y'); ?> David Watson. All
Rights Reserved.
</footer>
```

Each time the page is requested, the date function is run, it gets the current date and time from the server in the form of a string of digits and then formats it as requested. In this case, the format string upper-case Y tells the function to return the full, 4-digit year number. The echo function then tells the PHP parser to print the year number as plain text. The server then sends the file out to the client.

It's just like magic

Yes, it is and the whole process is completely hidden from the user. If try to view the source of a PHP file, you won't see any PHP code; you only see the HTML that has been generated.

Can I do other stuff with the date function?

Yes, you can. The date can be formatted in many different ways. For example, this bit of PHP:

```
<?php echo date('jS F Y H:i:s'); ?>
```

Will return a date/time that looks something like this:

12th November 2014 10:23:55

How can I find out how the formatting works?

All PHP functions are described in detail on the PHP manual website. The page for the date function includes a description of all the formatting characters — you'll be amazed at the number of options.

Variables

How do variables work in PHP?

Using variables in PHP is very simple. We don't have to worry about data types or anything like that because PHP is smart enough to deal with all that stuff behind the scenes.

Just remind me what a variable is

You can think of a variable as being a named box into which we can place (assign) just about any kind of data like a number or a line of text (string). Variables can also be used to store complex data like arrays, but we'll keep things simple for now.

Do PHP variables have special names?

Yes, PHP variables always begin with the dollar sign ($), for example `$name`. Other than that, names can contain any combination of letters, numbers and the underscore character. The underscore character is often used as a separator like `$current_date`.

There's one other thing about variable names you should be aware of; they are case sensitive, so `$name` and `$Name` are different variables. It's best practice to name all your variables using lower-case characters to avoid confusion.

OK, so how do we declare a variable and assign a value to it?

In PHP, declaring a variable and assigning a value to it is all part of the same process and can be done with a single statement:

```
$my_name = "David";
```

The statement above declares a variable called *$my_name* and uses the equals sign (the assignment operator) to give it a value; in this case a bit of text, referred to as a string.
Just like all statements in PHP, it is ended with the semi-colon character.

OK, show me some more variables being assigned

The following script assigns 3 variables, a string, a number and a value returned by a function. Pay special attention to the syntax in each case:

```
<?php
$building = "Seattle Space Needle";
$height = 184;
$today = date("l");
?>
```

Notice that string values are quoted but numbers are not. Notice also that you don't have to assign an explicit value; you can assign a value returned by a function. In the example above, the date function is used to generate the day name (e.g. “Monday”) and that is the value assigned to `$today`.

Wow, that's really cool

Yes, it is and it's the ability to store dynamically generated values in variables that give scripting languages like PHP their power.

What can I do with variables once I've got them?

Well, you can do many things; you can compare them, use them to make decisions, and get them to interact with each other in various ways but one of the most basic things you need to know is how to print them:

```
<?php echo $today; ?>
```

As you can see, printing variables is easy, we just use the echo function that we met earlier, and pass it the name of our variable.

What if I want to add more text to that echo statement?

Typically, you might want to print something like “Today is Monday”. To do that, you would use the following statement:

```
<?php echo "Today is $today"; ?>
```

Ah, so the quotes are important?

Yes, the quotes are very important and they must be double quotes and not single quotes because variables are evaluated when placed between double quotes and not evaluated when placed between single quotes.

Could you just clarify that?

Yes, the following statement:

```
echo "Today is $today";
```

will print: *Today is Monday*

whereas this statement:

```
echo 'Today is $today';
```

will print: *Today is $today*

That's a bit confusing isn't it?

Maybe a little, but there is a good reason for these two methods of quoting strings. Single quotes are more efficient for PHP because there's no extra work involved, all strings are explicit. Whereas double quotes means that PHP has to look out for and evaluate any variables that happen to be in a string.

There's no rule about this, you need to use your own judgement when using quotes to decide which is best in any given situation.

David Watson – January 2014